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

Alex Graves, Santiago Fernandez, Faustino Gomez
Jürgen Schmidhuber

Presented by Aleksei Kalinov

16 February 2018

Talk Outline

1. Temporal Classification Problem
2. Connectionist Temporal Classification
    a. Motivation
    b. Architecture Description
    c. Training Procedure
    d. Results

Talk Outline

1. Temporal Classification Problem
2. Connectionist Temporal Classification
    a. Motivation
    b. Architecture Description
    c. Training Procedure
    d. Results

Temporal Classification

Informally: Annotate real-valued, possibly noisy, input streams with strings of discrete labels, e.g. given a video stream as an input provide text captions as an output.

Temporal Classification

Informally: Annotate real-valued, possibly noisy, input streams with strings of discrete labels, e.g. given a video stream as an input provide text captions as an output.

Formally: Let $\mathbf{X} = (\mathbb{R}^m)^*$ — space of input sequences,

$\mathbf{Z} = \mathbf{L}^*$ — space of output sequences, where L is a finite alphabet,

$\mathbf{S} \subset \mathbf{X} \times \mathbf{Z}$, s.t. $\forall\ (\mathbf{x},\mathbf{z}) \in \mathbf{S}\ \ len(\mathbf{x}) \geq len(\mathbf{z})$ — training set.

Using $\mathbf{S}$ find a temporal classifier $h : \mathbf{X} \mapsto \mathbf{Z}$ that minimizes task-dependent error measure.

Label error rate

$$LER(h, \mathbf{S}') = 1/\mathrm{R}\ (\Sigma_{(\mathbf{x}, \mathbf{z}) \in \mathbf{S}'} ED(h(\mathbf{x}), \mathbf{z})),$$

where R *is the total number of target labels in* $\mathbf{S}'$
and $ED(\mathbf{x}, \mathbf{y})$ *is the edit distance between* $\mathbf{x}$ *and* $\mathbf{y}$.

It is a natural quantity to measure if we want to minimize the rate of transcription mistakes (think, speech recognition).

Talk Outline

1. Temporal Classification Problem
2. Connectionist Temporal Classification
    a. Motivation
    b. Architecture Description
    c. Training Procedure
    d. Results

Motivation for CTC

Why can we not just use old but gold FFNN or pure RNN?

- Outputs in these models are independent of each other, leading to necessity of training data segmentation and post-processing during inference.

What about HMM that were utilized to solve this problem?

- Require task specific knowledge and dependency assumptions.

Architecture

Informally: Use RNN, add a softmax layer on top with $|\mathbf{L}| + 1$ output (the additional one is for a blank output), treat these as probabilities.

Formally: Let $\mathbf{L}' = \mathbf{L} \cup \{blank\}$. Then define $\mathcal{N}: (\mathbb{R}^{\mathbf{m}})^{\mathbf{T}} \mapsto (\mathbb{R}^{|\mathbf{L}|+1})^{\mathbf{T}}$ — a continuous map computed by recurrent network with softmax outputs. If $\mathbf{y} = \mathcal{N}(\mathbf{x})$, then y^t_k — activation of k^{th} output at time t.

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi_t}, \ \forall \pi \in L'^T$$

Architecture

Define a map B: $\mathbf{L'^{T}} \mapsto \mathbf{L'^{\leq T}}$ that removes consecutive characters and blanks in the original sequence. This map lets us calculate the conditional probability of target sequences:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x})$$

The mode of the output is our prediction:

$$h(\mathbf{x}) = \arg \max_{\mathbf{l} \in L^{\leq T}} p(\mathbf{l}|\mathbf{x})$$

Computing distribution

Iterating over every possible path to compute a conditional probability is a no-go. But if we know the answer for a prefix of $\mathbf{l}$, we can extend it to one more character. It's dynamic programming!

Define the following forward variable:

$$\alpha_t(s) \stackrel{\text{def}}{=} \sum_{\substack{\pi \in N^T: \\ \mathcal{B}(\pi_{1:t}) = \mathbf{l}_{1:s}}} \prod_{t'=1}^{t} y_{\pi_{t'}}^{t'}$$

Note: These variables are actually defined on a sequence $\mathbf{l}'$, which is a modified sequence $\mathbf{l}$ with blanks inserted between each character.

Forward pass of DP

1. Initialize variables:

$$\alpha_1(1) = y_b^1$$
$$\alpha_1(2) = y_{\mathbf{l}_1}^1$$
$$\alpha_1(s) = 0, \ \forall s > 2$$

2. Perform the step:

$$\bar{\alpha}_t(s) \overset{\text{def}}{=} \alpha_{t-1}(s) + \alpha_{t-1}(s-1)$$

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y_{\mathbf{l}'_s}^t & \text{if } \mathbf{l}'_s = b \text{ or } \mathbf{l}'_{s-2} = \mathbf{l}'_s \\ \big(\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)\big) y_{\mathbf{l}'_s}^t & \text{otherwise} \end{cases}$$

Forward pass of DP

3. Compute the answer: $$p(\mathbf{l}|\mathbf{x}) = \alpha_T(|\mathbf{l}'|) + \alpha_T(|\mathbf{l}'| - 1)$$

The same procedure could be done backwards with suffixes.

$$\beta_t(s) \stackrel{\text{def}}{=} \sum_{\substack{\pi \in N^T: \\ \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}|}}} \prod_{t'=t}^{T} y^{t'}_{\pi_{t'}}$$

Architecture

Cool, but how do compute the actual output?

