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

Alex Graves[1] ALEX@IDSIA.CH
Santiago Fernández[1] SANTIAGO@IDSIA.CH
Faustino Gomez[1] TINO@IDSIA.CH
Jürgen Schmidhuber[1,2] JUERGEN@IDSIA.CH

[1] Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Galleria 2, 6928 Manno-Lugano, Switzerland
[2] Technische Universität München (TUM), Boltzmannstr. 3, 85748 Garching, Munich, Germany

Abstract

Many real-world sequence learning tasks require the prediction of sequences of labels from noisy, unsegmented input data. In speech recognition, for example, an acoustic signal is transcribed into words or sub-word units. 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, their applicability has so far been limited. This paper presents a novel method for training RNNs to label unsegmented sequences directly, thereby solving both problems. An experiment on the TIMIT speech corpus demonstrates its advantages over both a baseline HMM and a hybrid HMM-RNN.

1. Introduction

Labelling unsegmented sequence data is a ubiquitous problem in real-world sequence learning. It is particularly common in perceptual tasks (e.g. handwriting recognition, speech recognition, gesture recognition) where noisy, real-valued input streams are annotated with strings of discrete labels, such as letters or words.

Currently, graphical models such as hidden Markov Models (HMMs; Rabiner, 1989), conditional random fields (CRFs; Lafferty et al., 2001) and their variants, are the predominant framework for sequence labelling. While these approaches have proved successful for many problems, they have several drawbacks: (1) they usually require a significant amount of task specific knowledge, e.g. to design the state models for HMMs, or choose the input features for CRFs; (2) they require explicit (and often questionable) dependency assumptions to make inference tractable, e.g. the assumption that observations are independent for HMMs; (3) for standard HMMs, training is generative, even though sequence labelling is discriminative.

Recurrent neural networks (RNNs), on the other hand, require no prior knowledge of the data, beyond the choice of input and output representation. They can be trained discriminatively, and their internal state provides a powerful, general mechanism for modelling time series. In addition, they tend to be robust to temporal and spatial noise.

So far, however, it has not been possible to apply RNNs directly to sequence labelling. The problem is that the standard neural network objective functions are defined separately for each point in the training sequence; in other words, RNNs can only be trained to make a series of independent label classifications. This means that the training data must be pre-segmented, and that the network outputs must be post-processed to give the final label sequence.

At present, the most effective use of RNNs for sequence labelling is to combine them with HMMs in the so-called hybrid approach (Bourlard & Morgan, 1994; Bengio., 1999). Hybrid systems use HMMs to model the long-range sequential structure of the data, and neural nets to provide localised classifications. The HMM component is able to automatically segment the sequence during training, and to transform the network classifications into label sequences. However, as well as inheriting the aforementioned drawbacks of

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

HMMs, hybrid systems do not exploit the full potential of RNNs for sequence modelling.

This paper presents a novel method for labelling sequence data with RNNs that removes the need for presegmented training data and post-processed outputs, and models all aspects of the sequence within a single network architecture. The basic idea is to interpret the network outputs as a probability distribution over all possible label sequences, conditioned on a given input sequence. Given this distribution, an objective function can be derived that directly maximises the probabilities of the correct labellings. Since the objective function is differentiable, the network can then be trained with standard backpropagation through time (Werbos, 1990).

In what follows, we refer to the task of labelling unsegmented data sequences as *temporal classification* (Kadous, 2002), and to our use of RNNs for this purpose as *connectionist temporal classification* (CTC). By contrast, we refer to the independent labelling of each time-step, or frame, of the input sequence as *framewise classification*.

The next section provides the mathematical formalism for temporal classification, and defines the error measure used in this paper. Section 3 describes the output representation that allows RNNs to be used as temporal classifiers. Section 4 explains how CTC networks can be trained. Section 5 compares CTC to hybrid and HMM systems on the TIMIT speech corpus. Section 6 discusses some key differences between CTC and other temporal classifiers, giving directions for future work, and the paper concludes with section 7.

