An online sequence-to-sequence model using partial conditioning

Navdeep Jaitly, Quoc V. Le, Oriol Vinyals, Ilya Sutskever & Samy Bengio
Google Brain
Mountain View, CA 94043, USA
{ndjaitly,qvl,vinyals,ilyasu,bengio}@google.com

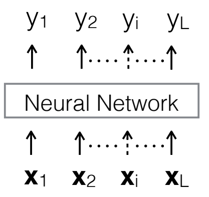
Abstract

Sequence-to-sequence models have achieved impressive results on various tasks. However, they are unsuitable for tasks that require incremental predictions to be made as more data arrives. This is because they generate an output sequence conditioned on an entire input sequence. In this paper, we present a new model that can make incremental predictions as more input arrives, without redoing the entire computation. Unlike sequence-to-sequence models, our method computes the next-step distribution conditioned on the *partial* input sequence observed and the partial sequence generated. It accomplishes this goal using an *encoder* recurrent neural network (RNN) that computes features at the same frame rate as the input, and a *transducer* RNN that operates over blocks of input steps. The transducer RNN extends the sequence produced so far using a local sequence-to-sequence model. During training, our method uses alignment information to generate supervised targets for each block. Approximate alignment is easily available for tasks such as speech recognition, action recognition in videos, etc. During inference (decoding), beam search is used to find the most likely output sequence for an input sequence. This decoding is performed *online* - at the end of each block, the best candidates from the previous block are extended through the local sequence-to-sequence model. On TIMIT, our online method achieves 19.8% phone error rate (PER). For comparison with published sequence-to-sequence methods, we used a bidirectional encoder and achieved 18.7% PER compared to 17.6% from the best reported sequence-to-sequence model. Importantly, unlike sequence-to-sequence our model is minimally impacted by the length of the input. On artificially created longer utterances, it achieves 20.9% with a unidirectional model, compared to 20% from the best bidirectional sequence-to-sequence models.

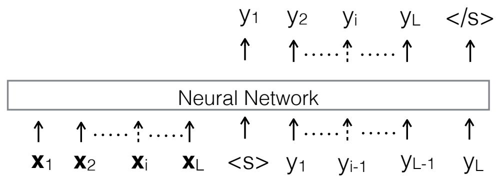
1 Introduction

The recently introduced sequence-to-sequence model has shown tremendous success in various tasks that map sequences to sequences (Sutskever et al., 2014; Cho et al., 2014; Bahdanau et al., 2015a; Chorowski et al., 2014; 2015; Chan et al., 2015; Bahdanau et al., 2015b; Vinyals et al., 2015b;a; Vinyals & Le, 2015). However, this method is unsuitable for tasks where it is important to produce outputs as the input sequence arrives. Speech recognition is an example of such an *online* task – users prefer seeing an ongoing transcription of speech over receiving it at the "end" of an utterance. This limitation of the sequence-to-sequence model is due to the fact that ouput predictions are conditioned on the entire input sequence.

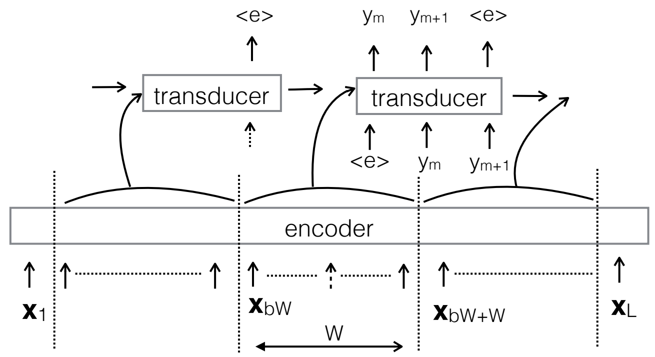
In this paper, we present a modification to sequence-to-sequence models such that they can produce chunks of outputs (possibly of zero length) as blocks of inputs arrive - thus satisfying the condition of being "online" (see Figure 1(c) for an overview). The model generates outputs for each block by using a transducer RNN that implements a sequence-to-sequence model. The inputs to the transducer RNN come from two sources: the encoder RNN and its own recurrent state. In other words, the transducer RNN generates local extensions to the output sequence, conditioned on the features computed for the block by an encoder RNN and the recurrent state of the transducer RNN at the last step of the previous block. Similar to standard sequence-to-sequence models, conditioning over the history of tokens produced in a given block is achieved by feeding in targets of previous steps to the



