Connectionist Temporal Classification: Labelling Unsegmented Sequences with Recurrent Neural Networks

Research Project Report – Probabilistic Graphical Models course

ALEX AUVOLAT
Department of Computer Science
École Normale Supérieure de Paris
`alex.auvolat@ens.fr`

THOMAS MESNARD [0]
Department of Computer Science
École Normale Supérieure de Paris
`thomas.mesnard@ens.fr`

Abstract

Many real-world sequence learning tasks require the prediction of sequences of labels from noisy, unsegmented input data. Recurrent neural networks (RNNs) are powerful sequence learners that would seem well suited to such tasks. However, because they require pre-segmented training data, and post-processing to transform their outputs into label sequences, they cannot be applied directly. Connectionist Temporal Classification is a method for training RNNs to label unsegmented sequences directly, thereby solving both problems.

Project Advisor: Guillaume Obozinski
Course: *"Probabilistic Graphical Models"*, by F. Bach, S. Lacoste-Julien, G. Obozinski
For the Mathématiques, Vision, Apprentissage (MVA) Master 2 at ENS de Cachan.

[0] If needed, see online at `https://github.com/thomasmesnard/CTC-LSTM` for the implementation, open-sourced under the Apache license.

1 Model

Connectionist Temporal Classification (CTC) introduces a new cost function for training recurrent neural networks to label unsegmented sequences directly. To use this cost function, we introduce an additional blank symbol in the possible labels that the recurrent neural networks can output. We remind that the output layer of the recurrent neural networks correspond to probabilities over all possible labels.

This additional blank symbol give freedom to the RNN to give the label for a section of input at any moment, and especially when it is sure of its answer, simply by outputting the blank label the rest of the time.

The blank label also allows the network to give a strong probability to the correct label at a very localized point in time, whereas in a classical setup we observe distributed probabilities over all labels at each time step. An example of both behaviors can be observed on speech data in Figure 1.

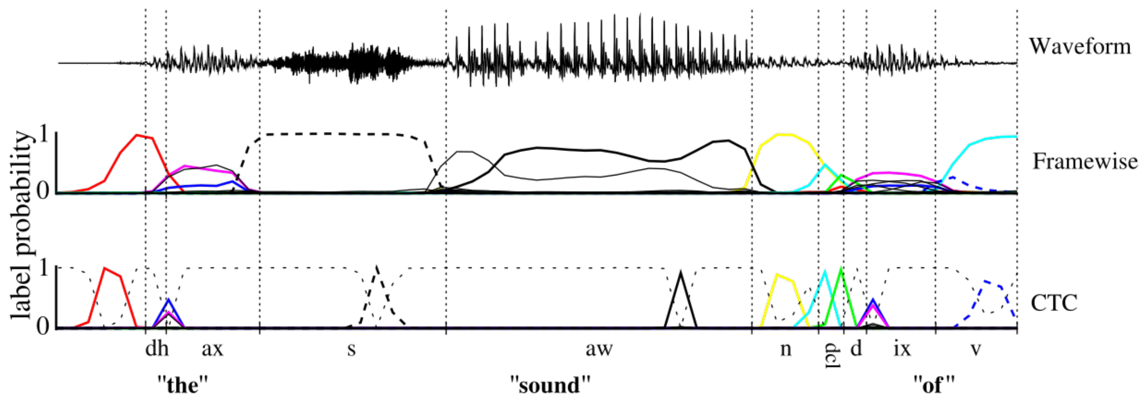

Figure 1: Output of classic framewise phoneme classification and RNN trained with CTC

As the output sequence is shorter than the input sequence, there are many possible alignments with the correct label sequence. We want the recurrent neural network to learn one of these correct alignments on its own. Using dynamic programing to sum over all the possible alignments, CTC provides gradients for the backpropagation phase to train the RNN to learn a good alignment.

In the basic setup shown in Figure 2, we use stacked bidirectional recurrent neural networks. The CTC brick takes as inputs the output of the last bidirectional recurrent neural network as well as the target label sequence, and calculates the cost of the label. When differentiated, the CTC brick gives back gradients to train the RNN and learn a good alignment.

We use the following notation:

- y_k^t: output at time t for symbol k
- l: the label sequence for which we want to calculate a cost
- l': the same label sequence but with blanks added between each letters

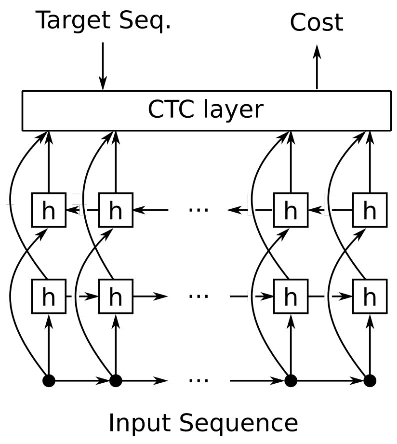

Figure 2: Simple bidirectional RNN model with CTC cost layer

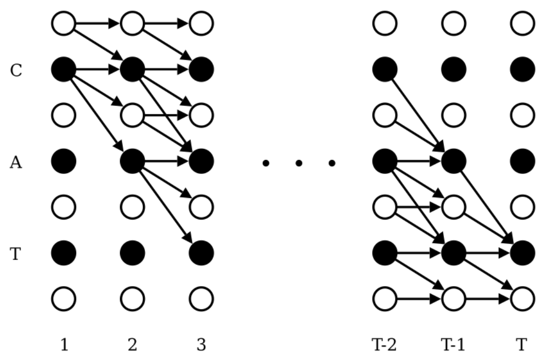

Figure 3: Computation graph for $\alpha_t(s)$ (corresponds to an unrolling of the automaton that recognizes $\mathcal{B}^{-1}(l)$)

