Towards End-to-End Speech Recognition with Recurrent Neural Networks

Alex Graves GRAVES@CS.TORONTO.EDU
Google DeepMind, London, United Kingdom

Navdeep Jaitly NDJAITLY@CS.TORONTO.EDU
Department of Computer Science, University of Toronto, Canada

Abstract

This paper presents a speech recognition system that directly transcribes audio data with text, without requiring an intermediate phonetic representation. The system is based on a combination of the deep bidirectional LSTM recurrent neural network architecture and the Connectionist Temporal Classification objective function. A modification to the objective function is introduced that trains the network to minimise the expectation of an arbitrary transcription loss function. This allows a direct optimisation of the word error rate, even in the absence of a lexicon or language model. The system achieves a word error rate of 27.3% on the Wall Street Journal corpus with no prior linguistic information, 21.9% with only a lexicon of allowed words, and 8.2% with a trigram language model. Combining the network with a baseline system further reduces the error rate to 6.7%.

1. Introduction

Recent advances in algorithms and computer hardware have made it possible to train neural networks in an end-to-end fashion for tasks that previously required significant human expertise. For example, convolutional neural networks are now able to directly classify raw pixels into high-level concepts such as object categories (Krizhevsky et al., 2012) and messages on traffic signs (Ciresan et al., 2011), without using hand-designed feature extraction algorithms. Not only do such networks require less human effort than traditional approaches, they generally deliver superior performance. This is particularly true when very large amounts of training data are available, as the benefits of holistic optimisation tend to outweigh those of prior knowledge.

While automatic speech recognition has greatly benefited from the introduction of neural networks (Bourlard & Morgan, 1993; Hinton et al., 2012), the networks are at present only a single component in a complex pipeline. As with traditional computer vision, the first stage of the pipeline is input feature extraction: standard techniques include mel-scale filterbanks (Davis & Mermelstein, 1980) (with or without a further transform into Cepstral coefficients) and speaker normalisation techniques such as vocal tract length normalisation (Lee & Rose, 1998). Neural networks are then trained to classify individual frames of acoustic data, and their output distributions are reformulated as emission probabilities for a hidden Markov model (HMM). The objective function used to train the networks is therefore substantially different from the true performance measure (sequence-level transcription accuracy). This is precisely the sort of inconsistency that end-to-end learning seeks to avoid. In practice it is a source of frustration to researchers, who find that a large gain in frame accuracy can translate to a negligible improvement, or even deterioration in transcription accuracy. An additional problem is that the frame-level training targets must be inferred from the alignments determined by the HMM. This leads to an awkward iterative procedure, where network retraining is alternated with HMM re-alignments to generate more accurate targets. Full-sequence training methods such as Maximum Mutual Information have been used to directly train HMM-neural network hybrids to maximise the probability of the correct transcription (Bahl et al., 1986; Jaitly et al., 2012). However these techniques are only suitable for retraining a system already trained at frame-level, and require the careful tuning of a large number of hyper-parameters—typically even more than the tuning required for deep neural networks.

While the transcriptions used to train speech recognition systems are lexical, the targets presented to the networks

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

are usually phonetic. A pronunciation dictionary is therefore needed to map from words to phoneme sequences. Creating such dictionaries requires significant human effort and often proves critical to overall performance. A further complication is that, because multi-phone contextual models are used to account for co-articulation effects, 'state tying' is needed to reduce the number of target classes—another source of expert knowledge. Lexical states such as graphemes and characters have been considered for HMM-based recognisers as a way of dealing with out of vocabulary (OOV) words (Galescu, 2003; Bisani & Ney, 2005), however they were used to augment rather than replace phonetic models.

Finally, the acoustic scores produced by the HMM are combined with a language model trained on a text corpus. In general the language model contains a great deal of prior information, and has a huge impact on performance. Modelling language separately from sound is perhaps the most justifiable departure from end-to-end learning, since it is easier to learn linguistic dependencies from text than speech, and arguable that literate humans do the same thing. Nonetheless, with the advent of speech corpora containing tens of thousands of hours of labelled data, it may be possible to learn the language model directly from the transcripts.

The goal of this paper is a system where as much of the speech pipeline as possible is replaced by a single recurrent neural network (RNN) architecture. Although it is possible to directly transcribe raw speech waveforms with RNNs (Graves, 2012, Chapter 9) or features learned with restricted Boltzmann machines (Jaitly & Hinton, 2011), the computational cost is high and performance tends to be worse than conventional preprocessing. We have therefore chosen spectrograms as a minimal preprocessing scheme.

The spectrograms are processed by a deep bidirectional LSTM network (Graves et al., 2013) with a Connectionist Temporal Classification (CTC) output layer (Graves et al., 2006; Graves, 2012, Chapter 7). The network is trained directly on the text transcripts: no phonetic representation (and hence no pronunciation dictionary or state tying) is used. Furthermore, since CTC integrates out over all possible input-output alignments, no forced alignment is required to provide training targets. The combination of bidirectional LSTM and CTC has been applied to character-level speech recognition before (Eyben et al., 2009), however the relatively shallow architecture used in that work did not deliver compelling results (the best character error rate was almost 20%).

The basic system is enhanced by a new objective function that trains the network to directly optimise the word error rate.

Experiments on the Wall Street Journal speech corpus demonstrate that the system is able to recognise words to reasonable accuracy, even in the absence of a language model or dictionary, and that when combined with a language model it performs comparably to a state-of-the-art pipeline.