(a) DNN-HMM



(b) seq2seq



(c) oseq2seq

Figure 1: High-level comparison of our method with other speech recognition models. (a) Hybrid models (e.g., DNN/RNN-HMM and CTC-HMM). (b) Sequence-to-sequence models (Chan et al., 2015; Bahdanau et al., 2015b). (c) The online sequence-to-sequence model (this paper) uses sequence-to-sequence models per block, and transfers hidden state of the transducer across blocks.

transducer. Conditioning over the history of output tokens is achieved by the transducer, through the use of its recurrent connections across blocks. Conditioning on all input seen so far is achieved through the propagation of information through the recurrent states of the encoder RNN. This partial conditioning on the input sequence seen and output sequence generated so far is what makes the model different from sequence-to-sequence models. Partial conditioning allows this model to produce output tokens as data arrives, instead of waiting until all the input sequence has arrived.

During training, the model assumes that approximate alignments of output tokens to the input sequence are given. The model uses the alignments to group outputs into blocks. An end-of-block symbol ($<e>$) suffix is added to each local target sequence. This allows the model to learn to terminate the sequence produced for a block. During inference, the emission of the end-of-block symbol causes the transducer to stop generating more tokens for the current block and to move to the next block to generate extensions. The need for approximate alignment is not a big hurdle for many domains. For example, in speech recognition, alignments can be easily generated by bootstrapping from a different model. Our experiments indicate that these alignments do not need to be very precise - the transducer is able to use its recurrent state to deal with outputs being on different sides of block boundaries.

We experimented with several variations of sequence-to-sequence models and found attention-based models to be most useful. In our model, the transducer uses attention over the features within each block, to generate the next output token. In order to learn a model of how attention changes from one output step to the next we also experimented with a new attention mechanism that uses a Long Short Term Memory (LSTM) recurrent neural network (Hochreiter & Schmidhuber, 1997). This attention mechanism is different from that of Chorowski et al. (2015) in that, instead of taking a softmax over attention scalars, we input these values into an LSTM-RNN which produces the attention vector as its output. The LSTM can thus learn to modulate attention at one output step, using the attention generated at the previous output step.

We applied the model to the phone recognition task on the TIMIT dataset. Using this method, we achieved 19.8% PER on TIMIT using three layers of 250 LSTM cells in both the encoder and the transducer along with the LSTM-based attention model mentioned above. Unlike previous work on sequence-to-sequence models this model uses only unidirectional LSTMs for both encoder and transducer networks, and decoding can be performed online by extending beams from one block of inputs to the next. Further, the model trains relatively easily without careful regularization.

While prior work on sequence-to-sequence models have not reported results on a unidirectional setting, the above performance matches the 19.6% PER achieved by Connectionist Temporal Classification (CTC) with unidirectional LSTMs for the same task (Graves et al., 2013a). However, CTC makes independent predictions at each step (see figure 1(a)), and as such can only produce high accuracy on large vocabulary continuous speech recognition (LVCSR) tasks when it is coupled to a strong language model and pronounciation dictionary (Bahdanau et al., 2015b). Our model is inherently an end-to-end model that can incorporate a language model within its parameters, and should produce accurate results on LVCSR even without strong language models.

For a comparison with previously reported sequence-to-sequence results on TIMIT, we used a bidirectional encoder and achieved 18.7% PER compared to the best reported number of 17.6% achieved by Chorowski et al. (2015). [1] Even with the bidirectional encoder, our model is different from the sequence-to-sequence model in that it uses a slightly different training objective. Our objective function is geared towards generating partial outputs by the ends of blocks. The partial outputs are provided to the model using alignment information and may not be the best for state of the art performance. Nevertheless, the comparable performance of these two models in spite of these, and other architectural differences between the models imply that little is lost by training towards partial outputs from blocks.

2 Related Work

In the last few years, multiple deep learning architectures have been proposed for speech recognition. Figure 1 shows the comparison of our work against these previous approaches.

The predominant methodology for speech recognition until recently was to use a Hidden Markov Model coupled with a Deep Neural Network, a Convolutional Network or a Recurrent Network (Hinton et al., 2012; Abdel-Hamid et al., 2012; Graves et al., 2013b). More recently, a CTC approach was also proposed for this task (Graves et al., 2013a; Hannun et al., 2014; Sak et al., 2015). An important aspect of these approaches is that the model makes predictions at every input time step. A high-level picture of this architecture is shown in Figure 1(a). A weakness of these models is that they typically assume conditional independence between the predictions at each output step.