1.1 First Definition

CTC is a dynamic programming algorithm that calculates the following variable:

$$\alpha_t(s) = \sum_{\substack{\pi \in N^T: \\ \mathcal{B}(\pi_{1:t}) = l'_{1:s}}} \prod_{t'=1}^{t} y^{t'}_{\pi_{t'}} \tag{1}$$

Where:

- $\mathcal{B}$ is the transform that removes all blanks and duplicates
- $\pi \in N^T : \mathcal{B}(\pi_{1:t}) = l'_{1:s}$ corresponds to all possible paths among all possible labels from time step 1 to t which give the s first correct labels after we apply the $\mathcal{B}$ transform.
- $y^{t'}$ corresponds to the output of the RNN at time t'

We can see that $\alpha_t(s)$ corresponds to the sum, over all possible paths between time step 1 and t that will give the s first correct labels after removing all banks and duplicates, of the product of the probabilities output by the RNN on these paths.

The formulation described in Equation 1 is equivalent to the unrolled automaton presented in Figure 3. To allow for blanks in the output paths, we consider a modified label sequence l', with blanks added to the beginning and the end and inserted between every pair of labels. The length of l' is therefore $2|l|+1$. This allows transition between blank and non-blank labels, and also those between any pair of distinct non-blank labels.

1.2 Recursive Definition

The definition of $\alpha_t(s)$ given above enables us to understand what this function calculates, but unfortunately it is not practical to compute. We will now see a recursive definition of the $\alpha_t(s)$ which provides a dynamic programming algorithm for our problem. This calculation is illustrated in Figure 3.

We first have to initiate α_1 with at first a blank label, then the first correct label and finally just zeros. Indeed, it is impossible that more than one correct label derive from only one output of the RNN:

$$
\begin{aligned}
\alpha_1(1) &= y_b^1 \\
\alpha_1(2) &= y_{l_1}^1 \\
\alpha_1(s) &= 0, \forall s > 2
\end{aligned}
$$

We then define the recurrence relations:

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

With:

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

Finally, we have:

$$p(l|x) = \alpha_T(|l'|) + \alpha_T(|l'|-1)$$

Which give us, given an input, the probability of having a particular sequence.

1.3 Analysis of the Gradient Provided by CTC

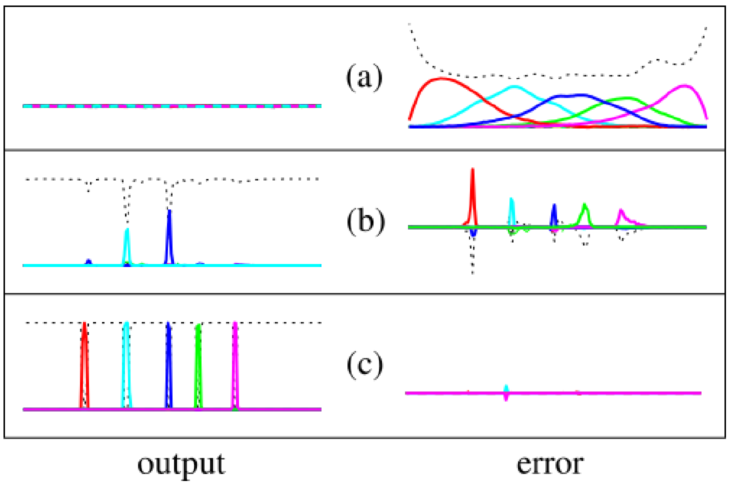

Figure 4: Evolution of the CTC error signal

In Figure 4, the left column shows the output activations for the same sequence at various stages of training. The dashed line corresponds to the blank unit. The right column shows the corresponding error signals, i.e. the gradient which is sent back to the RNN.

On the early stage of the training (a), the network does not make predictions because it has small random weights, the error is only determined by the target sequence and is localized only very vaguely.

During the training (b), the network begins to make predictions and the error localizes around them.

At the end of the training (c), the network strongly predicts the correct labels at very specific times, and the rest of the time outputs the blank label. The error signal becomes null since there is no error.

2 Experiments

We made an implementation of the model using Theano and Blocks which is a Theano framework for building and training neural networks. The code is available at the following address: `https://github.com/thomasmesnard/CTC-LSTM`.