2. Network Architecture

Given an input sequence $\boldsymbol{x} = (x_1, \dots, x_T)$, a standard recurrent neural network (RNN) computes the hidden vector sequence $\boldsymbol{h} = (h_1, \dots, h_T)$ and output vector sequence $\boldsymbol{y} = (y_1, \dots, y_T)$ by iterating the following equations from $t = 1$ to T:

$$h_t = \mathcal{H}\left(W_{ih}x_t + W_{hh}h_{t-1} + b_h\right) \quad (1)$$

$$y_t = W_{ho}h_t + b_o \quad (2)$$

where the W terms denote weight matrices (e.g. W_{ih} is the input-hidden weight matrix), the b terms denote bias vectors (e.g. b_h is hidden bias vector) and $\mathcal{H}$ is the hidden layer activation function.

$\mathcal{H}$ is usually an elementwise application of a sigmoid function. However we have found that the Long Short-Term Memory (LSTM) architecture (Hochreiter & Schmidhuber, 1997), which uses purpose-built *memory cells* to store information, is better at finding and exploiting long range context. Fig. 1 illustrates a single LSTM memory cell. For the version of LSTM used in this paper (Gers et al., 2002) $\mathcal{H}$ is implemented by the following composite function:

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \quad (3)$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \quad (4)$$

$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \quad (5)$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \quad (6)$$

$$h_t = o_t \tanh(c_t) \quad (7)$$

where σ is the logistic sigmoid function, and i, f, o and c are respectively the *input gate*, *forget gate*, *output gate* and *cell* activation vectors, all of which are the same size as the hidden vector h. The weight matrix subscripts have the obvious meaning, for example W_{hi} is the hidden-input gate matrix, W_{xo} is the input-output gate matrix etc. The weight matrices from the cell to gate vectors (e.g. W_{ci}) are diagonal, so element m in each gate vector only receives input from element m of the cell vector. The bias terms (which are added to i, f, c and o) have been omitted for clarity.

One shortcoming of conventional RNNs is that they are only able to make use of previous context. In speech recognition, where whole utterances are transcribed at once, there is no reason not to exploit future context as well. Bidirectional RNNs (BRNNs) (Schuster & Paliwal, 1997)

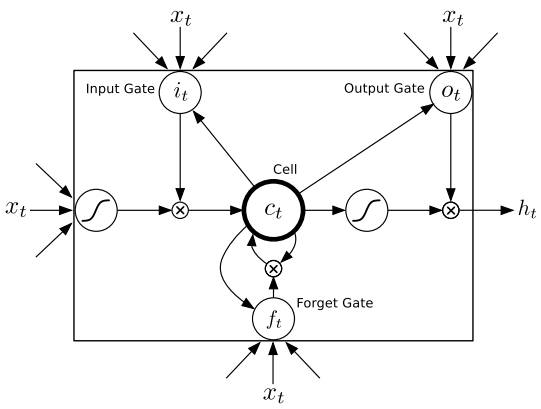

Figure 1. **Long Short-term Memory Cell.**

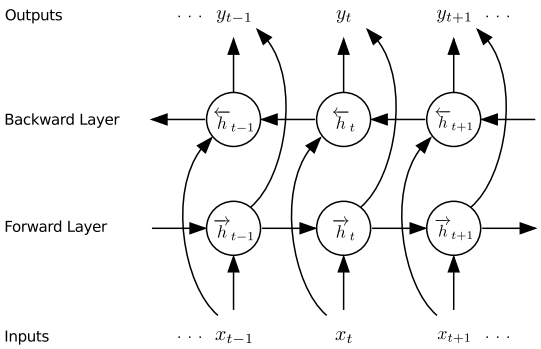

Figure 2. **Bidirectional Recurrent Neural Network.**

do this by processing the data in both directions with two separate hidden layers, which are then fed forwards to the same output layer. As illustrated in Fig. 2, a BRNN computes the *forward* hidden sequence $\overrightarrow{\boldsymbol{h}}$, the *backward* hidden sequence $\overleftarrow{\boldsymbol{h}}$ and the output sequence $\boldsymbol{y}$ by iterating the backward layer from $t = T$ to 1, the forward layer from $t = 1$ to T and then updating the output layer:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right) \quad (8)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right) \quad (9)$$

$$y_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_o \quad (10)$$

Combing BRNNs with LSTM gives bidirectional LSTM (Graves & Schmidhuber, 2005), which can access long-range context in both input directions.

A crucial element of the recent success of hybrid systems is the use of *deep* architectures, which are able to build up progressively higher level representations of acoustic data. *Deep RNNs* can be created by stacking multiple RNN hidden layers on top of each other, with the output sequence of one layer forming the input sequence for the next, as shown in Fig. 3. Assuming the same hidden layer function is used

