Improving LSTM-CTC based ASR performance in domains with limited training data

Jayadev Billa*
jbilla2004@gmail.com

Abstract

This paper addresses the observed performance gap between automatic speech recognition (ASR) systems based on Long Short Term Memory (LSTM) neural networks trained with the connectionist temporal classification (CTC) loss function and systems based on hybrid Deep Neural Networks (DNNs) trained with the cross entropy (CE) loss function on domains with limited data. We step through a number of experiments that show incremental improvements on a baseline EESEN toolkit based LSTM-CTC ASR system trained on the Librispeech 100hr (`train-clean-100`) corpus. Our results show that with effective combination of data augmentation and regularization, a LSTM-CTC based system can exceed the performance of a strong Kaldi based baseline trained on the same data.

1 Introduction

Long Short Term Memory (LSTM), and in general, recurrent neural network (RNN) based ASR systems [1–5] trained with connectionist temporal classification (CTC) [6] have recently been shown to work extremely well when there is an abundance of training data, matching and exceeding the performance of hybrid DNN systems [2, 5]. However, these systems lag comparable hybrid DNN systems when trained on smaller training sets (e.g., discussion in [7]). LSTM-CTC systems are much simpler to train; they can be trained as an end-to-end speech recognition system as opposed to the iterative multi-process approach to hybrid DNN ASR system training. In this paper, we describe methods that eliminate the performance gap between these two approaches and thus provide a path to wider usage of the simpler LSTM-CTC approach in domains with lower amounts of available training data.

The work we present, in the following sections, encompasses model initialization, data augmentation, feature stacking and striding, and regularization via dropout. In particular, we show that a combination of these approaches significantly improves performance. We will first describe the key elements of our baseline EESEN toolkit [8] system, then describe our efforts in the listed areas. Our results show that an LSTM-CTC based speech recognition can exceed the performance of a strong Kaldi toolkit [9] baseline trained on the same data.

2 Baseline LSTM-CTC ASR system

Our baseline system is based on the publicly available EESEN Toolkit [8] trained on the publicly available Librispeech corpus [10]. The acoustic model in EESEN is a deep bidirectional LSTM neural network. In this paper, an LSTM model will always imply a bidirectional LSTM model.

A typical LSTM cell is illustrated in Figure 1. The LSTM cell consists of three gates: input, forget, and output, which control, respectively, what fraction of the input is passed to the "memory" cell, what fraction of the stored cell memory is retained, and what fraction of the cell memory is output.

*Author is currently unaffiliated.

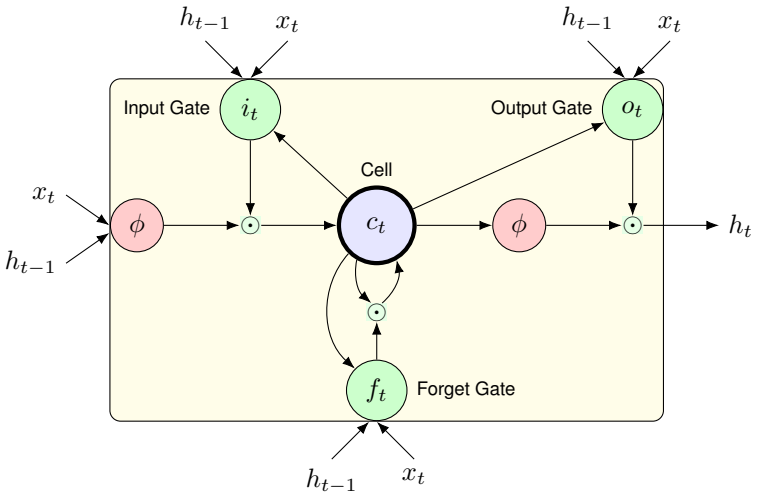

Figure 1: LSTM Memory cell.

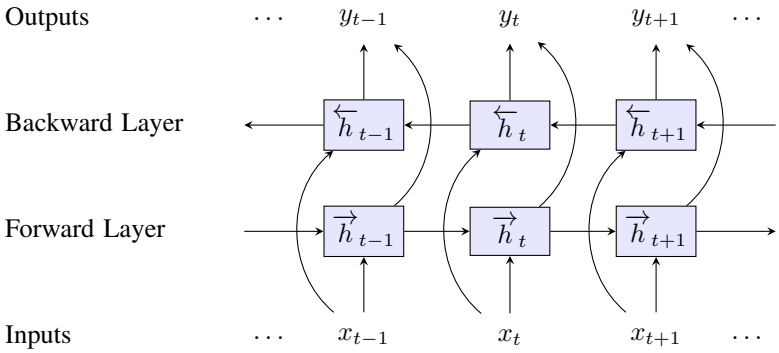

Figure 2: Bidirectional LSTM Layer, boxes represent a single LSTM memory cell.

