Framewise and CTC Training of Neural Networks for Handwriting Recognition

Théodore Bluche*†, Hermann Ney †‡, Jérôme Louradour* and Christopher Kermorvant*§
*A2iA SA, Paris, France
†LIMSI CNRS, Spoken Language Processing Group, Orsay, France
‡RWTH Aachen University, Human Language Technology and Pattern Recognition, Aachen, Germany
§Teklia SAS, Paris, France

***Abstract*—In recent years, Long Short-Term Memory Recurrent Neural Networks (LSTM-RNNs) trained with the Connectionist Temporal Classification (CTC) objective won many international handwriting recognition evaluations. The CTC algorithm is based on a forward-backward procedure, avoiding the need of a segmentation of the input before training. The network outputs are characters labels, and a special non-character label. On the other hand, in the hybrid NN/HMM framework, networks are trained with framewise criteria to predict HMM state labels. In this paper, we show that CTC training is close to forward-backward training of NN/HMMs, and can be extended to more standard HMM topologies. We apply this method to MLPs, and investigate the properties of CTC, namely the modeling of character by single labels and the role of the special label.**

I. Introduction

Graves et al. [1] proposed a method to train Recurrent Neural Networks (RNNs) to label unsegmented sequences: Connectionist Temporal Classification (CTC). The main argument that motivated this method was that it does not require segmented training data, *i.e.* it requires only an input sequence and an output label sequence. The training algorithm consists of a forward-backward computation in a graph representing all possible segmentations of the input sequence given the expected output. Moreover, in the original formulation of CTC, the RNN has one output for each modeled character, plus one special *non-character* label, which we will name *blank* label in the remaining of this paper. The association of LSTM-RNNs and the CTC objective function for training won many handwriting recognition contests in the last few years (*e.g.* OpenHaRT [2], Maurdor [3]). In the cited examples, the RNN are the optical model of hybrid NN/HMM systems, allowing an easy integration of lexical constraints (vocabulary and *n-gram* language model) in the decoding procedure.

Typical HMM-based recognition methods, including hybrid systems, are borrowed from the advances in the speech recognition community. In these setups, character (or phone) HMMs consist of several states, and neural networks are trained on a framewise basis. Usually, a boostrapping system computes the forced alignments of training sequences, in order to segment the data and have a label (q) for each input (x) in the sequence. Then, standard NN training methods can be employed, such as stochastic gradient descent, and classification objective functions (*e.g.* cross-entropy). Hidden Markov Models training procedures (*e.g.* the Baum-Welch algorithm [4]) involve a forward-backward procedure to consider all possible segmentations of the input. Researchers have investigated forward-backward training of hybrid systems, to include both the NN and HMM in the training procedure, and to avoid the need for a bootstrapping system. Most of this work took place in the nineties [5], [6], [7], and do not seem to receive much attention nowadays. Forward-backward procedures are still found in sequence-discriminative training methods (MMI, MPE, sMBR [8], [9])

This paper is a study of the CTC training algorithm, in comparison to framewise training methods. First, in Section II, we show that the equations of CTC are very similar to those of forward-backward training of neural networks, and we point out the differences. Section III introduces the experiments that arise from these observations, namely the particularity of the structure of the CTC, which corresponds to modeling characters with a single state, and adding a non-character model, and the fact that this training algorithm is not limited to RNNs. In particular, we investigated in Section IV different topologies (CTC-like and more classical HMM topologies) for different optical models (GMMs, MLPs, RNNs). We trained MLPs with CTC, either with the topology defined in the CTC formulation, or with the best HMM topology. Finally, we compared, for all these setups, framewise and CTC training. We show that the CTC training improves the performance of MLP/HMMs, when used with the classical topology, while the best RNN results are obtained not only with CTC training, but also with the CTC topology. We summarize our conclusions in Section V.