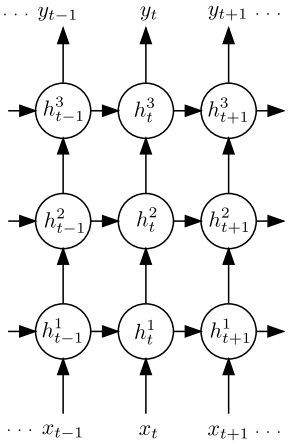

Figure 3. **Deep Recurrent Neural Network.**

for all N layers in the stack, the hidden vector sequences $\boldsymbol{h}^n$ are iteratively computed from $n = 1$ to N and $t = 1$ to T:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n}h_t^{n-1} + W_{h^nh^n}h_{t-1}^n + b_h^n\right) \quad (11)$$

where $\boldsymbol{h}^0 = \boldsymbol{x}$. The network outputs y_t are

$$y_t = W_{h^Ny}h_t^N + b_o \quad (12)$$

Deep bidirectional RNNs can be implemented by replacing each hidden sequence $\boldsymbol{h}^n$ with the forward and backward sequences $\overrightarrow{\boldsymbol{h}}^n$ and $\overleftarrow{\boldsymbol{h}}^n$, and ensuring that every hidden layer receives input from both the forward and backward layers at the level below. If LSTM is used for the hidden layers the complete architecture is referred to as deep bidirectional LSTM (Graves et al., 2013).

3. Connectionist Temporal Classification

Neural networks (whether feedforward or recurrent) are typically trained as frame-level classifiers in speech recognition. This requires a separate training target for every frame, which in turn requires the alignment between the audio and transcription sequences to be determined by the HMM. However the alignment is only reliable once the classifier is trained, leading to a circular dependency between segmentation and recognition (known as Sayre's paradox in the closely-related field of handwriting recognition). Furthermore, the alignments are irrelevant to most speech recognition tasks, where only the word-level transcriptions matter. Connectionist Temporal Classification (CTC) (Graves, 2012, Chapter 7) is an objective function that allows an RNN to be trained for sequence transcription tasks without requiring any prior alignment between the input and target sequences.

The output layer contains a single unit for each of the transcription labels (characters, phonemes, musical notes etc.), plus an extra unit referred to as the 'blank' which corresponds to a null emission. Given a length T input sequence $\boldsymbol{x}$, the output vectors y_t are normalised with the softmax function, then interpreted as the probability of emitting the label (or blank) with index k at time t:

$$\Pr(k, t|\boldsymbol{x}) = \frac{\exp\left(y_t^k\right)}{\sum_{k'} \exp\left(y_t^{k'}\right)} \tag{13}$$

where y_t^k is element k of y_t. A CTC alignment $\boldsymbol{a}$ is a length T sequence of blank and label indices. The probability $\Pr(\boldsymbol{a}|\boldsymbol{x})$ of $\boldsymbol{a}$ is the product of the emission probabilities at every time-step:

$$\Pr(\mathbf{a}|\boldsymbol{x}) = \prod_{t=1}^{T} \Pr(\boldsymbol{a}_t, t|\boldsymbol{x}) \tag{14}$$

For a given transcription sequence, there are as many possible alignments as there different ways of separating the labels with blanks. For example (using '$-$' to denote blanks) the alignments $(a, -, b, c, -, -)$ and $(-, -, a, -, b, c)$ both correspond to the transcription (a, b, c). When the same label appears on successive time-steps in an alignment, the repeats are removed: therefore (a, b, b, b, c, c) and $(a, -, b, -, c, c)$ also correspond to (a, b, c). Denoting by $\mathcal{B}$ an operator that removes first the repeated labels, then the blanks from alignments, and observing that the total probability of an output transcription $\boldsymbol{y}$ is equal to the sum of the probabilities of the alignments corresponding to it, we can write

$$\Pr(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{a} \in \mathcal{B}^{-1}(\boldsymbol{y})} \Pr(\boldsymbol{a}|\boldsymbol{x}) \tag{15}$$

This 'integrating out' over possible alignments is what allows the network to be trained with unsegmented data. The intuition is that, because we don't know where the labels within a particular transcription will occur, we sum over all the places where they *could* occur. Eq. (15) can be efficiently evaluated and differentiated using a dynamic programming algorithm (Graves et al., 2006). Given a target transcription $\boldsymbol{y}^*$, the network can then be trained to minimise the CTC objective function:

$$CTC(\boldsymbol{x}) = -\log \Pr(\boldsymbol{y}^*|\boldsymbol{x}) \tag{16}$$

4. Expected Transcription Loss

The CTC objective function maximises the log probability of getting the sequence transcription completely correct. The relative probabilities of the incorrect transcriptions are therefore ignored, which implies that they are all equally bad. In most cases however, transcription performance is assessed in a more nuanced way. In speech recognition, for example, the standard measure is the word error rate (WER), defined as the edit distance between the true word sequence and the most probable word sequence emitted by the transcriber. We would therefore prefer transcriptions with high WER to be more probable than those with low WER. In the interest of reducing the gap between the objective function and the test criteria, this section proposes a method that allows an RNN to be trained to optimise the expected value of an arbitrary loss function defined over output transcriptions (such as WER).

The network structure and the interpretation of the output activations as the probability of emitting a label (or blank) at a particular time-step, remain the same as for CTC.