The vector formulas that describe the LSTM cell are

$$\mathbf{i}_t = \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{R}_i\mathbf{h}_{t-1} + \mathbf{P}_i\mathbf{c}_{t-1} + \mathbf{b}_i) \quad \textit{input gate} \quad (1)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{R}_f\mathbf{h}_{t-1} + \mathbf{P}_f\mathbf{c}_{t-1} + \mathbf{b}_f) \quad \textit{forget gate} \quad (2)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \phi(\mathbf{W}_c\mathbf{x}_t + \mathbf{R}_c\mathbf{h}_{t-1} + \mathbf{b}_c) \quad \textit{cell state} \quad (3)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{R}_o\mathbf{h}_{t-1} + \mathbf{P}_o\mathbf{c}_t + \mathbf{b}_o) \quad \textit{output gate} \quad (4)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \phi(\mathbf{c}_t) \quad \textit{output} \quad (5)$$

where $\mathbf{x}_t$ is the input vector at time t, $\mathbf{W}$ are rectangular input weight matrices connecting inputs to the LSTM cell, $\mathbf{R}$ are square recurrent weight matrices connecting the previous memory cell state to the LSTM cell, $\mathbf{P}$ are diagonal peephole weight matrices and $\mathbf{b}$ are bias vectors. Functions σ and ϕ are the *logistic sigmoid* ($\frac{1}{1+e^{-x}}$) and *hyperbolic tangent* nonlinearities respectively. Operator $\odot$ represents the point-wise multiplication of two vectors.

These cells are then arranged into a bidirectional layer where data is processed independently in the forward and backward directions [11]. The outputs from both forward and backward directions are concatenated to form the input to the next recurrent layer as illustrated in Figure 2.

$$\mathbf{y}_t = [\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t] \quad (6)$$

The final model is then composed of multiple stacked bidirectional LSTM layers, resulting in a deep bidirectional LSTM neural network followed by a softmax layer with $K + 1$ outputs nodes, where K is the count of output labels with an additional node to represent a *blank* label ϕ; the blank label allows

Table 1: Baseline results using Librispeech 100hr (`train-clean-100`) corpus as training data and pruned 3-gram ARPA LM (`tgmed`). For comparison, the last two rows show performance when Librispeech 360hr (`train-clean-360`) and Librispeech 500hr (`train-clean-500`) training sets are incrementally added.

System	%WER (*dev-clean*)
Grapheme	13.01
Phoneme	11.81
Reduced Phoneme	11.15
Kaldi (p-norm DNN, LDA+MLLT+SAT, 100hrs training)†	7.91
Kaldi (p-norm DNN, LDA+MLLT+SAT, 460hrs training)†	7.16
Kaldi (p-norm DNN, LDA+MLLT+SAT, 960hrs training)†	6.57

† `https://github.com/kaldi-asr/kaldi/blob/master/egs/librispeech/s5/RESULTS` (retrieved 5/30/2017).

the network to output a "none" label when evidence is insufficient to output a specific label. The bidirectional LSTM network is trained with the CTC loss function, which minimizes the negative log summed probability of the correct label sequence given the input, via a forward-backward algorithm that sums across all possible alignments. Details can be found in [6]. Unless otherwise indicated, the LSTM model in our experiments consists of 4 bidirectional stacked layers of 640 LSTM cells (320 in each direction).

Our system training procedure mirrors the EESEN WSJ recipe; inputs are 40 dimensional mel warped filterbank features with Δ and $\Delta\Delta$ features, normalized with per speaker means subtraction and variance normalization, and outputs are the grapheme or phoneme sequence. EESEN uses a weighted finite state grammar (WFST) to incorporate the language model and generate the final word sequence from the network outputs. Details can be found in [8].