II. Relation between CTC and Forward-Backward Training of Hybrid NN/HMMs

Let $\mathcal{Q}_n(\mathbf{W})$ be the set of all state sequences of length n representing some word sequence $\mathbf{W}$. For notation convenience, we also define $\mathcal{Q}(\mathbf{W})$, the list of all states in the model of $\mathbf{W}$ (a sequence of $\mathcal{Q}_n(\mathbf{W})$ is made of element of $\mathcal{Q}(\mathbf{W})$). Since in practice the same state can be used several times (and appear several times in $\mathcal{Q}(\mathbf{W})$ (*e.g.* the states of the HMM for character *"e"* appear twice in word *"tee"*), we define $\mathcal{Q}^*$ the set of all distinct states, and a mapping $\mu : \mathcal{Q}(\mathbf{W}) \mapsto \mathcal{Q}^*$ which provides the identity of a considered state. This is needed since HMMs have a single emission model for elements of $\mathcal{Q}^*$, when the forward-backward algorithms often consider elements of $\mathcal{Q}(\mathbf{W})$.

The basic idea of forward-backward training of neural networks is to use the scaled neural network posterior in the

HMM formulation:

$$p(\mathbf{x}|\mathbf{W}) = \sum_{\mathbf{q}\in\mathcal{Q}_{|\mathbf{x}|}(\mathbf{W})} \pi(q_1)\frac{p(q_1|x_1)}{p(q_1)}\prod_{t=2}^{|\mathbf{x}|}\frac{p(q_t|x_t)}{p(q_t)}p(q_t|q_{t-1}) \quad (1)$$

and to apply the same forward-backward procedure in the HMM as for Baum-Welch training, with forward and backward variables:

$$\begin{aligned}\alpha_t(s) &= p(x_{1:t}, q_t = s|\mathbf{W})\\ \beta_t(s) &= p(x_{t+1:T}|q_t = s, \mathbf{W})\end{aligned}$$

where $s \in \mathcal{Q}(\mathbf{W})$. These variables are computed iteratively, and the following recurrence equations apply with the neural network posteriors:

$$\alpha_t(s) = \frac{p(q_t = s|x_t)}{p(s)} \times \sum_r \alpha_{t-1}(r)p(q_t = s|q_{t-1} = r) \quad (2)$$

$$\beta_t(s) = \sum_r \frac{p(q_{t+1} = r|x_{t+1})}{p(r)}p(q_{t+1} = r|q_t = s)\beta_{t+1}(r) \quad (3)$$

Since $\frac{p(q|x)}{p(q)} = \frac{p(x|q)}{p(x)}$, this forward-backward procedure actually computes

$$\frac{p(\mathbf{x}|\mathbf{W})}{\prod_{t=1}^{|\mathbf{x}|}p(x_t)} = \frac{p(\mathbf{x}|\mathbf{W})}{p(\mathbf{x})} = \sum_{s\in\mathcal{Q}(\mathbf{W})}\alpha_t(s)\beta_t(s) \quad (4)$$

and we can derive the state posteriors given the word sequence $\mathbf{W}$:

$$p(q_t = s \in \mathcal{Q}(\mathbf{W})|\mathbf{x}, \mathbf{W}) = \frac{\alpha_t(s)\beta_t(s)}{\sum_r \alpha_t(r)\beta_t(r)}$$

Summing over all occurrences of a given state in the word sequence HMM, we get the posterior probability of a state given the observation and HMM:

$$p(q_t = k \in \mathcal{Q}^*|\mathbf{x}, \mathbf{W}) = \sum_{s:\mu(s)=k} p(q_t = s|\mathbf{x}, \mathbf{W}) \quad (5)$$

At every time t, the neural network computes a posterior distribution over elements of $\mathcal{Q}^*$, and thus we can use the distribution computed with Eqn. 5 as the target distribution in the cross-entropy training criterion. The backpropagated error is

$$\begin{aligned}\frac{\partial E}{\partial a_k^t} &= y_k^t - p(q_t = k \in \mathcal{Q}^*|\mathbf{x}, \mathbf{W}) &(6)\\ &= y_k^t - \sum_{s:\mu(s)=k}\frac{\alpha_t(s)\beta_t(s)}{\sum_r \alpha_t(r)\beta_t(r)} &(7)\end{aligned}$$

where y_k^t is the output of the neural network at time t for class k, *i.e.* $p(q_t = k|x_t)$, and a_k^t are the activations before the softmax.

Several papers about forward-backward training of neural networks were published. The idea is to replace the "hard" Viterbi segmentation of the input sequence, where the classification targets are HMM states, by considering all possible segmentations, as in the Baum-Welch algorithm for HMMs. In some works, such as [6], [7], it is assumed that $\frac{p(q|x)}{p(q)} \approx p(x|q)$, so Eqns. 1 and 4 both compute $p(\mathbf{x}|\mathbf{W})$. The "soft" targets are obtained with Eqn. 5.