2. Temporal Classification

Let S be a set of training examples drawn from a fixed distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Z}}$. The input space $\mathcal{X} = (\mathbb{R}^m)^*$ is the set of all sequences of m dimensional real valued vectors. The target space $\mathcal{Z} = L^*$ is the set of all sequences over the (finite) alphabet L of labels. In general, we refer to elements of L^* as *label sequences* or *labellings*. Each example in S consists of a pair of sequences $(\mathbf{x}, \mathbf{z})$. The target sequence $\mathbf{z} = (z_1, z_2, ..., z_U)$ is at most as long as the input sequence $\mathbf{x} = (x_1, x_2, ..., x_T)$, i.e. $U \leq T$. Since the input and target sequences are not generally the same length, there is no *a priori* way of aligning them.

The aim is to use S to train a temporal classifier $h : \mathcal{X} \mapsto \mathcal{Z}$ to classify previously unseen input sequences in a way that minimises some task specific error measure.

2.1. Label Error Rate

In this paper, we are interested in the following error measure: given a test set $S' \subset \mathcal{D}_{\mathcal{X}\times\mathcal{Z}}$ disjoint from S, define the *label error rate* (LER) of a temporal classifier h as the mean normalised edit distance between its classifications and the targets on S', i.e.

$$LER(h, S') = \frac{1}{|S'|} \sum_{(\mathbf{x},\mathbf{z})\in S'} \frac{ED(h(\mathbf{x}), \mathbf{z})}{|\mathbf{z}|} \qquad (1)$$

where $ED(\mathbf{p}, \mathbf{q})$ is the edit distance between two sequences $\mathbf{p}$ and $\mathbf{q}$ — i.e. the minimum number of insertions, substitutions and deletions required to change $\mathbf{p}$ into $\mathbf{q}$.

This is a natural measure for tasks (such as speech or handwriting recognition) where the aim is to minimise the rate of transcription mistakes.

3. Connectionist Temporal Classification

This section describes the output representation that allows a recurrent neural network to be used for CTC. The crucial step is to transform the network outputs into a conditional probability distribution over label sequences. The network can then be used a classifier by selecting the most probable labelling for a given input sequence.

3.1. From Network Outputs to Labellings

A CTC network has a softmax output layer (Bridle, 1990) with one more unit than there are labels in L. The activations of the first $|L|$ units are interpreted as the probabilities of observing the corresponding labels at particular times. The activation of the extra unit is the probability of observing a 'blank', or no label. Together, these outputs define the probabilities of all possible ways of aligning all possible label sequences with the input sequence. The total probability of any one label sequence can then be found by summing the probabilities of its different alignments.

More formally, for an input sequence $\mathbf{x}$ of length T, define a recurrent neural network with m inputs, n outputs and weight vector w as a continuous map $\mathcal{N}_w : (\mathbb{R}^m)^T \mapsto (\mathbb{R}^n)^T$. Let $\mathbf{y} = \mathcal{N}_w(\mathbf{x})$ be the sequence of network outputs, and denote by y^t_k the activation of output unit k at time t. Then y^t_k is interpreted as the probability of observing label k at time t, which defines a distribution over the set L'^T of length T sequences over the alphabet $L' = L \cup \{blank\}$:

$$p(\pi|\mathbf{x}) = \prod_{t=1}^{T} y^t_{\pi_t}, \ \forall \pi \in L'^T. \qquad (2)$$