During training, we use the "newbob" learning rate (LR) schedule, where the LR is first set to a fixed value (0.00004) and halved every epoch once the improvement in token accuracy on a withheld cross-validation (CV) set falls below 0.5%. Training is stopped once the improvement in CV token accuracy falls below 0.1%. In later experiments, discussed in this paper, we modified this approach to allow training to continue until the token accuracy had either plateaued over several epochs or was clearly not going to exceed that of other methods, before triggering LR halving.

We choose the Librispeech 100hr (`train-clean-100`) corpus as our training set. This corpus is publicly available and allows others to easily reproduce our results using our modified EESEN code available at `https://github.com/jb1999/eesen`. Table 1 presents the results of this system on the Librispeech *dev-clean* test set when trained on grapheme and phoneme output units. Two variants of the phoneme system were trained, one that used the phoneme set provided with the Librispeech corpus that includes phoneme stress markers, and another where the stress markers were removed creating a reduced phoneme set. For the phoneme based systems, we use the Librispeech provided dictionary, but remove any alternate pronunciations so that each word has only one pronunciation.

In all experiments, the Librispeech supplied 3-gram ARPA LM, pruned with threshold 1e-7, `tgmed` in the recipe, is used as the language model during decoding. Given that the reduced phoneme model showed the best performance across the trained EESEN systems, we adopted this variant as our canonical model, all subsequent experiments in this paper incrementally add capabilities to this system.

Also shown in Table 1 is the Kaldi baseline performance following the s5 recipe for Librispeech 100hrs with the same language model. This Kaldi model uses composite mel frequency cepstral coefficient (MFCC) features over which Linear Discriminant Analysis (LDA), Maximum Likelihood Linear Transform (MLLT) and Speaker Adaption Transform (SAT) transformations are applied to generate 40 dimensional features used during training; the DNN architecture itself consists of 4 p-norm layers with 3486 outputs corresponding to the context dependent clusters. We should note here that the Kaldi baseline system in Table 1 has about half the trainable parameters, 4.5M parameters, as opposed to the corresponding EESEN systems with 9.1M parameters. We did increase the size of the Kaldi baseline model by proportionally adjusting the p-norm layer input and output dimensions to approximately match the EESEN system trainable parameters. However, the larger Kaldi system's

Table 2: Forget gate bias initialization.

System	%WER (*dev-clean*)
Random initialization	11.15
$\mathbf{b}_f = \mathbf{1}$	12.23
$\mathbf{b}_f = \mathbf{1}$ + minimum 6 epochs before LR halving	10.76

performance was worse, with a 9.02% WER, albeit with no specific tuning/optimization on our part. Note also that the Kaldi baseline incorporates context dependency modeling whereas the EESEN models output context independent phonemes/graphemes.

3 LSTM Initialization

Typically, LSTM parameters are initialized to small random values in an interval, e.g., in our experiments we initialize weights and biases to random values in the interval $[-0.1, 0.1]$. However, for the forget gate, this is a suboptimal choice, where small weights effectively close the gate, preventing cell memory and its gradients from flowing in time. One approach to address this issue is to initialize the forget gate bias, $\mathbf{b}_f$, to a large value, say 1, that will force the gate to be initialized in an open position and allow memory cell gradients in time to flow more readily. This practical detail has been noted in earlier work [12–14]. In our work, however, we found that simply increasing the forget gate bias did not provide the anticipated gains, and during training, accuracy in early epochs plateaued very quickly and triggered our stopping criterion, with worse results than if we had left the forget gate bias set to a random small initial value. However, if we disable the stopping criterion and force a minimum number of epochs, in this case 6 epochs, we did observe consistent gains with $\mathbf{b}_f = \mathbf{1}$. The results from our experiments with forget gate bias initialization are summarized in Table 2.

As a practical matter, there is no reliable way to predetermine the minimum epochs for any particular system. In later experiments, we opted to simply disable the stopping criterion, monitor accuracy on our withheld cross validation set, and manually trigger "newbob" learning rate (LR) schedule, when the token accuracy had plateaued over several epochs.

4 Data Augmentation

Data augmentation is used in state of the art ASR systems to increase the available training data, improve generalization capability, as well as improve system robustness to the deployment environment. Typically, data is augmented by either adding noise from other sources [3, 4] or by explicit transformation of the data either in the time domain as speed perturbation [15] or frequency domain warping using vocal tract length perturbation (VTLP) [16, 17]. In the first case, we need a suitable noise source and method to add the appropriate noise to the training data. For speed perturbation, we need an external process that modifies the data prior to feature extraction. In VTLP, either the utterance is warped with a random factor [16], or the existing warp factor for the speaker is jittered [17]. Another variation, Stochastic Feature Mapping [17], seeks to augment data by converting one speaker's speech to another speaker.