Given input sequence $\boldsymbol{x}$, the distribution $\Pr(\boldsymbol{y}|\boldsymbol{x})$ over transcriptions sequences $\boldsymbol{y}$ defined by CTC, and a real-valued transcription loss function $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y})$, the expected transcription loss $\mathcal{L}(\boldsymbol{x})$ is defined as

$$\mathcal{L}(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \Pr(\boldsymbol{y}|\boldsymbol{x}) \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) \tag{17}$$

In general we will not be able to calculate this expectation exactly, and will instead use Monte-Carlo sampling to approximate both $\mathcal{L}$ and its gradient. Substituting Eq. (15) into Eq. (17) we see that

$$\mathcal{L}(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \sum_{\boldsymbol{a} \in \mathcal{B}^{-1}(\boldsymbol{y})} \Pr(\boldsymbol{a}|\boldsymbol{x}) \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) \tag{18}$$

$$= \sum_{\boldsymbol{a}} \Pr(\boldsymbol{a}|\boldsymbol{x}) \mathcal{L}(\boldsymbol{x}, \mathcal{B}(\boldsymbol{a})) \tag{19}$$

Eq. (14) shows that samples can be drawn from $\Pr(\boldsymbol{a}|\boldsymbol{x})$ by independently picking from $\Pr(k, t|\boldsymbol{x})$ at each time-step and concatenating the results, making it straightforward to approximate the loss:

$$\mathcal{L}(\boldsymbol{x}) \approx \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{x}, \mathcal{B}(\boldsymbol{a}^i)), \quad \boldsymbol{a}^i \sim \Pr(\boldsymbol{a}|\boldsymbol{x}) \tag{20}$$

To differentiate $\mathcal{L}$ with respect to the network outputs, first observe from Eq. (13) that

$$\frac{\partial \log Pr(\boldsymbol{a}|\boldsymbol{x})}{\partial \Pr(k, t|\boldsymbol{x})} = \frac{\delta_{\boldsymbol{a}_t k}}{\Pr(k, t|\boldsymbol{x})} \tag{21}$$

Then substitute into Eq. (19), applying the identity $\nabla_x f(x) = f(x) \nabla_x \log f(x)$, to yield:

$$\frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial \Pr(k, t|\boldsymbol{x})} = \sum_{\boldsymbol{a}} \frac{\partial \Pr(\boldsymbol{a}|\boldsymbol{x})}{\partial \Pr(k, t|\boldsymbol{x})} \mathcal{L}(\boldsymbol{x}, \mathcal{B}(\boldsymbol{a}))$$

$$= \sum_{\boldsymbol{a}} \Pr(\boldsymbol{a}|\boldsymbol{x}) \frac{\partial \log \Pr(\boldsymbol{a}|\boldsymbol{x})}{\partial \Pr(k, t|\boldsymbol{x})} \mathcal{L}(\boldsymbol{x}, \mathcal{B}(\boldsymbol{a}))$$

$$= \sum_{\boldsymbol{a}: \boldsymbol{a}_t = k} \Pr(\boldsymbol{a}|\boldsymbol{x}, \boldsymbol{a}_t = k) \mathcal{L}(\boldsymbol{x}, \mathcal{B}(\boldsymbol{a}))$$

This expectation can also be approximated with Monte-Carlo sampling. Because the output probabilities are independent, an unbiased sample $\boldsymbol{a}^i$ from $\Pr(\boldsymbol{a}|\boldsymbol{x})$ can be converted to an unbiased sample from $\Pr(\boldsymbol{a}|\boldsymbol{x}, \boldsymbol{a}_t = k)$ by setting $\boldsymbol{a}^i_t = k$. Every $\boldsymbol{a}^i$ can therefore be used to provide a gradient estimate for every $\Pr(k, t|\boldsymbol{x})$ as follows:

$$\frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial \Pr(k, t|\boldsymbol{x})} \approx \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{x}, \mathcal{B}(\boldsymbol{a}^{i,t,k})) \tag{22}$$

with $\boldsymbol{a}^i \sim \Pr(\boldsymbol{a}|\boldsymbol{x})$ and $\boldsymbol{a}^{i,t,k}_{t'} = \boldsymbol{a}^i_{t'} \forall t' \neq t,\ \boldsymbol{a}^{i,t,k}_t = k$. The advantage of reusing the alignment samples (as opposed to picking separate alignments for every k, t) is that the noise due to the loss variance largely cancels out, and only the *difference* in loss due to altering individual labels is added to the gradient. As has been widely discussed in the policy gradients literature and elsewhere (Peters & Schaal, 2008), noise minimisation is crucial when optimising with stochastic gradient estimates. The $\Pr(k, t|\boldsymbol{x})$ derivatives are passed through the softmax function to give:

$$\frac{\partial \mathcal{L}(\boldsymbol{x})}{\partial y^k_t} \approx \frac{\Pr(k, t|\boldsymbol{x})}{N} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{x}, \mathcal{B}(\boldsymbol{a}^i)) - \mathcal{Z}(\boldsymbol{a}^i, t)$$

where

$$\mathcal{Z}(\boldsymbol{a}^i, t) = \sum_{k'} \Pr(k', t|\boldsymbol{x}) \mathcal{L}(\boldsymbol{x}, \mathcal{B}(\boldsymbol{a}^{i,t,k'}))$$

The derivative added to y^k_t by a given $\boldsymbol{a}^i$ is therefore equal to the difference between the loss with $\boldsymbol{a}^i_t = k$, and the expected loss with $\boldsymbol{a}^i_t$ sampled from $\Pr(k', t|\boldsymbol{x})$. This means the network only receives an error term for changes to the alignment that alter the loss. For example, if the loss function is the word error rate and the sampled alignment yields the character transcription "WTRD ERROR RATE" the gradient would encourage outputs changing the second output label to 'O', discourage outputs making changes to the other two words and be close to zero everywhere else.