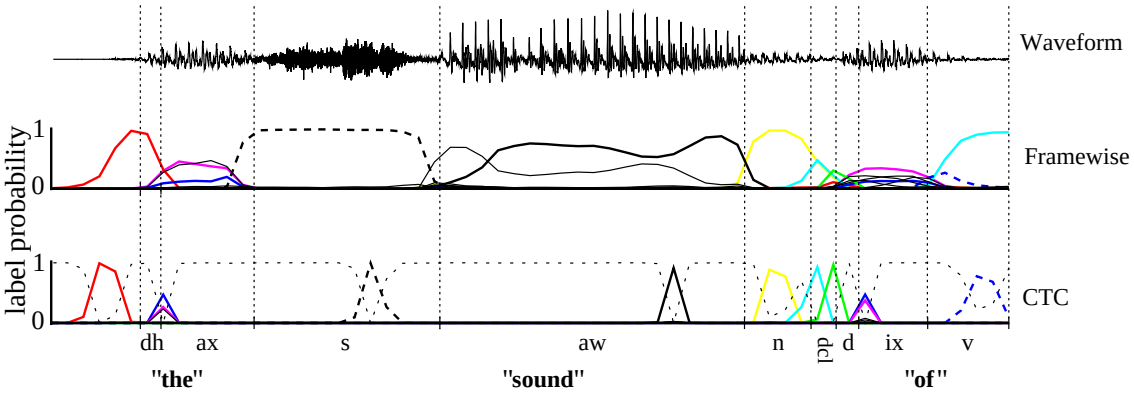

Figure 1. **Framewise and CTC networks classifying a speech signal**. The shaded lines are the output activations, corresponding to the probabilities of observing phonemes at particular times. The CTC network predicts only the sequence of phonemes (typically as a series of spikes, separated by 'blanks', or null predictions), while the framewise network attempts to align them with the manual segmentation (vertical lines). The framewise network receives an error for misaligning the segment boundaries, even if it predicts the correct phoneme (e.g. 'dh'). When one phoneme always occurs beside another (e.g. the closure 'dcl' with the stop 'd'), CTC tends to predict them together in a double spike. The choice of labelling can be read directly from the CTC outputs (follow the spikes), whereas the predictions of the framewise network must be post-processed before use.

From now on, we refer to the elements of L'^T as *paths*, and denote them π.

Implicit in (2) is the assumption that the network outputs at different times are conditionally independent, given the internal state of the network. This is ensured by requiring that no feedback connections exist from the output layer to itself or the network.

The next step is to define a many-to-one map $\mathcal{B} : L'^T \mapsto L^{\leq T}$, where $L^{\leq T}$ is the set of possible labellings (i.e. the set of sequences of length less than or equal to T over the original label alphabet L). We do this by simply removing all blanks and repeated labels from the paths (e.g. $\mathcal{B}(a - ab-) = \mathcal{B}(-aa - -abb) = aab$). Intuitively, this corresponds to outputting a new label when the network switches from predicting no label to predicting a label, or from predicting one label to another (c.f. the CTC outputs in figure 1). Finally, we use $\mathcal{B}$ to define the conditional probability of a given labelling $\mathbf{l} \in L^{\leq T}$ as the sum of the probabilities of all the paths corresponding to it:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{\pi \in \mathcal{B}^{-1}(\mathbf{l})} p(\pi|\mathbf{x}). \tag{3}$$

3.2. Constructing the Classifier

Given the above formulation, the output of the classifier should be the most probable labelling for the input sequence:

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

Using the terminology of HMMs, we refer to the task of finding this labelling as *decoding*. Unfortunately, we do not know of a general, tractable decoding algorithm for our system. However the following two approximate methods give good results in practice.

The first method (best path decoding) is based on the assumption that the most probable path will correspond to the most probable labelling:

$$h(\mathbf{x}) \approx \mathcal{B}(\pi^*) \tag{4}$$
$$\text{where } \pi^* = \arg\max_{\pi \in N^t} p(\pi|\mathbf{x}).$$

Best path decoding is trivial to compute, since π^* is just the concatenation of the most active outputs at every time-step. However it is not guaranteed to find the most probable labelling.

The second method (prefix search decoding) relies on the fact that, by modifying the forward-backward algorithm of section 4.1, we can efficiently calculate the probabilities of successive extensions of labelling prefixes (figure 2).

Given enough time, prefix search decoding always finds the most probable labelling. However, the maximum number of prefixes it must expand grows exponentially with the input sequence length. If the output distribution is sufficiently peaked around the mode, it will nonetheless finish in reasonable time. For the experiment in this paper though, a further heuristic was required to make its application feasible.

Observing that the outputs of a trained CTC network

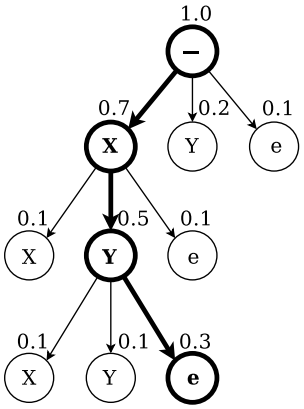

Figure 2. **Prefix search decoding on the label alphabet X,Y**. Each node either ends ('e') or extends the prefix at its parent node. The number above an extending node is the total probability of all labellings beginning with that prefix. The number above an end node is the probability of the single labelling ending at its parent. At every iteration the extensions of the most probable remaining prefix are explored. Search ends when a single labelling (here 'XY') is more probable than any remaining prefix.

tend to form a series of spikes separated by strongly predicted blanks (figure 1), we divide the output sequence into sections that are very likely to begin and end with a blank. We do this by choosing boundary points where the probability of observing a blank label is above a certain threshold. We then calculate the most probable labelling for each section individually and concatenate these to get the final classification.