An alternate perspective is to view data augmentation as an exploration method to push the model to explore and find a more beneficial space on the manifold, which in turn, provides a path to a deeper minimum. This is conceptually similar to simulated annealing [18], in that we consider data augmentation as analogous to changing temperatures to allow for more robust exploration for a global minimum. Since we are looking to encourage model exploration vs generalization, our implementation of data augmentation diverges from a more typical implementation in two ways:

- Data augmentation is applied to introduce data variety without attempting to faithfully retain audio characteristics. In our case, we choose to uniformly warp all the data in the frequency domain using VTLN warp factors 0.8 and 1.2 across all the training data, in addition to retaining the original unwarped features. This clearly is wrong in many cases, where these warp factors result in either excessive compression or expansion of the frequency spectrum. To provide an additional level of variance, we also modified the frame rate to speed up or

Table 3: Improvements from data augmentation.

System	%WER (*dev-clean*)
prior result with $\mathbf{b}_f = \mathbf{1}$	10.76
9-fold max perturbation	9.78
20-fold max perturbation	9.95
3-fold speed perturbation	10.22

slow down the features. We use frame rates of 8ms and 11ms in addition to the original 10ms frame rate.

- We train the model with different set of augmented data every epoch and cycle through the different augmented data sets, as opposed to selecting from a different augmented data set for every mini-batch or utterance.

We refer to this collective approach to data augmentation as *max* perturbation to distinguish from approaches that attempt to more faithfully retain the audio characteristics and seek to introduce small perturbations during training. Further, this approach has the advantage of leveraging existing feature extraction tools and training process, albeit with different parameter settings, without requiring additional tools or processes.

Table 3 summarizes the gains from using max perturbation in comparison with the speed perturbation approach, which in earlier work has been shown to provide better performance than frequency based transformation [15]. With 9-fold max perturbation, we observed a 9.1% relative reduction in WER, whereas 3-fold speed perturbation as described in [15] but with per epoch augmented training described above, resulted in a 5.0% relative reduction in WER. For comparison, we note that Ko et. al. in [15] show speech perturbation results across a variety of data sets with relative reduction in WER ranging from 6.7% to 0.32% . Our speed perturbation result of 5.0% relative improvement is consistent with, and in the high end of, these prior results. In other experiments (not shown here), we observed that 75% of the max perturbation gain can be attributed to VTLN warping data augmentation and 25% of the gain to frame rate data augmentation, and that both these gains were orthogonal and largely additive.

In order to explore how max perturbation behaves with more data variants, we also experimented with 20-fold max perturbation with frame rates 8, 10,11 and 12ms, and VTLN warp factors 0.7, 0.8, 1.0, 1.2 and 1.3. We found that 20-fold max perturbation showed an improvement over no data augmentation and 3-fold speed perturbation, but was slightly worse than 9-fold max perturbation with a 7.5% relative reduction in WER over no data augmentation. This result is quite interesting considering that we are well outside of traditional norms in terms of how much we can distort speech data and still retain modeling value.

Given the relative outperformance of max perturbation data augmentation vis-a-vis speed perturbation we used 9-fold max perturbation in all of our subsequent experiments.

5 Stacking and striding frames

Frame stacking is where consecutive feature frames are concatenated to create a composite feature – this increases the size of the input feature. Frame striding is where feature frames at particular times are skipped and not presented to the network – this is equivalent to downsampling the feature frame sequence. Both approaches have been used independently and concurrently in prior DNN and LSTM-CTC systems [2, 4, 9] to better capture discriminative features as well as to reduce computation and system training time.

We conducted several experiments with stacking and striding, and observed a consistent drop in performance. Illustrative results are shown in Table 4, with ± 1 feature frames, for a composite stacked feature of 3 frames, and a stride of 3, for an effective 30ms frame rate vs. the original 10ms frame rate. We find that 9-fold max perturbation data augmentation provides a slight, but relatively stronger, performance improvement (10.1% vs. 9.1% relative) when compared to the model based on the base features. While these results are not compelling by themselves, as we show in the next

Table 4: Experiments with frame stacking and striding.