For the sampling procedure to be effective, there must be a reasonable probability of picking alignments whose variants receive different losses. The vast majority of alignments drawn from a randomly initialised network will give completely wrong transcriptions, and there will therefore be little chance of altering the loss by modifying a single output. We therefore recommend that expected loss minimisation is used to retrain a network already trained with CTC, rather than applied from the start.

Sampling alignments is cheap, so the only significant computational cost in the procedure is recalculating the loss for the alignment variants. However, for many loss functions (including word error rate) this could be optimised by only recalculating that part of the loss corresponding to the alignment change. For our experiments, five samples per sequence gave sufficiently low variance gradient estimates for effective training.

Note that, in order to calculate the word error rate, an end-of-word label must be used as a delimiter.

5. Decoding

Decoding a CTC network (that is, finding the most probable output transcription $\boldsymbol{y}$ for a given input sequence $\boldsymbol{x}$) can be done to a first approximation by picking the single most probable output at every timestep and returning the corresponding transcription:

$$\text{argmax}_{\boldsymbol{y}} \Pr(\boldsymbol{y}|\boldsymbol{x}) \approx \mathcal{B}\left(\text{argmax}_{\boldsymbol{a}} \Pr(\boldsymbol{a}|\boldsymbol{x})\right)$$

More accurate decoding can be performed with a beam search algorithm, which also makes it possible to integrate a language model. The algorithm is similar to decoding methods used for HMM-based systems, but differs slightly due to the changed interpretation of the network outputs. In a hybrid system the network outputs are interpreted as posterior probabilities of state occupancy, which are then combined with transition probabilities provided by a language model and an HMM. With CTC the network outputs themselves represent transition probabilities (in HMM terms, the label activations are the probability of making transitions into different states, and the blank activation is the probability of remaining in the current state). The situation is further complicated by the removal of repeated label emissions on successive time-steps, which makes it necessary to distinguish alignments ending with blanks from those ending with labels.

The pseudocode in Algorithm 1 describes a simple beam search procedure for a CTC network, which allows the integration of a dictionary and language model. Define $\Pr^-(\boldsymbol{y}, t)$, $\Pr^+(\boldsymbol{y}, t)$ and $\Pr(\boldsymbol{y}, t)$ respectively as the *blank*, *non-blank* and *total* probabilities assigned to some (partial) output transcription $\boldsymbol{y}$, at time t by the beam search, and set $\Pr(\boldsymbol{y}, t) = \Pr^-(\boldsymbol{y}, t) + \Pr^+(\boldsymbol{y}, t)$. Define the *extension probability* $\Pr(k, \boldsymbol{y}, t)$ of $\boldsymbol{y}$ by label k at time t as follows:

$$\Pr(k, \boldsymbol{y}, t) = \Pr(k, t|\boldsymbol{x}) \Pr(k|\boldsymbol{y}) \begin{cases} \Pr^-(\boldsymbol{y}, t-1) \text{ if } \boldsymbol{y}^e = k \\ \Pr(\boldsymbol{y}, t-1) \text{ otherwise} \end{cases}$$

where $\Pr(k, t|\boldsymbol{x})$ is the CTC *emission probability* of k at t, as defined in Eq. (13), $\Pr(k|\boldsymbol{y})$ is the *transition probability* from $\boldsymbol{y}$ to $\boldsymbol{y} + k$ and $\boldsymbol{y}^e$ is the final label in $\boldsymbol{y}$. Lastly, define $\hat{\boldsymbol{y}}$ as the prefix of $\boldsymbol{y}$ with the last label removed, and $\varnothing$ as the empty sequence, noting that $\Pr^+(\varnothing, t) = 0\ \forall t$.

The transition probabilities $\Pr(k|\boldsymbol{y})$ can be used to integrate prior linguistic information into the search. If no

Algorithm 1 CTC Beam Search

Initalise: $B \leftarrow \{\varnothing\}$; $\Pr^-(\varnothing, 0) \leftarrow 1$
for $t = 1 \dots T$ **do**
  $\hat{B} \leftarrow$ the W most probable sequences in B
  $B \leftarrow \{\}$
  for $\boldsymbol{y} \in \hat{B}$ **do**
    if $\boldsymbol{y} \neq \varnothing$ **then**
      $\Pr^+(\boldsymbol{y}, t) \leftarrow \Pr^+(\boldsymbol{y}, t-1) \Pr(\boldsymbol{y}^e, t|\boldsymbol{x})$
      if $\hat{\boldsymbol{y}} \in \hat{B}$ **then**
        $\Pr^+(\boldsymbol{y}, t) \leftarrow \Pr^+(\boldsymbol{y}, t) + \Pr(\boldsymbol{y}^e, \hat{\boldsymbol{y}}, t)$
    $\Pr^-(\boldsymbol{y}, t) \leftarrow \Pr(\boldsymbol{y}, t-1) \Pr(-, t|\boldsymbol{x})$
    Add $\boldsymbol{y}$ to B
    for k = $1 \dots K$ **do**
      $\Pr^-(\boldsymbol{y}+k, t) \leftarrow 0$
      $\Pr^+(\boldsymbol{y}+k, t) \leftarrow \Pr(k, \boldsymbol{y}, t)$
      Add $(\boldsymbol{y}+k)$ to B
Return: $\max_{\boldsymbol{y} \in B} \Pr^{\frac{1}{|\boldsymbol{y}|}}(\boldsymbol{y}, T)$