In practice, prefix search works well with this heuristic, and generally outperforms best path decoding. However it does fail in some cases, e.g. if the same label is predicted weakly on both sides of a section boundary.

4. Training the Network

So far we have described an output representation that allows RNNs to be used for CTC. We now derive an objective function for training CTC networks with gradient descent.

The objective function is derived from the principle of maximum likelihood. That is, minimising it maximises the log likelihoods of the target labellings. Note that this is the same principle underlying the standard neural network objective functions (Bishop, 1995). Given the objective function, and its derivatives with respect to the network outputs, the weight gradients can be calculated with standard backpropagation through time. The network can then be trained with any of the gradient-based optimisation algorithms currently in use for neural networks (LeCun et al., 1998; Schraudolph, 2002).

We begin with an algorithm required for the maximum likelihood function.

4.1. The CTC Forward-Backward Algorithm

We require an efficient way of calculating the conditional probabilities $p(\mathbf{l}|\mathbf{x})$ of individual labellings. At first sight (3) suggests this will be problematic: the sum is over all paths corresponding to a given labelling, and in general there are very many of these.

Fortunately the problem can be solved with a dynamic programming algorithm, similar to the forward-backward algorithm for HMMs (Rabiner, 1989). The key idea is that the sum over paths corresponding to a labelling can be broken down into an iterative sum over paths corresponding to prefixes of that labelling. The iterations can then be efficiently computed with recursive *forward* and *backward* variables.

For some sequence $\mathbf{q}$ of length r, denote by $\mathbf{q}_{1:p}$ and $\mathbf{q}_{r-p:r}$ its first and last p symbols respectively. Then for a labelling $\mathbf{l}$, define the forward variable $\alpha_t(s)$ to be the total probability of $\mathbf{l}_{1:s}$ at time t. i.e.

$$\alpha_t(s) \stackrel{\text{def}}{=} \sum_{\substack{\pi \in N^T: \\ \mathcal{B}(\pi_{1:t})=\mathbf{l}_{1:s}}} \prod_{t'=1}^{t} y^{t'}_{\pi_{t'}}. \tag{5}$$

As we will see, $\alpha_t(s)$ can be calculated recursively from $\alpha_{t-1}(s)$ and $\alpha_{t-1}(s-1)$.

To allow for blanks in the output paths, we consider a modified label sequence $\mathbf{l}'$, with blanks added to the beginning and the end and inserted between every pair of labels. The length of $\mathbf{l}'$ is therefore $2|\mathbf{l}|+1$. In calculating the probabilities of prefixes of $\mathbf{l}'$ we allow all transitions between blank and non-blank labels, and also those between any pair of *distinct* non-blank labels. We allow all prefixes to start with either a blank (b) or the first symbol in $\mathbf{l}$ (l_1).

This gives us the following rules for initialisation

$$\begin{aligned}
\alpha_1(1) &= y^1_b \\
\alpha_1(2) &= y^1_{\mathbf{l}_1} \\
\alpha_1(s) &= 0, \ \forall s > 2
\end{aligned}$$

and recursion

$$\alpha_t(s) = \begin{cases} \bar{\alpha}_t(s) y^t_{l'_s} & \text{if } l'_s = b \text{ or } l'_{s-2} = l'_s \\ \big(\bar{\alpha}_t(s) + \alpha_{t-1}(s-2)\big) y^t_{l'_s} & \text{otherwise} \end{cases} \tag{6}$$

where

$$\bar{\alpha}_t(s) \stackrel{\text{def}}{=} \alpha_{t-1}(s) + \alpha_{t-1}(s-1). \tag{7}$$

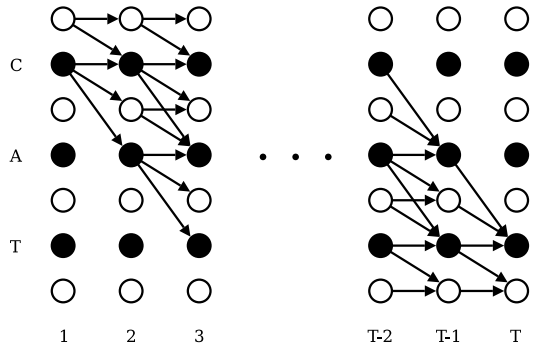

Figure 3. **illustration of the forward backward algorithm applied to the labelling 'CAT'**. Black circles represent labels, and white circles represent blanks. Arrows signify allowed transitions. Forward variables are updated in the direction of the arrows, and backward variables are updated against them.