System	%WER (*dev-clean*)
Baseline	10.76
Baseline + 9-fold max perturbation	9.78
3 frame stack, stride = 3	11.92
3 frame stack, stride = 3 + 9-fold max perturbation	10.72

section, stacking and striding is a key element in driving performance improvements with dropout in LSTM-CTC systems.

6 Dropout

In many of our initial experiments, we observed consistent overfitting, as also noted by others [19]. In order to minimize overfitting we explored various approaches to dropout on feedforward and recurrent connections. While the general approach to apply dropout is well established for feedforward networks [20], application to recurrent neural networks, however, has seen a number of variants [21–24].

Initial implementations of dropout for recurrent neural networks focused on application of dropout on the feedforward connections only [21, 25]. In subsequent work, Moon et. al. [22] introduced RNNDrop, in which dropout is applied to LSTM cell memory at a sequence level, i.e., the dropout mask is kept fixed across each training sequence, or in the case of speech, each utterance. Gal et. al. [24] Variational RNN approach is similar, but dropout is applied to the LSTM cell output recurrent connection as well as the forward non-recurrent connections, also at a sequence level. An interesting variant is recurrent dropout without memory loss [23], where dropout is applied to the LSTM cell memory update, which prevents the LSTM cell memory from being reset as is the case with RNNDrop. More recently, Cheng et. al. [26] explore dropout in projected LSTM based ASR systems. Given the slightly different architecture of the projected LSTM cell, they investigate different dropout location approaches and further incorporate a schedule driven dropout rate, where the dropout factor changes over time and show relative WER reductions on the order of 3% to 7% on a variety of data sets.

In our experiments, detailed below, we systematically explore the application of dropout to the feedforward and recurrent connections, separately, as well as together. We show that careful application of dropout, when coupled with max perturbation and stacked and strided features, is a significant driver of system performance.

We should note that in our implementation of dropout, we scale the mask during training with no change to the test paradigm. To maintain a common operating point for our experiments, we fix the dropout rate to 0.2 for forward and recurrent connections as applicable. It is likely that a more comprehensive exploration of dropout rates will provide additional performance improvements.

6.1 Dropout on feedforward connections

Dropout on feedforward connections in LSTM networks is closely aligned with the original formulation of dropout where the composite LSTM cell is the unit to be dropped. In earlier work, Pham et. al. [25] and Zaremba et. al. [21], make the argument for applying dropout only on the feedforward connections and not the recurrent connections so as to minimize impact on the sequence modeling capability of a recurrent network. In the implementation described in [21], dropout is applied every time step to the feedforward connections. However, given the within layer recurrence, application of dropout at every time step would be a noisy implementation of the classical dropout approach, i.e., each sampling of the networks would receive input from another sampling of the network via the recurrent connections. We argue that a more faithful implementation for recurrent networks would be to retain the dropout mask across a complete sequence/utterance for the forward non-recurrent connections to eliminate this cross-sampling noise. In this instance, each sequence is trained on a particular sampling of the network, rather than a multitude of sampled networks driving a noisy

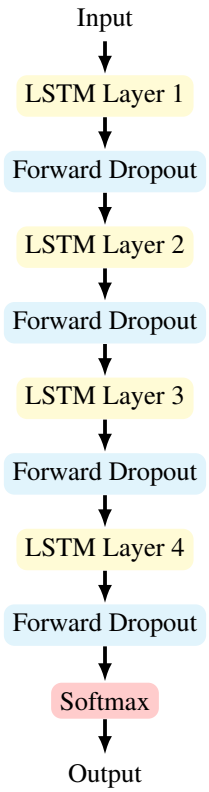

Figure 3: Forward Dropout as implemented in our experiments. The dropout mask can be sampled either every time step or every sequence. In the latter, the dropout mask is fixed for all time steps in any specific sequence.

Table 5: Experiments with dropout on feedforward connections. Forward-step indicates feedforward connection dropout with dropout mask sampled every time step. Forward-sequence indicates feedforward connection dropout with dropout mask sampled every sequence.

System	%WER (*dev-clean*)
Baseline (9-fold max perturbation)	9.78
Forward-step	9.51
Forward-sequence	10.03

recurrence. We note that this is identical to the Gal et. al. [24] dropout implementation for forward connections. Figure 3 illustrates the forward dropout implementation in our experiments.