Sequence-to-sequence models make no such assumptions – the output sequence is generated by next step prediction, conditioning on the entire input sequence and the partial output sequence generated so far (Chorowski et al., 2014; 2015; Bahdanau et al., 2015b; Chan et al., 2015). Figure 1(b) shows the high-level picture of this architecture. As can be seen from the figure, these models have a significant limitation in that they have to wait until the end of the speech utterance to start decoding. This property makes them unattractive for real time speech recognition.

Conceptually, speech is ordered left-to-right, and localization of relevant frames using attention over the entire input seems to be overkill. Moreover, it has been empirically observed that these models perform significantly worse on longer inputs – presumably because inacuracies in attention at one step can negatively impact the entire transcripts produced subsequently (Chan et al., 2015). Chorowski et al. (2015) ameliorate (but not eliminate) this problem by introducing a heuristic that requires the attention to move from one step to the next. However, this is still a heuristic, and their published model reports results on inputs from a bidirectional encoder (Schuster & Paliwal, 1997), which prevents the system from being able to do online decoding.

Our model on the other hand uses alignment information such that it can decode online, extending transcripts at the ends of blocks. Alignments of speech units (phones or words or even phrases) to segments of speech are easily obtain with orthogonal methods such as manual labeling or performing

[1]Note that the use of a bidirectional encoder here makes both models offline, since bidirectional features needed by the transducer cannot be computed until the end of the utterance.

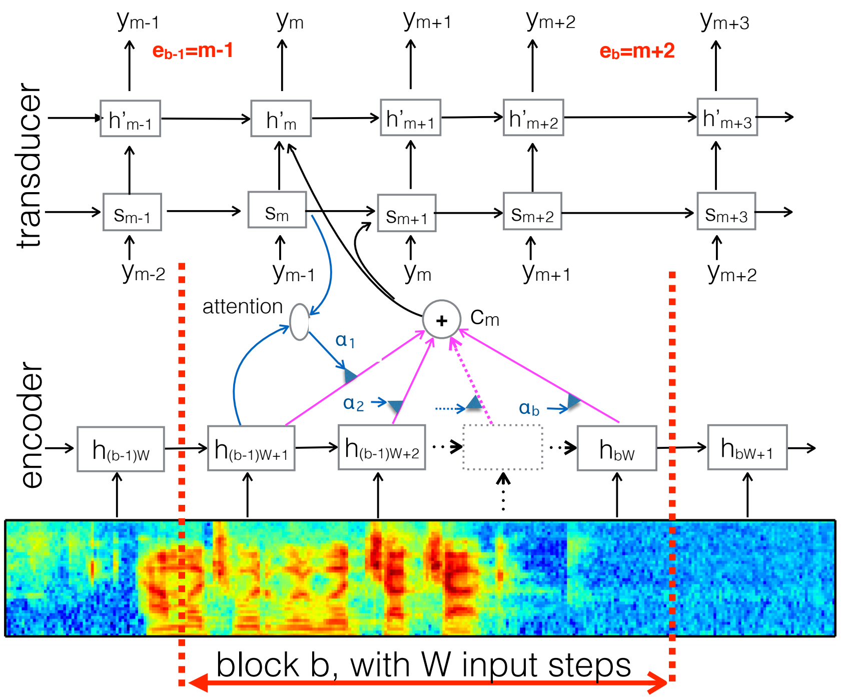


Figure 2: An overview of the online sequence-to-sequence architecture for speech. The input acoustic sequence is processed by the encoder to produce hidden state vectors $\mathbf{h}_i$ at each time step i, $i = 1 \cdots L$. The transducer runs at a different frame rate, producing tokens at the end of each block of W input steps, using next step prediction, using not only input from the encoder, but the label sequence emitted so far. For each block, the transducer produces the tokens that were known to be emitted between the end of the current block and the end of the last block (the number of tokens produced is variable and depends on the alignment input to the model). The transducer maintains its state across the blocks through the use of recurrent connections to the previous time steps and blocks. The figure above shows the transducer producing tokens for block b. The subsequence emitted in this block is $y_m y_{m+1} y_{m+2}$.

forced alignment with an existing model. Using this information we can train a model that uses the sequence-to-sequence model with partial conditioning on inputs seen and outputs produced, to extend output subsequences as input arrives. The blocked architecture also reduces the impact of the length of the input on the accuracy of the transcript, as each new block is relatively unaffected by the errors in the earlier blocks. Figure 1(c) shows the difference between our method and previous sequence-to-sequence models.

Note that our model does not require frame level alignments that are used by the DNN/RNN-HMM systems. Instead it requires approximate alignments that determine which tokens to emit for a block. Moreover, although we only performed phone recognition experiments for this paper, like Bahdanau et al. (2015b); Chan et al. (2015) the model can use words or phrases as tokens, and can be trained as an end-to-end model.