In [5], the same recurrences on α and β compute $\frac{p(\mathbf{x}|\mathbf{W})}{p(\mathbf{x})}$, and posteriors are again obtained with Eqn. 5. The original goal of that work was to optimize directly $p(\mathbf{W}|\mathbf{x})$, in which we are interested for decoding. Some assumptions, such as the usual limitation of the dependency of the HMM state q_t to the current observation (or some local context) given the previous state ($p(q_t|\mathbf{x}, q_{t-1}) = p(q_t|x_t, q_{t-1})$) leads to the REMAP formulation and special kind of neural network [10]. When furthermore the dependency on the previous state is dropped, given the current observation, we obtain the equations from [5].

In these works, the networks are first trained with Viterbi alignments. Then, the targets are re-estimated with the forward-backward procedure, and the network is trained with the obtained posterior probabilities. We can show (simple derivation of the expressions of the posteriors in terms of α and β) that when the cross-entropy is the optimized criterion, and the re-estimation is made after each batch (epoch or training example), the method of [6], [7] is equivalent to training the network with the observation negative log-likelihood (NLL) ($-\log p(\mathbf{x}|\mathbf{W})$), while in [5], the NLL of the word sequences is minimized ($-\log p(\mathbf{W}|\mathbf{x})$), provided that the prior distribution of word sequences (language model) is considered constant.

The goal of CTC [1] is to use a neural network to transform an input sequence $\mathbf{x}$ into a (shorter) output sequence of labels $\mathbf{L}$ (*e.g.* a sequence of characters) using the NN predictions with no complicated post-processing. The proposed method defines the NN outputs to be the set of possible labels, plus a *blank* output ($\varnothing$). This way, a mapping transforms sequence of predictions into the final transcriptions by first removing successive identical labels, and then blanks, for example:

$$a\ a \varnothing \varnothing\ b\ b \varnothing\ \ b\ a \mapsto abba$$

One of the motivations for the special blank label is to be able to use this simple mapping function and still output two identical labels in the output sequence [1]. The authors also suggest that this label should model everything outside the relevant parts of the input sequences, such as the connections or short wihtespaces between characters in an image. A blank prediction is only mandatory between two consecutive and identical labels.

With this labeling problem at hand, the NN is trained to minimize the NLL of the label sequence given the input sequence ($-\log p(\mathbf{L}|\mathbf{x})$). Several prediction sequences yield the same label sequence (*e.g.* "$a\ a\ b\ b$", "$a\ a\ a\ b$", "$a\varnothing b\ b$", ...). To simplify the analogy with the methods presented previously, let $\mathcal{Q}_n(\mathbf{L})$ be the set of all output sequences mapping to $\mathbf{L}$. Then:

$$p(\mathbf{L}|\mathbf{x}) = \sum_{\mathbf{q}\in\mathcal{Q}_{|\mathbf{x}|}(\mathbf{L})} p(\mathbf{q}|\mathbf{x})$$

In [1], the authors assume that the predictions made at different timesteps are independent given the observation sequence, hence:

$$p(\mathbf{L}|\mathbf{x}) = \sum_{\mathbf{q}\in\mathcal{Q}_{|\mathbf{x}|}(\mathbf{L})}\prod_{t=1}^{|\mathbf{x}|} p(q_t|\mathbf{x})$$

This quantity can also be efficiently computed with a forward-backward procedure. The mapping defines the allowed transitions between labels: either continue to predict the same label, jump to the next one if it is different, or jump to a blank. The forward and backward variables are defined as follows, with $\mathbf{L} = l_1 \dots l_n$ and $\mathbf{L}' = l'_1 \dots l'_n = \varnothing l_1 \varnothing \dots \varnothing l_n \varnothing$

$$\begin{aligned}\alpha_t(l'_s) &= p(q_{1:t} \in \mathcal{Q}_t(\mathbf{L}_{1:s/2}), q_t = l'_s | \mathbf{x}) \\ \beta_t(l'_s) &= p(q_{t+1:T} \in \mathcal{Q}_{T-t}(\mathbf{L}_{s/2+1:|\mathbf{L}|}), q_t = l'_s | \mathbf{x})\end{aligned}$$

and the recurrences are:

$$\alpha_t(l'_s) = p(q_t = l'_s|\mathbf{x}) \sum_{n=0}^{k} \alpha_{t-1}(l'_{s-n})$$

$$\beta_t(l'_s) = \sum_{n=0}^{k} p(q_{t+1} = l'_{s+n}|\mathbf{x}) \beta_{t+1}(l'_{s+n})$$

where $k = 1$ whenever $l'_s = \varnothing$ or $l'_s = l'_{s-2}$ (*resp.* $l'_s = l'_{s+2}$) for forward (*resp.* backward) variables, and 2 otherwise. Again, the network targets are given by Eqn. 5.