such knowledge is present (as with standard CTC) then all $\Pr(k|\boldsymbol{y})$ are set to 1. Constraining the search to dictionary words can be easily implemented by setting $\Pr(k|\boldsymbol{y}) = 1$ if $(\boldsymbol{y}+k)$ is in the dictionary and 0 otherwise. To apply a statistical language model, note that $\Pr(k|\boldsymbol{y})$ should represent normalised label-to-label transition probabilities. To convert a word-level language model to a label-level one, first note that any label sequence $\boldsymbol{y}$ can be expressed as the concatenation $\boldsymbol{y} = (\boldsymbol{w} + \boldsymbol{p})$ where $\boldsymbol{w}$ is the longest complete sequence of dictionary words in $\boldsymbol{y}$ and $\boldsymbol{p}$ is the remaining word prefix. Both $\boldsymbol{w}$ and $\boldsymbol{p}$ may be empty. Then we can write

$$\Pr(k|\boldsymbol{y}) = \frac{\sum_{w' \in (\boldsymbol{p}+k)*} \Pr^\gamma(w'|\boldsymbol{w})}{\sum_{w' \in \boldsymbol{p}*} \Pr^\gamma(w'|\boldsymbol{w})} \tag{23}$$

where $\Pr(w'|\boldsymbol{w})$ is the probability assigned to the transition from the word history $\boldsymbol{w}$ to the word w', $\boldsymbol{p}*$ is the set of dictionary words prefixed by $\boldsymbol{p}$ and γ is the language model weighting factor.

The length normalisation in the final step of the algorithm is helpful when decoding with a language model, as otherwise sequences with fewer transitions are unfairly favoured; it has little impact otherwise.

6. Experiments

The experiments were carried out on the Wall Street Journal (WSJ) corpus (available as LDC corpus LDC93S6B and LDC94S13B). The RNN was trained on both the 14 hour subset 'train-si84' and the full 81 hour set, with the 'test-dev93' development set used for validation. For both training sets, the RNN was trained with CTC, as described in Section 3, using the characters in the transcripts as the target sequences. The RNN was then retrained to minimise the expected word error rate using the method from Section 4, with five alignment samples per sequence.

There were a total of 43 characters (including upper case letters, punctuation and a space character to delimit the words). The input data were presented as spectrograms derived from the raw audio files using the 'specgram' function of the 'matplotlib' python toolkit, with width 254 Fourier windows and an overlap of 127 frames, giving 128 inputs per frame.

The network had five levels of bidirectional LSTM hidden layers, with 500 cells in each layer, giving a total of $\sim$ 26.5M weights. It was trained using stochastic gradient descent with one weight update per utterance, a learning rate of 10^{-4} and a momentum of 0.9.

The RNN was compared to a baseline deep neural network-HMM hybrid (DNN-HMM). The DNN-HMM was created using alignments from an SGMM-HMM system trained using Kaldi recipe 's5', model 'tri4b'(Povey et al., 2011). The 14 hour subset was first used to train a Deep Belief Network (DBN) (Hinton & Salakhutdinov, 2006) with six hidden layers of 2000 units each. The input was 15 frames of Mel-scale log filterbanks (1 centre frame ± 7 frames of context) with 40 coefficients, deltas and accelerations. The DBN was trained layerwise then used to initialise a DNN. The DNN was trained to classify the central input frame into one of 3385 triphone states. The DNN was trained with stochastic gradient descent, starting with a learning rate of 0.1, and momentum of 0.9. The learning rate was divided by two at the end of each epoch which failed to reduce the frame error rate on the development set. After six failed attempts, the learning rate was frozen. The DNN posteriors were divided by the square root of the state priors during decoding.

The RNN was first decoded with no dictionary or language model, using the space character to segment the character outputs into words, and thereby calculate the WER. The network was then decoded with a 146K word dictionary, followed by monogram, bigram and trigram language models. The dictionary was built by extending the default WSJ dictionary with 125K words using some augmentation rules implemented into the Kaldi recipe 's5'. The languge models were built on this extended dictionary, using data from the WSJ CD (see scripts 'wsj_extend_dict.sh' and 'wsj_train_lms.sh' in recipe 's5'). The language model weight was optimised separately for all experiments. For the RNN experiments with no linguistic information, and those with only a dictionary, the beam search algorithm in Section 5 was used for decoding. For the RNN experiments with a language model, an alternative method was used, partly due to implementation difficulties and partly to ensure a fair comparison with the baseline system: an N-best list of at most 300 candidate transcriptions was extracted from the baseline DNN-HMM and rescored by the RNN using Eq. (16). The RNN scores were then combined with

Table 1. **Wall Street Journal Results.** All scores are word error rate/character error rate (where known) on the evaluation set. 'LM' is the Language model used for decoding. '14 Hr' and '81 Hr' refer to the amount of data used for training.

SYSTEM	LM	14 HR	81 HR
RNN-CTC	NONE	74.2/30.9	30.1/9.2
RNN-CTC	DICTIONARY	69.2/30.0	24.0/8.0
RNN-CTC	MONOGRAM	25.8	15.8
RNN-CTC	BIGRAM	15.5	10.4
RNN-CTC	TRIGRAM	13.5	8.7
RNN-WER	NONE	74.5/31.3	27.3/8.4
RNN-WER	DICTIONARY	69.7/31.0	21.9/7.3
RNN-WER	MONOGRAM	26.0	15.2
RNN-WER	BIGRAM	15.3	9.8
RNN-WER	TRIGRAM	13.5	8.2
BASELINE	NONE	—	—
BASELINE	DICTIONARY	56.1	51.1
BASELINE	MONOGRAM	23.4	19.9
BASELINE	BIGRAM	11.6	9.4
BASELINE	TRIGRAM	9.4	7.8
COMBINATION	TRIGRAM	—	6.7

the language model to rerank the N-best lists and the WER of the best resulting transcripts was recorded. The best results were obtained with an RNN score weight of 7.7 and a language model weight of 16.