3 METHODS

In this section we describe the model in more detail. Please refer to Figure 2 for an overview.

3.1 MODEL

Let $\mathbf{x}_{1\cdots L}$ be the input data that is L time steps long, where $\mathbf{x}_i$ represents the features at input time step i. Correspondingly, let $y_{1\cdots S}$ be the one-hot encoding of the target sequence of length S, from a dictionary of K targets. Let W be the block size, i.e., the periodicity with which the transducer emits output tokens, and $N = \lceil \frac{L}{W} \rceil$ be the number of blocks, $e_b, b \in 1 \cdots N$ be the index of the last token emitted in the b^{th} block. Note that $e_0 = 0$ and $e_N = S$. We append the $<e>$ token to each block of outputs. Thus $y_{e_b} = <e>$ for each block b.

We compute the probability of seeing output sequence $y_{1 \cdots e_b}$ by the end of block b as follows:

$$p\left(y_{1\cdots e_b}|\mathbf{x}_{1\cdots bW}\right) = p\left(y_{1\cdots e_1}|\mathbf{x}_{1\cdots W}\right) \prod_{b'=2}^{b} p\left(y_{\left(e_{b'-1}+1\right)\cdots e'_b}|\mathbf{x}_{1\cdots b'W}, y_{1\cdots e_{b'-1}}\right) \tag{1}$$

Each of the terms in this equation is itself computed by the chain rule decomposition, i.e., for any block b,

$$p\left(y_{(e_{b-1}+1)\cdots e_b}|\mathbf{x}_{1\cdots bW}, y_{1\cdots e_{b-1}}\right) = \prod_{m=e_{(b-1)}+1}^{e_b} p\left(y_m|\mathbf{x}_{1\cdots bW}, y_{1\cdots(m-1)}\right) \tag{2}$$

The next step probability terms, $p\left(y_m|\mathbf{x}_{1\cdots bW}, y_{1\cdots(m-1)}\right)$, in Equation 2 are computed by the transducer using the encoding of the input $\mathbf{x}_{1\cdots bW}$ computed by the encoder, and the label prefix $y_{1\cdots(m-1)}$ that was input into the transducer, at previous emission steps. We describe this in more detail in the next subsection.

3.2 NEXT STEP PREDICTION

We again refer the reader to Figure 2 for this discussion. The example shows a transducer with two hidden layers, with units $\mathbf{s}_m$ and $\mathbf{h'}_m$ at output step m. In the figure, the next step prediction is shown for block b. For this block, the index of the first output symbol is $m = e_{b-1} + 1$, and the index of the last output symbol is $m + 2$ (i.e. $e_b = m + 2$).

The transducer computes the next step prediction, using parameters, θ, of the neural network through the following sequence of steps:

$$\begin{aligned}
\mathbf{s}_m &= f_{RNN}\left(\mathbf{s}_{m-1}, \left[\mathbf{c}_{m-1}; y_{m-1}\right]; \theta\right) && (3)\\
\mathbf{c}_m &= f_{context}\left(\mathbf{s}_m, \mathbf{h}_{((b-1)W+1)\cdots bW}; \theta\right) && (4)\\
\mathbf{h'}_m &= f_{RNN}\left(\mathbf{h'}_{m-1}, \left[\mathbf{c}_m; s_m\right]; \theta\right) && (5)\\
p\left(y_m|\mathbf{x}_{1\cdots bW}, y_{1\cdots(m-1)}\right) &= f_{softmax}\left(y_m; \mathbf{h'}_m, \theta\right) && (6)
\end{aligned}$$

where $f_{RNN}\left(\mathbf{a_{m-1}}, \mathbf{b_m}; \theta\right)$ is the recurrent neural network function (such as an LSTM or a sigmoid or tanh RNN) that computes the state vector $\mathbf{a}_m$ for a layer at a step using the recurrent state vector $\mathbf{a}_{m-1}$ at the last time step, and input $\mathbf{b}_m$ at the current time step;[2] $f_{softmax}\left(\cdot; \mathbf{a}_m; \theta\right)$ is the softmax distribution computed by a softmax layer, with input vector $\mathbf{a_m}$; and $f_{context}\left(\mathbf{s}_m, \mathbf{h}_{((b-1)W+1)\cdots bW}; \theta\right)$ is the context function, that computes the input to the transducer at output step m from the state s_m at the current time step, and the features $\mathbf{h}_{((b-1)W+1)\cdots bW}$ of the encoder for the current input block, b. We experimented with different ways of computing the context vector – with and without an attention mechanism. These are described subsequently in section 3.3.