The equations are close from the ones of forward-backward training of hybrid NN/HMM models, such as those of [5]. We had to introduce some notation to take into account the allowed transitions, but there are mainly only two differences. First, the CTC uses $p(q_t|\mathbf{x})$ instead of $p(q_t|x_t)$, because the CTC appear in the context of RNNs, which make predictions potentially based on the whole sequence. If we make the assumption from [5] that the output only depends on some local context, the CTC is not anymore limited to RNNs, and may be applied to any neural network.

The second difference is that CTC does not take into account the transition probabilities, and setting $\frac{p(r|s)}{p(r)} = 1$ when a transition from s to r exists and 0 otherwise in Eqns. 2 and 3, we obtain the CTC equations. Thus, CTC is *(i)* a simplification of the forward-backward training of hybrid NN/HMMs and *(ii)* associated with a simple topology where each character is modeled with a single state, plus an optional state between characters.

III. Experimental Setup

Is the CTC topology better than another one for CTC training of RNNs? The **transition model** in the CTC is principally designed so that the neural network can be used alone to predict the desired output outside the HMM framework. In practice, however, the HMM framework is convenient, in particular for the integration of the language model. We have included CTC trained RNNs in hybrid NN/HMM handwriting recognition systems and obtained good results with language models (*e.g.* in [2], [3], [11]). The HMMs were designed to match the CTC topology: one state for every character, plus an optional state for blanks, with self-loops. We also found that the inclusion of state (label) priors as in standard hybrids gives better results. In this framework, the CTC topology is not required. In the first set of experiments, we varied the number of states in CTC training of RNNs, with number of states per character ranging from one to seven.

Is the blank label useful for handwriting modeling? When and why is it useful? The remaining **motivation for a blank symbol** when we remove the need for a standalone neural network is the modeling of the inter-character signal. We tried the previous topologies with the different optical models (GMM, MLP, RNN), with and without the blank symbol.

We have seen that CTC training is close to forward-backward training of hybrid NN/HMM, which is not limited to the CTC topology. Thus, it can apply to other neural networks, such as MLPs. We complemented the previous experiments with **CTC training of MLPs**, and with **framewise training of RNNs**, for all variations of the topologies. The goal is to better answer the previous questions to understand the relationship between the topology, the blank symbol, and the CTC training algorithm, and to compare these different training methods.

The experiments are carried out on the public IAM database [12] of handwritten English text lines. The database consists of 747 documents (6,482 text lines) for training, 116 (976) for development and 336 (2,915) for test. The writers only contributed to one set. For all experiments, we preprocessed the image by applying skew and slant correction, contrast enhancement, and rescaling the images to a height of 72px (24px for each zone: ascenders, core, descenders). We scanned a sliding window of width 3px to the obtained image to extract 56 geometrical and statistical features [13]. The average number of frames per character is about 10. For decoding, we used a trigram language model, trained on the LOB, Wellington and Brown corpora, and limited to the most frequent 50k words. We report results on the development set, for which the language model has an out-of-vocabulary word rate of 4.3% and a perplexity of 298.

The training of GMMs follows the standard EM procedure, realigning the data with the Viterbi algorithm and updating the mixtures to maximize the data likelihood, increasing the number of Gaussians at each step, until no more improvement is observed on the validation data. We chose neural network architectures to be a tradeoff between model size (training time) and performance. The MLPs have two hidden layers of 1,024 sigmoid units, and with inputs consisting of 11 consecutive frames of 56-dimensional feature vectors to take into account the context (this number has been tuned empirically, and we selected the one yielding the lowest classification error on the validation set). The RNNs have two LSTM hidden layers with 100 units (one for each direction), and directly consider the sequence of feature vectors. In framewise training, the optimized criterion is the NLL at frame level. Training stops when the cost on the validation data does not decrease for 20 epochs, and the best network is selected.