Note that $\alpha_t(s) = 0 \ \forall s < |\mathbf{l}'| - 2(T-t) - 1$, because these variables correspond to states for which there are not enough time-steps left to complete the sequence (the unconnected circles in the top right of figure 3). Also $\alpha_t(s) = 0 \ \forall s < 1$.

The probability of $\mathbf{l}$ is then the sum of the total probabilities of $\mathbf{l}'$ with and without the final blank at time T.

$$p(\mathbf{l}|\mathbf{x}) = \alpha_T(|\mathbf{l}'|) + \alpha_T(|\mathbf{l}'| - 1). \tag{8}$$

Similarly, the backward variables $\beta_t(s)$ are defined as the total probability of $\mathbf{l}_{s:|\mathbf{l}|}$ at time t.

$$\beta_t(s) \stackrel{\text{def}}{=} \sum_{\substack{\pi \in N^T: \\ \mathcal{B}(\pi_{t:T}) = \mathbf{l}_{s:|\mathbf{l}|}}} \prod_{t'=t}^{T} y^{t'}_{\pi_{t'}} \tag{9}$$

$$\begin{aligned} \beta_T(|\mathbf{l}'|) &= y^T_b \\ \beta_T(|\mathbf{l}'| - 1) &= y^T_{\mathbf{l}_{|\mathbf{l}|}} \\ \beta_T(s) &= 0, \ \forall s < |\mathbf{l}'| - 1 \end{aligned}$$

$$\beta_t(s) = \begin{cases} \bar{\beta}_t(s) y^t_{l'_s} & \text{if } l'_s = b \text{ or } l'_{s+2} = l'_s \\ \big(\bar{\beta}_t(s) + \beta_{t+1}(s+2)\big) y^t_{l'_s} & \text{otherwise} \end{cases} \tag{10}$$

where

$$\bar{\beta}_t(s) \stackrel{\text{def}}{=} \beta_{t+1}(s) + \beta_{t+1}(s+1). \tag{11}$$

$\beta_t(s) = 0 \ \forall s > 2t$ (the unconnected circles in the bottom left of figure 3) and $\forall s > |\mathbf{l}'|$.

In practice, the above recursions will soon lead to underflows on any digital computer. One way of avoiding this is to rescale the forward and backward variables (Rabiner, 1989). If we define

$$C_t \stackrel{\text{def}}{=} \sum_s \alpha_t(s), \qquad \hat{\alpha}_t(s) \stackrel{\text{def}}{=} \frac{\alpha_t(s)}{C_t}$$

and substitute α for $\hat{\alpha}$ on the RHS of (6) and (7), the forward variables will remain in computational range. Similarly, for the backward variables we define

$$D_t \stackrel{\text{def}}{=} \sum_s \beta_t(s), \qquad \hat{\beta}_t(s) \stackrel{\text{def}}{=} \frac{\beta_t(s)}{D_t}$$

and substitute β for $\hat{\beta}$ on the RHS of (10) and (11).

To evaluate the maximum likelihood error, we need the natural logs of the target labelling probabilities. With the rescaled variables these have a particularly simple form:

$$ln(p(\mathbf{l}|\mathbf{x})) = \sum_{t=1}^{T} ln(C_t)$$

4.2. Maximum Likelihood Training

The aim of maximum likelihood training is to simultaneously maximise the log probabilities of all the correct classifications in the training set. In our case, this means minimising the following objective function:

$$O^{ML}(S, \mathcal{N}_w) = -\sum_{(\mathbf{x},\mathbf{z}) \in S} ln\big(p(\mathbf{z}|\mathbf{x})\big) \tag{12}$$

To train the network with gradient descent, we need to differentiate (12) with respect to the network outputs. Since the training examples are independent we can consider them separately:

$$\frac{\partial O^{ML}(\{(\mathbf{x},\mathbf{z})\}, \mathcal{N}_w)}{\partial y^t_k} = -\frac{\partial ln(p(\mathbf{z}|\mathbf{x}))}{\partial y^t_k} \tag{13}$$

We now show how the algorithm of section 4.1 can be used to calculate (13).