Note that since the encoder is an RNN, $\mathbf{h}_{(b-1)W\cdots bW}$ is actually a function of the entire acoustic input, $\mathbf{x}_{1\cdots bW}$ so far. Correspondingly, $\mathbf{s}_m$ is a function of the labels emitted so far, and the entire acoustic signal seen so far.[3] Similarly, $\mathbf{h'}_m$ is a function of the labels emitted so far and the entire acoustic signal seen so far.

3.3 COMPUTING $f_{context}$

We first describe how the context vector is computed by an attention model similar to earlier work (Chorowski et al., 2014; Bahdanau et al., 2015a; Chan et al., 2015). We call this model the MLP-attention model.

[2]Note that for LSTM, we would have to additionally factor in cell states from the previous states - we have ignored this in the notation for purpose of clarity. The exact details are easily worked out.

[3]For the first output step of a block it includes only the acoustic signal seen until the end of the last block.

In this model the context vector $\mathbf{c}_m$ is in computed in two steps - first a normalized attention vector α_m is computed from the state $\mathbf{s}_m$ of the transducer and next the hidden states $\mathbf{h}_{(b-1)W+1\cdots bW}$ of the encoder for the current block are linearly combined using α and used as the context vector. To compute α_m, a multi-layer perceptron computes a scalar value, e_j^m for each pair of transducer state $\mathbf{s}_m$ and encoder $\mathbf{h}_{(b-1)W+j}$. The attention vector is computed from the scalar values, e_j^m, $j = 1 \cdots W$. Formally:

$$e_j^m = f_{attention}\left(\mathbf{s}_m, \mathbf{h}_{(b-1)W+j}; \theta\right) \tag{7}$$

$$\alpha_m = softmax\left([e_1^m; e_2^m; \cdots e_W^m]\right) \tag{8}$$

$$\mathbf{c}_m = \sum_{j=1}^{W} \alpha_j^m \mathbf{h}_{(b-1)W+j} \tag{9}$$

We also experimented with using a simpler model for $f_{attention}$ that computed $e_j^m = \mathbf{s}_m^T \mathbf{h}_{(b-1)W+j}$. We refer to this model as DOT-attention model.

Both of these attention models have two shortcomings. Firstly there is no explicit mechanism that requires the attention model to move its focus forward, from one output time step to the next. Secondly, the energies computed as inputs to the softmax function, for different input frames j are independent of each other at each time step, and thus cannot modulate (e.g., enhance or suppress) each other, other than through the softmax function.

We attempt to address these two shortcomings using a new attention mechanism. In this model, instead of feeding $[e_1^m; e_2^m; \cdots e_W^m]$ into a softmax, we feed them into a recurrent neural network with one hidden layer that outputs the softmax attention vector at each time step. Thus the model should be able to modulate the attention vector both within a time step and across time steps. We refer to this model as LSTM-attention.

3.4 Addressing End of Blocks

Since the model only produces a small sequence of output tokens in each block, we have to address the mechanism for shifting the transducer from one block to the next. We experimented with three distinct ways of doing this. In the first approach, we introduced no explicit mechanism for end-of-blocks, hoping that the transducer neural network would implicitly learn a model from the training data. In the second approach we added end-of-block symbols, <e>, to the label sequence to demarcate the end of blocks, and we added this symbol to the target dictionary. Thus the softmax function in Equation 6 implicitly learns to either emit a token, or to move the transducer forward to the next block. In the third approach, we model moving the transducer forward, using a separate logistic function of the attention vector. The target of the logistic function is 0 or 1 depending on whether the current step is the last step in the block or not.

3.5 Training

The parameters, θ, of the model are learned by performing stochastic gradient descent (SGD) with momentum over the loss function,

$$l\left(\theta; \mathbf{x}_{1\cdots L}, y_{1\cdots S}\right) = \sum_{b=1}^{B} \sum_{m=1}^{S} \log p\left(y_{(e_{b-1}+1)\cdots e_b} | \mathbf{x}_{1\cdots bW}, y_{1\cdots(e_{b-1}+1)}\right) \tag{10}$$

3.6 Inference

For inference, given the input acoustics $\mathbf{x}_{1\cdots L}$, and the model parameters, θ, we find the sequence of labels $y_{1..M}$ that maximizes the probability of the labels, conditioned on the data, i.e.,

$$\tilde{y}_{1\cdots S} = \arg \max_{y_{1\cdots S'}, e_{1\cdots N}} \sum_{b=1}^{N} \log p\left(y_{e_{(b-1)+1}\cdots e_b} | \mathbf{x}_{1\cdots bW}, y_{1\cdots e_{(b-1)}}\right) \tag{11}$$