IV. Experiments and Results

A. Topology and Blank

First, we trained standard GMM/HMMs systems. Each state has its own emission probability distribution, hence its own set of parameters. We wanted to confirm that several states per character are better than one or two, and to check whether a *blank* model to take care of the inter-character parts could help. The results are shown on the top part of Table I. We see that adding states improves the results. We also notice by comparing the two lines that adding a blank model between characters helps too. However, we always got better results by incrementing the character model sizes than by inserting blanks between characters.

TABLE I: WER% (CER%) of different standard systems with different topologies.

States	1	2	3	4	5	6	7
GMM							
No blank		25.7 (15.5)	20.8 (10.7)	17.3 (8.2)	16.7 (7.7)	16.5 (7.4)	**16.3 (6.9)**
Blank	30.1 (18.0)	23.5 (12.6)	18.3 (8.7)	17.1 (7.7)	17.3 (7.4)	17.0 (7.4)	18.7 (8.6)
MLP							
No blank		17.8 (8.2)	15.0 (6.1)	13.6 (5.3)	13.2 (4.8)	**12.4 (4.6)**	14.8 (4.8)
Blank	19.6 (9.0)	16.0 (6.3)	14.4 (5.5)	14.1 (5.2)	13.9 (5.2)	14.3 (5.9)	16.0 (6.7)
RNN							
No blank		19.3 (8.0)	16.5 (6.1)	14.1 (5.3)	13.7 (5.0)	14.1 (4.9)	14.5 (5.1)
Blank	**13.1 (4.9)**	13.9 (5.0)	14.3 (5.2)	13.9 (5.1)	14.9 (5.4)	15.3 (5.8)	14.2 (5.4)

The MLPs were trained with the framewise NLL criterion, which focuses on the classification of individual frames. The ground-truth targets are obtained by forced alignment of the training data with a bootstrapping system (the corresponding GMM system). The results are presented in the middle part of Table I. Apart from the well-known huge improvements brought by these discriminative models over the generative GMMs, we draw the same conclusions concerning the number of states: adding states in the character models improves the results. The blank model, however, only helps when the character models are small.

The RNNs were trained with the CTC criterion, but without the CTC constraints of one output per character and blank. Since we use RNNs in hybrid mode with HMMs, the outputs are the different HMM states, as for MLPs. So we apply CTC with the transition model defined by the HMMs. The results are presented in the bottom part of Table I. Without blank, increasing the number of states still improves the performance. As for MLPs, adding a blank only helps when characters are modeled with a few states.

However, there is one noticeable difference with MLP results: for RNNs with the blank symbol, adding states to the character models leads to worse results. For the previous models, the CTC topology gave the worst results, but for CTC-trained RNNs, it looks like this topology is the best choice. A reasonable question to ask at this point is whether it is due to the CTC training or to the RNN, or maybe a combination of both. The next two sets of experiments (Section IV-B and IV-C) attempt to answer it.

B. CTC Training of MLPs

The CTC training criterion was proposed in the context of RNNs. The only part of the equations that makes it particularly suited to RNNs is the representation of NN outputs as $p(q_t|\mathbf{x})$, meaning that the prediction depends on the whole input sequence. The limitation disappears when we assume as in [5] that a state label only depends on some local context. The algorithm is then applicable to any model of $p(q_t|x_t)$. Moreover, we outlined in Section II the resemblance of the CTC criterion to the forward-backward training of hybrid NN/HMMs, and like we did for the last RNN experiments, we can apply the CTC criterion to MLP, with the best HMM topology (6 states/character, no blank symbol).

The results are presented on Table II, and compared to the best MLP trained with a framewise criterion. With the 6-state topology without blank (*"HMM"* topology), we observe some limited improvement, from 4.6% CER to 4.3%. On the other hand, with the classical CTC topology, consisting of one state per character and a blank model, the results are far worse. This choice of topology, which was the best one for CTC training of RNNs, is not suited to MLPs, even with CTC training. Note however that CTC training still improved the error rates for this topology over framewise training (19.6% WER / 9.0% CER in Table I). In the next experiment, we conduct a more thorough comparison of framewise and CTC training, with different topologies, for MLPs and RNNs.

TABLE II: CTC training of MLPs.