The key point is that, for a labelling $\mathbf{l}$, the product of the forward and backward variables at a given s and t is the probability of all the paths corresponding to $\mathbf{l}$ that go through the symbol s at time t. More precisely, from (5) and (9) we have:

$$\alpha_t(s)\beta_t(s) = \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = l_s}} y^t_{l_s} \prod_{t=1}^{T} y^t_{\pi_t}.$$

Rearranging and substituting in from (2) gives

$$\frac{\alpha_t(s)\beta_t(s)}{y^t_{l_s}} = \sum_{\substack{\pi \in \mathcal{B}^{-1}(\mathbf{l}): \\ \pi_t = l_s}} p(\pi|\mathbf{x}).$$

From (3) we can see that this is the portion of the total probability $p(\mathbf{l}|\mathbf{x})$ due to those paths going through l_s at time t. We can therefore sum over all s and t to get:

$$p(\mathbf{l}|\mathbf{x}) = \sum_{t=1}^{T} \sum_{s=1}^{|\mathbf{l}|} \frac{\alpha_t(s)\beta_t(s)}{y_{l_s}^t}. \tag{14}$$

Because the network outputs are conditionally independent (section 3.1), we need only consider the paths going through label k at time t to get the partial derivatives of $p(\mathbf{l}|\mathbf{x})$ with respect to y_k^t. Noting that the same label may be repeated several times in a single labelling $\mathbf{l}$, we define the set of positions where label k occurs as $lab(\mathbf{l}, k) = \{s : l_s = k\}$, which may be empty. We then differentiate (14) to get:

$$\frac{\partial p(\mathbf{l}|\mathbf{x})}{\partial y_k^t} = -\frac{1}{{y_k^t}^2} \sum_{s \in lab(\mathbf{l},k)} \alpha_t(s)\beta_t(s). \tag{15}$$

Observing that

$$\frac{\partial ln(p(\mathbf{l}|\mathbf{x}))}{\partial y_k^t} = \frac{1}{p(\mathbf{l}|\mathbf{x})} \frac{\partial p(\mathbf{l}|\mathbf{x})}{\partial y_k^t}$$

we can set $\mathbf{l} = \mathbf{z}$ and substitute (8) and (15) into (13) to differentiate the objective function.

Finally, to backpropagate the gradient through the softmax layer, we need the objective function derivatives with respect to the *unnormalised* outputs u_k^t.

If the rescaling of section 4.1 is used, we have:

$$\frac{\partial O^{ML}(\{(\mathbf{x}, \mathbf{z})\}, \mathcal{N}_w)}{\partial u_k^t} = y_k^t - \frac{Q_t}{y_k^t} \sum_{s \in lab(\mathbf{z},k)} \hat{\alpha}_t(s)\hat{\beta}_t(s) \tag{16}$$

where

$$Q_t \overset{\text{def}}{=} D_t \prod_{t'=t+1}^{T} \frac{D_{t'}}{C_{t'}}.$$

Eqn (16) is the 'error signal' received by the network during training (figure 4).

5. Experiments

We compared the performance of CTC with that of both an HMM and an HMM-RNN hybrid on a real-world temporal classification problem: phonetic labelling on the TIMIT speech corpus. More precisely, the task was to annotate the utterances in the TIMIT test set with the phoneme sequences that gave the lowest possible label error rate (as defined in section 2.1).

To make the comparison fair, the CTC and hybrid networks used the same RNN architecture: bidirectional Long Short-Term Memory (BLSTM; Graves & Schmidhuber, 2005). BLSTM combines the ability of Long Short-Term Memory (LSTM; Hochreiter & Schmidhuber, 1997) to bridge long time lags with the access of bidirectional RNNs (BRNNs; Schuster & Paliwal, 1997) to past and future context. We stress that any other architecture could have been used instead. We chose BLSTM because our experiments with standard BRNNs and unidirectional networks gave worse results on the same task.