1. The lazy approach: Find the best path $\boldsymbol{\pi}^*$ in $\mathbf{L}'$ and compute $B(\boldsymbol{\pi}^*)$.
   a. Fast to compute: just take the argmax at each activation unit.
   b. Not guaranteed to produce the best result.

2. The lazier approach: Use dynamic programming on prefixes with already computed forward variables.
   a. Always finds a optimal value.
   b. Search space can grow exponentially. Reduced by heuristic of splitting sequence in parts.

Training

Adopt **maximum likelihood** approach trying to maximize the probabilities of correct classes in the training set:

$$O^{ML}(S, \mathcal{N}_w) = -\sum_{(\mathbf{x},\mathbf{z})\in S} ln\big(p(\mathbf{z}|\mathbf{x})\big)$$

We can use forward and backward variables to compute it! Let's firstly observe:

$$\alpha_t(s)\beta_t(s) = \sum_{\substack{\pi\in\mathcal{B}^{-1}(\mathbf{l}):\\ \pi_t=\mathbf{l}'_s}} y^t_{\mathbf{l}'_s} \prod_{t=1}^{T} y^t_{\pi_t}$$

Training

By summation and rearrangement we get:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{s=1}^{|\mathbf{l}'|} \frac{\alpha_t(s)\beta_t(s)}{y^t_{\mathbf{l}'_s}}$$

Now finding gradient is easy. Define $lab(\mathbf{l}, k) = \{s : \mathbf{l}'_s = k\}$.

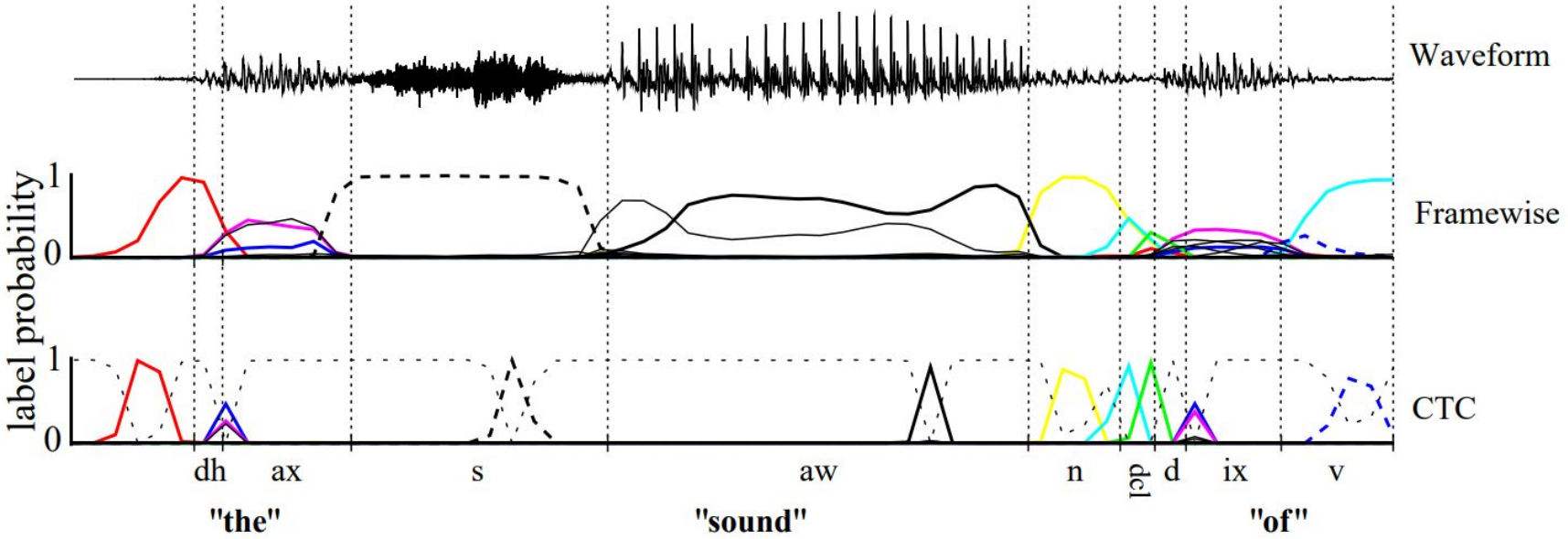
$$\frac{\partial p(\mathbf{l}|\mathbf{x})}{\partial y^t_k} = \frac{1}{{y^t_k}^2} \sum_{s \in lab(\mathbf{l},k)} \alpha_t(s)\beta_t(s)$$

Results

Table 1. **Label Error Rate (LER) on TIMIT**. CTC and hybrid results are means over 5 runs, ± standard error. All differences were significant ($p < 0.01$), except between weighted error BLSTM/HMM and CTC (best path).

System	LER
Context-independent HMM	38.85 %
Context-dependent HMM	35.21 %
BLSTM/HMM	33.84 ± 0.06 %
Weighted error BLSTM/HMM	31.57 ± 0.06 %
CTC (best path)	31.47 ± 0.21 %
CTC (prefix search)	30.51 ± 0.19 %

Results


Waveform
Framewise
CTC
label probability
1
0
1
0
dh
ax
s
aw
n
dcl
d
ix
v
"the"
"sound"
"of"

Conclusions

1. Temporal Classification may be tricky and can require a substantial number of task adjustments.
2. Connectionist Temporal Classification reduces the burden by eliminating the need to segment the training data.
3. CTC comes with the price of not being able to segment the inference data.

Be the boss, use CTC loss.

Alex Graves et al. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. https://www.cs.toronto.edu/~graves/icml_2006.pdf (Accessed 16 Feb 2018)