Training	Topology	WER%	CER%
Framewise	HMM	12.4	4.6
CTC	HMM	12.6	4.3
	CTC	17.6	7.4

C. Framewise vs CTC Training

In this section, we compare the results of framewise and CTC (forward-backward) training of neural networks. For each topology (1 to 7 states, with and without blank), we trained MLPs and RNNs. The results are summarized on Table III, and in Fig. 1. With blank, CTC training gives better results than framewise training when characters are represented by a few states (one or two). With more states, there was only small differences for RNNs, but the results tend to be worse with CTC for MLPs. We make the opposite observation without blank: for MLPs and RNNs, the results are improved with CTC training when there are many states per characters, but degraded for one or two states.

TABLE III: WER%/CER% with framewise vs CTC training.

		Without blank		With blank	
	States	Framewise	CTC	Framewise	CTC
MLP	1	-	-	19.6 / 9.0	17.6 / 7.4
	2	17.8 / 8.2	19.1 / 8.5	16.0 / 6.3	16.4 / 6.7
	3	15.0 / 6.1	15.2 / 6.1	14.4 / 5.5	16.4 / 6.5
	4	13.6 / 5.3	13.3 / 4.9	14.1 / 5.2	14.9 / 5.6
	5	13.2 / 4.8	13.0 / 4.5	13.9 / 5.2	14.9 / 5.6
	6	**12.4** / 4.6	12.6 / **4.3**	14.3 / 5.9	16.1 / 6.4
	7	12.8 / 4.8	12.7 / 4.3	16.0 / 6.7	17.4 / 7.0
RNN	1	-	-	18.7 / 8.2	**13.1 / 4.9**
	2	17.7 / 7.5	19.3 / 8.0	15.9 / 6.1	13.9 / 5.0
	3	15.4 / 5.6	16.5 / 6.1	14.4 / 5.8	14.3 / 5.2
	4	14.2 / 5.4	14.1 / 5.3	13.8 / 5.3	13.9 / 5.1
	5	14.2 / 5.1	13.7 / 5.0	14.3 / 5.2	14.2 / 5.1
	6	14.0 / 5.1	14.1 / 4.9	14.6 / 5.8	15.3 / 5.8
	7	14.6 / 5.2	14.5 / 5.1	15.7 / 6.4	14.2 / 5.4

Moreover, CTC training without blank and with less than 5 states per character converged to a poor local optimum, for both neural networks, where most of the outputs are whitespace predictions. The training algorithm did not manage to find a reasonable alignment, and the resulting WERs / CERs were above 90%. To obtain the presented results, we had to initialize the networks with one epoch of framewise training. This problem did not occur when a blank was added, suggesting that this symbol plays a role in the success in the alignment procedure in early stages of CTC training.

D. The Role of the Blank Symbol

The blank symbol helps only when characters are modeled with a few states. It is especially useful for CTC training in that

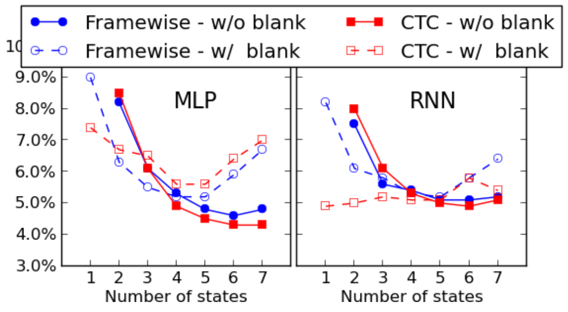

Fig. 1: Character Error Rates with different Neural Networks, training methods, and HMM topologies.

context, where it is crucial for convergence. It is particularly good for RNNs: the best results are obtained with one state per character, CTC training, and blanks.

Why does it help? A typical observation in CTC-trained RNNs is the dominance of blank predictions in the output sequence, with localized peaks for character predictions. Interestingly, this behaviour still occurs with more states (*e.g.* with four states: four peaks of states predictions with many blanks between each character), and with MLPs. Thus, we can conclude that this typical output is due to the interaction between blank and CTC training more than to character models or RNNs. This is not surprising. In the CTC graph, there is one blank between each character. In early stages of training, the NN outputs are more or less random. Any path is equally likely, but summing all blank posteriors for a given timestep, the target for blank with be much higher than for any other label. We indeed observe that the network start predicting only blanks. But predicting only blanks penalizes a lot the cost function and the network has only to find one location where to predict each character, to "jump" from one sequence of blanks to another. This is much easier for RNNs, which can use a context of arbitrary length to make a prediction at a specific timestep, and not use much of the input to make blank prediction. Conversely, all predictions in MLPs must be made from an arbitrary context, and it is much more difficult to only predict blanks except at specific timesteps.