As Theano provides an automated symbolic differentiation tool, we implemented the forward recursion for $\alpha_t(s)$ and let Theano derive the backward calculation. To avoid numerical underflow, two methods can be applied, which we both implemented:

- Normalize at each time step t the α_t: $C_t = \sum_s \alpha_t(s) \quad \hat{\alpha}_t(s) = \frac{\alpha_t(s)}{C_t}$
- Do our calculations in the logarithmic domain

2.1 Toy Dataset

We first tried our implementation on a simple task. Our dataset is composed of the following sequences, where each digit is randomly repeated a random number of times.

$$
\begin{aligned}
1^*2^*3^*4^*5^* &\rightarrow 1 \\
1^*2^*3^*2^*1^* &\rightarrow 2 \\
5^*4^*3^*2^*1^* &\rightarrow 3 \\
5^*4^*3^*4^*5^* &\rightarrow 4
\end{aligned}
$$

The network has to learn the previous association. The two first sequences have an overlap on the three first digits. The network has to wait until the fourth digit to know if it is either a 1 or a 2. We can see here the importance of the blank label. Indeed, the network will have to outputs the blank label until the fourth label is presented to be able to give the correct label. The same reasoning can be applied for the two last sequences.

We have two versions of the toy dataset: on the first version each piece of the sequence is perfectly identifiable as each character appears at least once (the input sequence is therefore at least 5 times as long as the target sequence). On the second version some characters may be omitted in the input, adding stochasticity to the task. CTC provides very good results on the two tasks (shown in Table 1 and Table 2).

Results	**train**	**valid**
Output sequence length	5 – 50	5 – 50
Error rate	0	0
Mean edit distance	0	0
Errors per character	0	0

Table 1: Performances of CTC on our toy dataset, with perfect input sequences

Results	**train**	**valid**
Output sequence length	5 – 20	5 – 20
Error rate	0.62	0.63
Mean edit distance	1.0	1.1
Errors per character	0.08	0.09

Table 2: Performances of CTC on our toy dataset, with imperfect input sequences

We observe that the model performs flawlessly on the first task. On the other task, the error rate is not as low as expected but we remind that it corresponds to the rate on which the network is able to find the complete target sequence. A more meaningful measure is the error rate per character which is only 9% on the validation set. We deduce from these two results that the model is able to perfectly learn the rule that maps the input sequences to the targets, but even though, it cannot achieve perfect results on the second task as information is sometimes missing in the input. Both implementations converge in less than 1000 time steps.

2.2 TIMIT

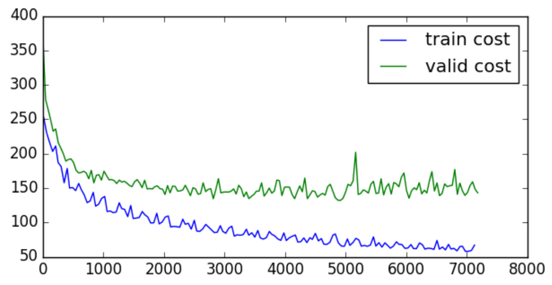

Figure 5: Training cost of the CTC model on TIMIT

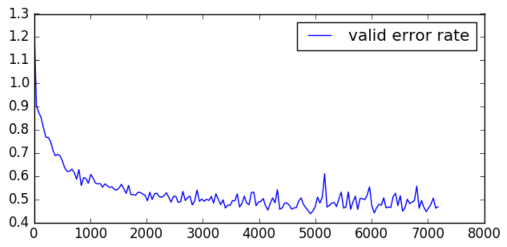

Figure 6: Error rate of the CTC model on TIMIT

Convolution layers	**Filters**	**Filter size**	**Stride**	**Skip**	**Normalize**
Layer 1	20	200	10	Yes	Yes
Layer 2	20	200	10	Yes	Yes
Layer 3	20	30	2	Yes	Yes
Layer 4	100	20	2	No	Yes
Recurrent layers	**Size**	**Type**	**Bidirectional**	**Skip**	**Normalize**
Layer 1	50	LSTM	Yes	Yes	No
Layer 2	50	LSTM	Yes	Yes	No

Table 3: Structure of the Deep Neural Network for the TIMIT dataset

We then tried on the classical TIMIT dataset. It is a raw speech signal dataset of 4120 sentences labelled by phonemes or by words. The average audio length is 50 000 samples and the average sentence length is 38 phonemes.

To avoid hand-crafted feature extraction on the speech signal, we use convolution layers before the bidirectional LSTM layers. We then use bidirectional LSTM layers, and of course the CTC cost function. We use noise and normalization on intermediate layers for regularization. The structure of our model is described in Table 3.

We were able to attain a 50% phoneme error rate on the validation set after about 150 epochs of training (see Figure 6). This result is not as good as the 30% achieved by the original paper [1], however it is an extremely good result for the model as we do not use hand-crafted preprocessing on the data. This shows that the convolution layers are able to learn the filters necessary for audio processing on speech by themselves.

3 Conclusion

CTC is a powerful cost function for training RNNs on unsegmented data, now largely used in major commercial applications. We were able to get very good results using CTC on a toy dataset. We proposed a way of processing speech data with convolutional neural networks and were able to train a convnet-LSTM-CTC model with satisfactory results on TIMIT.

References

[1] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.