Exact inference in this scheme is computationally expensive because the expression for log probability does not permit decomposition into smaller terms that can be independently computed. Instead, each candidate, $y_{1 \cdot S'}$, would have to be tested independently, and the best sequence over an exponentially large number of sequences would have to be discovered. Hence, we use a beam search heuristic to find the "best" set of candidates. To do this, at each output step m, we keep a heap of alternative n best prefixes, and extend each one by one symbol, trying out all the possible alternative extensions, keeping only the best n extensions. Included in the beam search is the act of moving the attention to the next input block. The beam search ends either when the sequence is longer than a pre-specified threshold, or when the end of token symbol is produced at the last block.

An alternative way of explaining inference that highlights its online property is as follows. Beam search uses a heap of n candidates at each step. For each new block, we extend each of the candidates from the last block, one symbol at a time, and add it back to the heap. A candidate is no longer extended in a block once it emits the $<e>$ token. When none of the candidates in the heap can be extended any further we move to the next block and repeat the inference step. Note that the candidate sequences can be extended without redoing any of the computations at the earlier blocks by storing the state of the transducer at the last step for each of the candidates.

4 Experiments and Results

We used TIMIT, a standard bechmark for speech recognition (Fisher et al., 1986) for our experiments. Log Mel filterbanks were computed every 10ms as inputs to the system. The targets were the 60 phones defined for the TIMIT dataset (the demarcating *h#* were relabelled as *pau* and merged with neighbouring *pau* if one existed). Alignments were generated from Kaldi toolkit with a simple Gaussian Mixture Model-Hidden Markov Model (GMM-HMM) trained on the data. The frame level alignments from Kaldi were converted into block level alignments by assigning each phone in the sequence to the block it was last observed in.

To train our model we use stochastic gradient descent with momentum with a batch size of 1 (i.e., one utterance per training step), starting with a learning rate of 0.05, and momentum of 0.9. We reduced the learning rate by a factor of 0.5 every time the average log prob over the validation set decreased.[4] The decrease was applied for a maximum of 4 times. The models were trained for 50 epochs and the parameters from the epochs with the best dev set log prob were used for decoding.

Table 1 shows a comparison of our method against a basic implementation of a sequence-to-sequence model that produces outputs for each block independent of the other blocks, and concatenates the produced sequences. Here, the sequence-to-sequence model produces the output conditioned on the state of the encoder at the end of the block. Both models used an encoder with two layers of 250 LSTM cells, without attention. The standard sequence-to-sequence model performs significantly worse than our model – the recurrent connections of the transducer across blocks are clearly helpful in improving the accuracy of the model.

Table 1: Impact of maintaining recurrent state of transducer across blocks on the PER. This table shows that maintaining the state of the transducer across blocks leads to much better results. For this experiment, a block size (W) of 15 frames was used. The reported number is the median of three different runs.

W	**BLOCK-RECURRENCE**	**PER**
15	No	34.3
15	Yes	20.6

Figure 3 shows the impact of block size on the accuracy of the different sequence-to-sequence variations we used. See Section 3.3 for a description of the {DOT,MLP,LSTM}-attention models. All models used a two LSTM layer encoder and a two LSTM layer transducer. As expected, sequence-to-sequence models without attention are very sensitive to the choice of the block size, but with attention, the precise value of the block size becomes less important. Our results also indicate that the LSTM-based attention model generally performs better than the other attention mechanisms we explored. Since this model performed best with W=25, we used this configuration for subsequent experiments.

[4]Note the TIMIT provides a validation set, called the *dev* set. We will use these terms interchangeably.

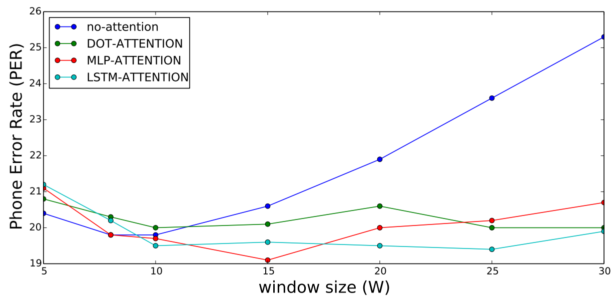


Figure 3: Impact of the number of frames (W) in a block and attention mechanism on PER. Figure shows the PER on the validation set as function of W, with and without the use of different types of attention mechanisms. Each number is the median value from three experiments.