For the 81 hour training set, the oracle error rates for the monogram, bigram and trigram candidates were 8.9%, 2% and 1.4% resepectively, while the anti-oracle (rank 300) error rates varied from 45.5% for monograms and 33% for trigrams. Using larger N-best lists (up to N=1000) did not yield significant performance improvements, from which we concluded that the list was large enough to approximate the true decoding performance of the RNN.

An additional experiment was performed to measure the effect of combining the RNN and DNN. The candidate scores for 'RNN-WER' trained on the 81 hour set were blended with the DNN acoustic model scores and used to rerank the candidates. Best results were obtained with a language model weight of 11, an RNN score weight of 1 and a DNN weight of 1.

The results in Table 1 demonstrate that on the full training set the character level RNN outperforms the baseline model when no language model is present. The RNN retrained to minimise word error rate (labelled 'RNN-WER' to distinguish it from the original 'RNN-CTC' network) performed particularly well in this regime. This is likely due to two factors: firstly the RNN is able to learn a more powerful acoustic model, as it has access to more acoustic context; and secondly it is able to learn an implicit language model from the training transcriptions. However the baseline system overtook the RNN as the LM was strengthened: in this case the RNN's implicit LM may work against it by interfering with the explicit model. Nonetheless the difference was small, considering that so much more prior information (audio pre-processing, pronunciation dictionary, state-tying, forced alignment) was encoded into the baseline system. Unsurprisingly, the gap between 'RNN-CTC' and 'RNN-WER' also shrank as the LM became more dominant.

The baseline system improved only incrementally from the 14 hour to the 81 hour training set, while the RNN error rate dropped dramatically. A possible explanation is that 14 hours of transcribed speech is insufficient for the RNN to learn how to 'spell' enough of the words it needs for accurate transcription—whereas it is enough to learn to identify phonemes.

The combined model performed considerably better than either the RNN or the baseline individually. The improvement of more than 1% absolute over the baseline is considerably larger than the slight gains usually seen with model averaging; this is presumably due to the greater difference between the systems.

7. Discussion

To provide character-level transcriptions, the network must not only learn how to recognise speech sounds, but how to transform them into letters. In other words it must learn how to spell. This is challenging, especially in an orthographically irregular language like English. The following examples from the evaluation set, decoded with no dictionary or language model, give some insight into how the network operates:

target: *TO ILLUSTRATE THE POINT A PROMINENT MIDDLE EAST ANALYST IN WASHINGTON RECOUNTS A CALL FROM ONE CAMPAIGN*
output: *TWO ALSTRAIT THE POINT A PROMINENT MIDILLE EAST ANALYST IM WASHINGTON RECOUNCACALL FROM ONE CAMPAIGN*

target: *T. W. A. ALSO PLANS TO HANG ITS BOUTIQUE SHINGLE IN AIRPORTS AT LAMBERT SAINT*
output: *T. W. A. ALSO PLANS TOHING ITS BOOTIK SINGLE IN AIRPORTS AT LAMBERT SAINT*

target: *ALL THE EQUITY RAISING IN MILAN GAVE THAT STOCK MARKET INDIGESTION LAST YEAR*
output: *ALL THE EQUITY RAISING IN MULONG GAVE THAT STACRK MARKET IN TO JUSTIAN LAST YEAR*

target: *THERE'S UNREST BUT WE'RE NOT GOING TO LOSE THEM TO DUKAKIS*
output: *THERE'S UNREST BUT WERE NOT GOING TO LOSE THEM TO DEKAKIS*

Like all speech recognition systems, the netwok makes phonetic mistakes, such as 'shingle' instead of 'single', and sometimes confuses homophones like 'two' and 'to'. The