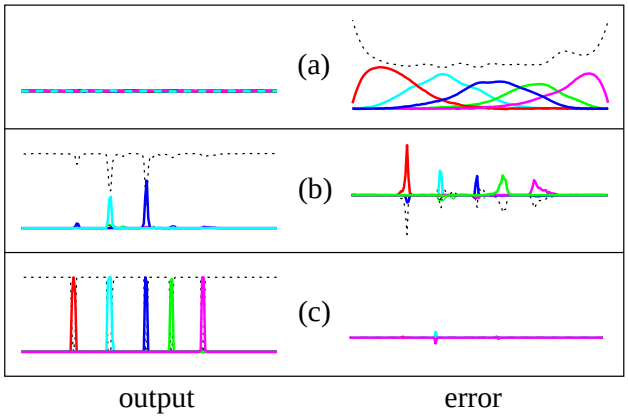

Figure 4. **Evolution of the CTC Error Signal During Training**. The left column shows the output activations for the same sequence at various stages of training (the dashed line is the 'blank' unit); the right column shows the corresponding error signals. Errors above the horizontal axis act to increase the corresponding output activation and those below act to decrease it. (a) Initially the network has small random weights, and the error is determined by the target sequence only. (b) The network begins to make predictions and the error localises around them. (c) The network strongly predicts the correct labelling and the error virtually disappears.

5.1. Data

TIMIT contain recordings of prompted English speech, accompanied by manually segmented phonetic transcripts. It has a lexicon of 61 distinct phonemes, and comes divided into training and test sets containing 4620 and 1680 utterances respectively. 5 % (184) of the training utterances were chosen at random and used as a validation set for early stopping in the hybrid and CTC experiments. The audio data was preprocessed into 10 ms frames, overlapped by 5 ms, using 12 Mel-Frequency Cepstrum Coefficients (MFCCs) from 26 filter-bank channels. The log-energy was also included, along with the first derivatives of all coefficients, giving a vector of 26 coefficients per frame in total. The coefficients were individually normalised to have mean 0 and standard deviation 1 over the training set.

5.2. Experimental Setup

The CTC network used an extended BLSTM architecture with peepholes and forget gates (Gers et al., 2002), 100 blocks in each of the forward and backward hidden layers, hyperbolic tangent for the input and output cell activation functions and a logistic sigmoid in the range $[0, 1]$ for the gates.

The hidden layers were fully connected to themselves and the output layer, and fully connected from the input layer. The input layer was size 26, the softmax output layer size 62 (61 phoneme categories plus the blank label), and the total number of weights was $114,662$.

Training was carried out with back propagation through time and online gradient descent (weight updates after every training example), using a learning rate of 10^{-4} and a momentum of 0.9. Network activations were reset to 0 at the start of each example. For prefix search decoding (section 3.2) the blank probability threshold was set at 99.99%. The weights were initialised with a flat random distribution in the range $[-0.1, 0.1]$. During training, Gaussian noise was added to the inputs with a standard deviation of 0.6 to improve generalisation.

The baseline HMM and hybrid systems were implemented as in (Graves et al., 2005). Briefly, baseline HMMs with context independent and context dependent three-states left-to-right models were trained and tested using the HTK Toolkit[1]. Observation probabilities were modelled by a mixture of Gaussians. Both the number of Gaussians and the insertion penalty were chosen to obtain the best performance on the task. Neither linguistic information nor probabilities of partial phone sequences were included in the system. There were more than $900,000$ parameters in total.

The hybrid system comprised an HMM and a BLSTM network, and was trained using Viterbi-based forced-alignment (Robinson, 1994). Initial estimation of transition and prior probabilities of the one-state 61 models was carried out using the correct transcription for the training set. Network output probabilities were divided by prior probabilities to obtain likelihoods for the HMM. The insertion penalty was chosen to obtain the best performance on the task.

The BLSTM architecture and parameters were identical to those used for CTC, with the following exceptions: (1) the learning rate for the hybrid network was 10^{-5}; (2) the injected noise had standard deviation 0.5; (3) the output layer had 61 units instead of 62 (no blank label). The noise and learning rate were set for the two systems independently, following a rough search in parameter space. The hybrid network had a total of $114,461$ weights, to which the HMM added 183 further parameters. For the weighted error experiment, the error signal was scaled to give equal weight to long and short phonemes (Robinson, 1991).