We combined our model with scheduled sampling (Bengio et al., 2015) but found no gains on this task. This is presumably because the regularizing effect of scheduled sampling is observed over longer sequences while our transducer outputs only a small number of tokens per block.

We also carried out experiments to see the impact of offsetting the labels from the input, so that the model was required to produce labels only after several time steps of seeing the input. Surprisingly, the model's accuracy was not impacted much by this choice. However, the model performed slightly worse (approximately 3% relative) if targets that straddled boundaries were randomly assigned to a side based on its length on that side. Thus the model does prefer to receive the relevant input before emitting the target; but beyond that, it does not require additional offset.

To improve the PER of the trained models, we used the PER on the validation set at the end of each epoch of training to decide whether to reduce the learning rate or not. We explored the impact of number of layers in the encoder and in the transducer (see Table 2). We found that a three layer encoder coupled to a three layer transducer performed best. Four layer transducers produced results with higher spread in accuracy - possibly because of the more difficult optimization involved. Using this, we were able to achieve a PER of **19.8%** on the TIMIT test set. These results could probably be improved with dropout training (Zaremba et al., 2014) or other regularization techniques (Chorowski et al., 2015) but we did not pursue those avenues in this paper.

Table 2: Impact of architecture on PER. Table shows the PER on the dev set as function of the number of layers (2 or 3) in the encoder and the number of layers in the transducer (1-4).

# of layers in encoder / transducer	**1**	**2**	**3**	**4**
2		19.2	18.9	18.8
3		18.5	**18.2**	19.4

For a comparison with previously published sequence-to-sequence models on this task, we used a three layer bidirectional LSTM encoder with 250 LSTM cells in each direction and achieved a PER of **18.7%**. By contrast, the best reported results using previous sequence-to-sequence models are 17.6% (Chorowski et al., 2015). However, this result comes with more careful training techniques than we attempted for this work.

We also explored the impact of the length of utterances on PER. Figure 4 shows that the PER of utterances is relatively consistenct across different lengths. This is very different from the results reported by sequence-to-sequence methods (Chan et al., 2015) that report strong decrease in accuracy of results as a function of length of utterances. Chorowski et al. (2014) reported a means of measuring this impact, by computing the PER of a dataset made by concatenating repetitions of the same utterance. They reported an increase in PER from 17.6% to 20% on concatenating each utterance ten times. By contrast, we find that our unidirectional online sequence-to-sequence model achieves 20.9% (up from 19.8%) PER on the ten-times replicated dataset. The bidirectional transducer meanwhile goes from 18.7% PER to 19.7%.

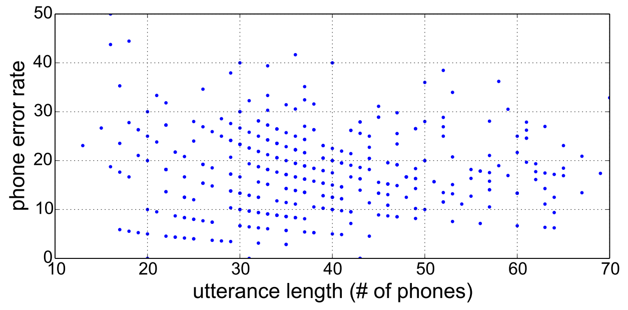


Figure 4: Impact of length of output sequence (# of phone) on PER. Figure shows the PER of different utterances in the dev set as function of the number of phones in the target utterance.

5 Discussion

Our model simplifies the earlier approach of Chorowski et al. (2014) by using alignment information in the data. However, note that unlike DNN-HMM models, the alignment used here is coarse in nature and does not use precise frame labeling of inputs. Instead, it uses coarse labels that indicate which tokens were emitted by the time a particular block of input was processed. As the alignment corresponds to block level information and not frame level alignments, it could include labels such as what words were verbalized by what block of the input data. The use of a transducer RNN that processes input at a slower frame rate reduces the importance of precise alignments.

One of the important side-effects of our model using partial conditioning with a blocked transducer is that it naturally alleviates the problem of "losing attention" suffered by sequence-to-sequence models. Because of this, sequence-to-sequence models perform worse on longer utterances (Chorowski et al., 2015; Chan et al., 2015). This problem is automatically tackled in our model because each new block automatically shifts the attention monotonically forward. Within a block, the model learns to move attention forward from one step to the next, and the attention mechanism rarely suffers, because both the size of a block, and the number of output steps for a block are relatively small. As a result, error in attention in one block, has minimal impact on the predictions at subsequent blocks.