Moreover, it looks like having this uninformative symbol between every character helps the network to reach good alignments in the CTC algorithm and to converge to good solutions. Without blank, the predominant symbol is the whitespace, and systems with few states converged to solutions where character predictions are made at the beginning and end of the sentence, the rest being overwhelmed by whitespace predictions. These observations also explain why the blank in CTC helps only with a few states per character. Indeed, the more states, the more successive peaks with a different label the network must predict, *i.e.* different labels for probably similar inputs.

Finally, this behaviour is also interesting for decoding. Models with one state and a blank yield much smaller decoding graphs. Furthermore, since the character predictions are very localized, and the blank is uninformative and shared by all word models, the number of predictions to change in order to recognize a different word is small. It means that correcting mistakes is not very costly. That also means that it is easier to keep more various hypotheses during beam search decoding. We observed that the optimal optical scale in decoding is related to the length of the characters (in the sequence of predictions). Without blank, there are roughly between 10 and 15 frames per characters in this database, and the best optical scale is always between 10^{-1} and 15^{-1}. With CTC training and blank, the predictions are localized and correspond to 1 or 2 timesteps (with one-state models, and around N timesteps for N-state ones), and the optimal optical scale is always around N^{-1}.

V. Conclusions

In this paper, we have studied the CTC framework for RNNs and MLPs, varying the output model. We have shown that CTC was quite similar to forward-backward training of hybrid NN/HMM systems. Through experiments, we observed that the original CTC formulation, with one output per character, and a special blank symbol, is especially suited to RNN optical models.

References

[1] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 369–376.

[2] T. Bluche, J. Louradour, M. Knibbe, B. Moysset, M. F. Benzeghiba, and C. Kermorvant, "The a2ia arabic handwritten text recognition system at the open hart2013 evaluation," in *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*. IEEE, 2014, pp. 161–165.

[3] B. Moysset, T. Bluche, M. Knibbe, M. Benzeghiba, R. Messina, J. Louradour, and C. Kermorvant, "The a2ia multi-lingual text recognition system at the maurdor evaluation," in *14th International Conference on Frontiers in Handwriting Recognition (ICFHR-2014)*, 2014.

[4] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[5] J. Hennebert, C. Ris, H. Bourlard, S. Renals, and N. Morgan, "Estimation of global posteriors and forward-backward training of hybrid hmm/ann systems." 1997.

[6] A. Senior and T. Robinson, "Forward-backward retraining of recurrent neural networks." *Advances in Neural Information Processing Systems*, pp. 743–749, 1996.

[7] Y. Yan, M. Fanty, and R. Cole, "Speech recognition using neural networks with forward-backward probability generated targets," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, vol. 4. IEEE Computer Society, 1997, pp. 3241–3241.

[8] B. Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *IEEE International Conference on Acoustics, Speech and Signal Processing, 2009. ICASSP 2009*. IEEE, 2009, pp. 3761–3764.

[9] K. Veselý, A. Ghoshal, L. Burget, and D. Povey, "Sequence-discriminative training of deep neural networks," in *Interspeech*, no. 1, 2013, pp. 3–7.

[10] Y. Konig, H. Bourlard, and N. Morgan, "Remap: Recursive estimation and maximization of a posteriori probabilities-application to transition-based connectionist speech recognition," *Advances in Neural Information Processing Systems*, pp. 388–394, 1996.

[11] V. Pham, C. Kermorvant, and J. Louradour, "Dropout improves Recurrent Neural Networks for Handwriting Recognition," Nov. 2013. [Online]. Available: http://arxiv.org/abs/1312.4569

[12] U. V. Marti and H. Bunke, "The IAM-database: an English sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, 2002.

[13] A.-L. Bianne, F. Menasri, R. Al-Hajj, C. Mokbel, C. Kermorvant, and L. Likforman-Sulem, "Dynamic and Contextual Information in HMM modeling for Handwriting Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 10, pp. 2066 – 2080, 2011.