[1] http://htk.eng.cam.ac.uk/

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 %

5.3. Experimental Results

The results in table 1 show that, with prefix search decoding, CTC outperformed both a baseline HMM recogniser and an HMM-RNN hybrid with the same RNN architecture. They also show that prefix search gave a small improvement over best path decoding.

Note that the best hybrid results were achieved with a weighted error signal. Such heuristics are unnecessary for CTC, as its objective function depends only on the *sequence* of labels, and not on their duration or segmentation.

Input noise had a greater impact on generalisation for CTC than the hybrid system, and a higher level of noise was found to be optimal for CTC.

6. Discussion and Future Work

A key difference between CTC and other temporal classifiers is that CTC does not explicitly segment its input sequences. This has several benefits, such as removing the need to locate inherently ambiguous label boundaries (e.g. in speech or handwriting), and allowing label predictions to be grouped together if it proves useful (e.g. if several labels commonly occur together). In any case, determining the segmentation is a waste of modelling effort if only the label sequence is required.

For tasks where segmentation *is* required (e.g. protein secondary structure prediction), it would seem problematic to use CTC. However, as can be seen from figure 1, CTC naturally tends to align each label predic-

tion with the corresponding part of the sequence. This should make it suitable for tasks like keyword spotting, where approximate segmentation is sufficient.

Another distinctive feature of CTC is that it does not explicitly model inter-label dependencies. This is in contrast to graphical models, where the labels are typically assumed to form a kth order Markov chain. Nonetheless, CTC *implicitly* models inter-label dependencies, e.g. by predicting labels that commonly occur together as a double spike (see figure 1).

One very general way of dealing with structured data would be a hierarchy of temporal classifiers, where the labellings at one level (e.g. letters) become inputs for the labellings at the next (e.g. words). Preliminary experiments with hierarchical CTC have been encouraging, and we intend to pursue this direction further.

Good generalisation is always difficult with maximum likelihood training, but appears to be particularly so for CTC. In the future, we will continue to explore methods to reduce overfitting, such as weight decay, boosting and margin maximisation.

7. Conclusions

We have introduced a novel, general method for temporal classification with RNNs. Our method fits naturally into the existing framework of neural network classifiers, and is derived from the same probabilistic principles. It obviates the need for pre-segmented data, and allows the network to be trained directly for sequence labelling. Moreover, without requiring any task-specific knowledge, it has outperformed both an HMM and an HMM-RNN hybrid on a real-world temporal classification problem.

Acknowledgements

We thank Marcus Hutter for useful mathematical discussions. This research was funded by SNF grants 200021-111968/1 and 200020-107534/1.

References

Bengio., Y. (1999). Markovian models for sequential data. *Neural Computing Surveys*, *2*, 129–162.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*, chapter 6. Oxford University Press, Inc.

Bourlard, H., & Morgan, N. (1994). *Connnectionist speech recognition: A hybrid approach*. Kluwer Academic Publishers.

Bridle, J. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In F. Soulie and J.Herault (Eds.), *Neurocomputing: Algorithms, architectures and applications*, 227–236. Springer-Verlag.

Gers, F., Schraudolph, N., & Schmidhuber, J. (2002). Learning precise timing with LSTM recurrent networks. *Journal of Machine Learning Research*, *3*, 115–143.

Graves, A., Fernández, S., & Schmidhuber, J. (2005). Bidirectional LSTM networks for improved phoneme classification and recognition. *Proceedings of the 2005 International Conference on Artificial Neural Networks*. Warsaw, Poland.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*, 602–610.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*, 1735–1780.

Kadous, M. W. (2002). *Temporal classification: Extending the classification paradigm to multivariate time series*. Doctoral dissertation, School of Computer Science & Engineering, University of New South Wales.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning* (pp. 282–289). Morgan Kaufmann, San Francisco, CA.

LeCun, Y., Bottou, L., Orr, G., & Muller, K. (1998). Efficient backprop. *Neural Networks: Tricks of the trade*. Springer.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE* (pp. 257–286). IEEE.

Robinson, A. J. (1991). *Several improvements to a recurrent error propagation network phone recognition system* (Technical Report CUED/F-INFENG/TR82). University of Cambridge.

Robinson, A. J. (1994). An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, *5*, 298–305.

Schraudolph, N. N. (2002). Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent. *Neural Comp.*, *14*, 1723–1738.

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*, 2673–2681.

Werbos, P. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, *78*, 1550 – 1560.