Finally, we note that increasing the block size, W, so that it is as large as the input utterance makes the model similar to other end-to-end models (Chorowski et al., 2014; Chan et al., 2015).

6 Conclusion

We have introduced a new model that uses partial conditioning on inputs to generate output sequences. Using a new attention mechanism, we applied the model to a phone recognition task and showed that is can produce results comparable to the state of the art from sequence-to-sequence models. The model tackles the two main hurdles to the adoption of sequence-to-sequence models as speech recognition systems. Firstly, the model can produce output transcripts as data arrives, without having to redo the entire decoding. Secondly, because of its blocked architecture, problems with attention in one part of the input are unlikely to impact the entire transcript – reducing the impact of the length of the input sequence on the accuracy of the transcript. Future work will aim to apply this method for end-to-end training on a larger corpus.

Acknowledgments

We thank George Dahl, Andrew Senior, Geoffrey Hinton, Erik McDermott, Tara Sainath, Vincent Vanhoucke and the Google Brain team for the help with the project. We thank William Chan for fruitful discussions on sequence-to-sequence models.

REFERENCES

Abdel-Hamid, Ossama, Mohamed, Abdel-rahman, Jiang, Hui, and Penn, Gerald. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pp. 4277–4280. IEEE, 2012.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. In *International Conference on Learning Representations*, 2015a.

Bahdanau, Dzmitry, Chorowski, Jan, Serdyuk, Dmitriy, Brakel, Philemon, and Bengio, Yoshua. End-to-end attention-based large vocabulary speech recognition. In *http://arxiv.org/abs/1508.04395*, 2015b.

Bengio, Samy, Vinyals, Oriol, Jaitly, Navdeep, and Shazeer, Noam. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *Neural Information Processing Systems*, 2015.

Chan, William, Jaitly, Navdeep, Le, Quoc V, and Vinyals, Oriol. Listen, attend and spell. *arXiv preprint arXiv:1508.01211*, 2015.

Cho, Kyunghyun, van Merrienboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwen, Holger, and Bengio, Yoshua. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014.

Chorowski, Jan, Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. End-to-end Continuous Speech Recognition using Attention-based Recurrent NN: First Results. In *Neural Information Processing Systems: Workshop Deep Learning and Representation Learning Workshop*, 2014.

Chorowski, Jan, Bahdanau, Dzmitry, Serdyuk, Dmitriy, Cho, Kyunghyun, and Bengio, Yoshua. Attention-Based Models for Speech Recognition. In *Neural Information Processing Systems*, 2015.

Fisher, William M, Doddington, George R, and Goudie-Marshall, Kathleen M. The darpa speech recognition research database: specifications and status. In *Proc. DARPA Workshop on speech recognition*, pp. 93–99, 1986.

Graves, Alan, Mohamed, Abdel-rahman, and Hinton, Geoffrey. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 6645–6649. IEEE, 2013a.

Graves, Alex, Jaitly, Navdeep, and Mohamed, Abdel-rahman. Hybrid Speech Recognition with Bidirectional LSTM. In *Automatic Speech Recognition and Understanding Workshop*, 2013b.

Hannun, Awni, Case, Carl, Casper, Jared, Catanzaro, Bryan, Diamos, Greg, Elsen, Erich, Prenger, Ryan, Satheesh, Sanjeev, Sengupta, Shubho, Coates, Adam, et al. Deepspeech: Scaling up end-to-end speech recognition. In *http://arxiv.org/abs/1412.5567*, 2014.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E., Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N., and Kingsbury, Brian. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

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

Sak, Hasim, Senior, Andrew W., Rao, Kanishka, and Beaufays, Françoise. Fast and accurate recurrent neural network acoustic models for speech recognition. *CoRR*, abs/1507.06947, 2015. URL `http://arxiv.org/abs/1507.06947`.

Schuster, Mike and Paliwal, Kuldip K. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to Sequence Learning with Neural Networks. In *Neural Information Processing Systems*, 2014.

Vinyals, Oriol and Le, Quoc V. A neural conversational model. In *ICML Deep Learning Workshop*, 2015.

Vinyals, Oriol, Kaiser, Lukasz, Koo, Terry, Petrov, Slav, Sutskever, Ilya, and Hinton, Geoffrey. Grammar as a foreign language. In *NIPS*, 2015a.

Vinyals, Oriol, Toshev, Alexander, Bengio, Samy, and Erhan, Dumitru. Show and Tell: A Neural Image Caption Generator. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015b.

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014. URL `http://arxiv.org/abs/1409.2329`.