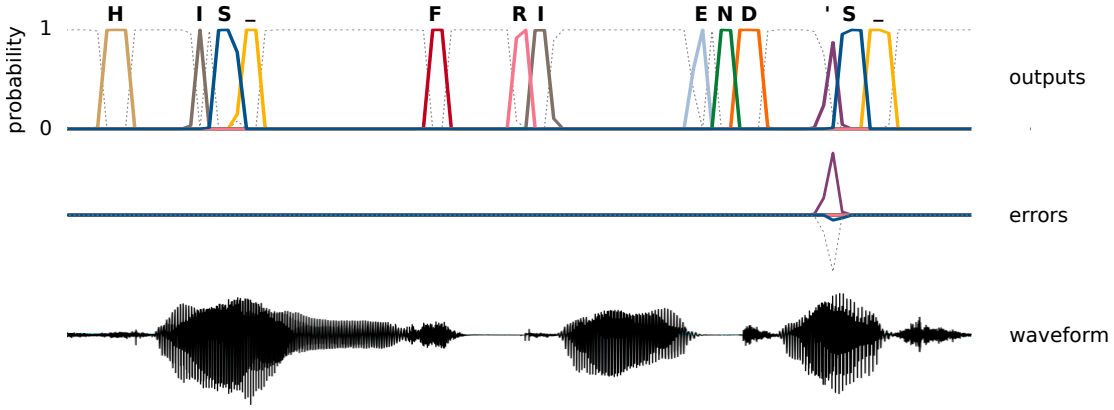

Figure 4. **Network outputs.** The figure shows the frame-level character probabilities emitted by the CTC layer (different colour for each character, dotted grey line for 'blanks'), along with the corresponding training errors, while processing an utterance. The target transcription was 'HIS_FRIENDS_', where the underscores are end-of-word markers. The network was trained with WER loss, which tends to give very sharp output decisions, and hence sparse error signals (if an output probability is 1, nothing else can be sampled, so the gradient is 0 even if the output is wrong). In this case the only gradient comes from the extraneous apostrophe before the 'S'. Note that the characters in common sequences such as 'IS', 'RI' and 'END' are emitted very close together, suggesting that the network learns them as single sounds.

latter problem may be harder than usual to fix with a language model, as words that are close in sound can be quite distant in spelling. Unlike phonetic systems, the network also makes lexical errors—e.g. 'bootik' for 'boutique'—and errors that combine the two, such as 'alstrait' for 'illustrate'.

It is able to correctly transcribe fairly complex words such as 'campaign', 'analyst' and 'equity' that appear frequently in financial texts (possibly learning them as special cases), but struggles with both the sound and spelling of unfamiliar words, especially proper names such as 'Milan' and 'Dukakis'. This suggests that out-of-vocabulary words may still be a problem for character-level recognition, even in the absence of a dictionary. However, the fact that the network can spell at all shows that it is able to infer significant linguistic information from the training transcripts, paving the way for a truly end-to-end speech recognition system.

8. Conclusion

This paper has demonstrated that character-level speech transcription can be performed by a recurrent neural network with minimal preprocessing and no explicit phonetic representation. We have also introduced a novel objective function that allows the network to be directly optimised for word error rate, and shown how to integrate the network outputs with a language model during decoding. Finally, by combining the new model with a baseline, we have achieved state-of-the-art accuracy on the Wall Street Journal corpus for speaker independent recognition.

In the future, it would be interesting to apply the system to datasets where the language model plays a lesser role, such as spontaneous speech, or where the training set is sufficiently large that the network can learn a language model from the transcripts alone. Another promising direction would be to integrate the language model into the CTC or expected transcription loss objective functions during training.

Acknowledgements

The authors wish to thank Daniel Povey for his assistance with Kaldi. This work was partially supported by the Canadian Institute for Advanced Research.

References

Bahl, L., Brown, P., De Souza, P.V., and Mercer, R. Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '86.*, volume 11, pp. 49–52, Apr 1986. doi: 10.1109/ICASSP.1986.1169179.

Bisani, Maximilian and Ney, Hermann. Open vocabulary speech recognition with flat hybrid models. In *INTERSPEECH*, pp. 725–728, 2005.

Bourlard, Herve A. and Morgan, Nelson. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, MA, USA, 1993. ISBN 0792393961.

Ciresan, Dan C., Meier, Ueli, Masci, Jonathan, and Schmidhuber, Jrgen. A committee of neural networks for traffic sign classification. In *IJCNN*, pp. 1918–1921. IEEE, 2011.

Davis, S. and Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366, August 1980.

Eyben, F., Wllmer, M., Schuller, B., and Graves, A. From speech to letters - using a novel neural network architecture for grapheme based asr. In *Proc. Automatic Speech Recognition and Understanding Workshop (ASRU 2009), Merano, Italy*. IEEE, 2009. 13.-17.12.2009.

Galescu, Lucian. Recognition of out-of-vocabulary words with sub-lexical language models. In *INTERSPEECH*, 2003.

Gers, F., Schraudolph, N., and Schmidhuber, J. Learning Precise Timing with LSTM Recurrent Networks. *Journal of Machine Learning Research*, 3:115–143, 2002.

Graves, A. and Schmidhuber, J. Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5-6):602–610, June/July 2005.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *ICML*, Pittsburgh, USA, 2006.

Graves, A., Mohamed, A., and Hinton, G. Speech recognition with deep recurrent neural networks. In *Proc ICASSP 2013*, Vancouver, Canada, May 2013.

Graves, Alex. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012.

Hinton, G. E. and Salakhutdinov, R. R. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George, rahman Mohamed, Abdel, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara, and Kingsbury, Brian. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 2012.

Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Jaitly, Navdeep and Hinton, Geoffrey E. Learning a better representation of speech soundwaves using restricted boltzmann machines. In *ICASSP*, pp. 5884–5887, 2011.

Jaitly, Navdeep, Nguyen, Patrick, Senior, Andrew W, and Vanhoucke, Vincent. Application of pretrained deep neural networks to large vocabulary speech recognition. In *INTERSPEECH*, 2012.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

Lee, Li and Rose, R. A frequency warping approach to speaker normalization. *Speech and Audio Processing, IEEE Transactions on*, 6(1):49–60, Jan 1998.

Peters, J. and Schaal, S. Reinforcement learning of motor skills with policy gradients. In *Neural Networks*, number 4, pp. 682–97, 2008.

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., and Vesely, K. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011.

Schuster, M. and Paliwal, K. K. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing*, 45:2673–2681, 1997.