In order to validate our thinking, we experimented with both time step and sequence forward dropout variants. Table 5 details our results. We expected a performance improvement with dropout mask sampled every time step and a larger improvement when the dropout mask is sampled every sequence. However, while we see an improvement with dropout mask sampling every time step, when the dropout mask is sampled every sequence, performance is worse than without dropout. Inspection of the training logs showed that this model started to overfit early, driving the lower performance.

Given the slightly better relative WER improvement with 9-fold max perturbation using stacked and strided features, we proceeded to investigate if dropout would also show a similar magnified effect when coupled with stacked and strided features. We applied the same forward dropout variants as in Table 5 with stacked and strided features; Table 6 details these results. Once again, we see a larger improvement when using stacked and strided features; models trained with stacked and strided features are much more responsive to dropout, showing a significant improvement in performance over models trained with the base features. Per time step dropout mask sampling with 9-max

Table 6: Experiments with dropout on feedforward connections with stacked/strided features.

System	%WER (*dev-clean*)
Baseline (9-fold max perturbation + stacked/strided features)	10.72
Forward-step	9.17
Forward-sequence	8.63

perturbation yields a 14.5% relative WER improvement over the corresponding baseline, and per sequence dropout mask sampling yields 19.5% relative WER improvement. To be conservative, compared to the base feature system with 9-fold max perturbation with 9.78% WER, we still see a relative WER improvement of 6.2% and 11.8% respectively for the time step and sequence variants.

During training we found that networks trained on stacked and strided features were able to train for many more epochs without overfitting when coupled with dropout, resulting in performance that is significantly better than using the base features with/without dropout or the stacked and strided features without dropout. Given the strong outperformance of stacked and strided features with dropout we shifted to using stacked and strided features for all of our subsequent dropout experiments.

6.2 Dropout on Recurrent connections

In our exploration of dropout for recurrent connections, we looked at two versions of recurrent connection dropout, RNNDrop [22] and recurrent dropout without memory loss [23]. As described earlier, RNNDrop applies dropout to the memory cell content, in particular, Equation 3 which describes the memory cell, changes as below

$$\mathbf{c}_t = \mathbb{m}_t \odot \big(\mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \phi(\mathbf{W}_c\mathbf{x}_t + \mathbf{R}_c\mathbf{h}_{t-1} + \mathbf{b}_c)\big) \tag{7}$$

where $\mathbb{m}_t$ represents the dropout mask at time t.

In the case of recurrent dropout without memory loss, dropout is only applied to the incremental memory cell update, Equation 3 changes as below

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbb{m}_t \odot \mathbf{i}_t \odot \phi(\mathbf{W}_c\mathbf{x}_t + \mathbf{R}_c\mathbf{h}_{t-1} + \mathbf{b}_c) \tag{8}$$

where again $\mathbb{m}_t$ represents the dropout mask at time t.

We expect recurrent dropout without memory loss to show better performance vis-à-vis RNNDrop since the cell memory is not being reset continually as is the case with RNNDrop.

Figure 4 illustrates the location of these two recurrent dropout approaches. Location 1 corresponds the application of RNNDrop, Location 2 corresponds to the application of recurrent dropout without memory loss.

As we did with forward dropout, we decided to explore both per time step and per sequence implementation variants on both RNNDrop and recurrent dropout without memory loss. However, during our experimentation, we discovered that the direct implementation of RNNDrop as described in [22] with per sequence dropout mask did not train and suffered from an exploding memory cell value problem, an issue predicted in [23], and was subsequently excluded from our experiments.

Table 7 details our results, where we have abbreviated recurrent dropout without memory loss as no memory loss (NML) dropout. From the table it is clear that application of dropout for recurrent connections is indeed beneficial. In line with our expectations, NML dropout worked better than RNNDrop with the per sequence dropout mask variant slightly edging out the per time step dropout mask variant, inline with trends reported in [23] albeit on different tasks. For the NML dropout variants we see relative WER reductions over 20% compared to the corresponding baseline, or $\sim$13% relative WER reduction compared to the prior best system without dropout WER of 9.78%.

We note that in the case of RNNDrop implemented with a per time step mask, with a dropout rate of 0.2, we effectively reset $\sim$99%[1] of all LSTM cells within 20 time steps, i.e., each LSTM cell retains at most 600ms of memory with our 30ms frame rate. An interesting corollary of this observation is that we can consider refactoring a bidirectional LSTM with RNNDrop system as an unidirectional LSTM system with suitably delayed output, with similar or near similar performance and lower latency.

[1] $1 - 0.8^{20}$

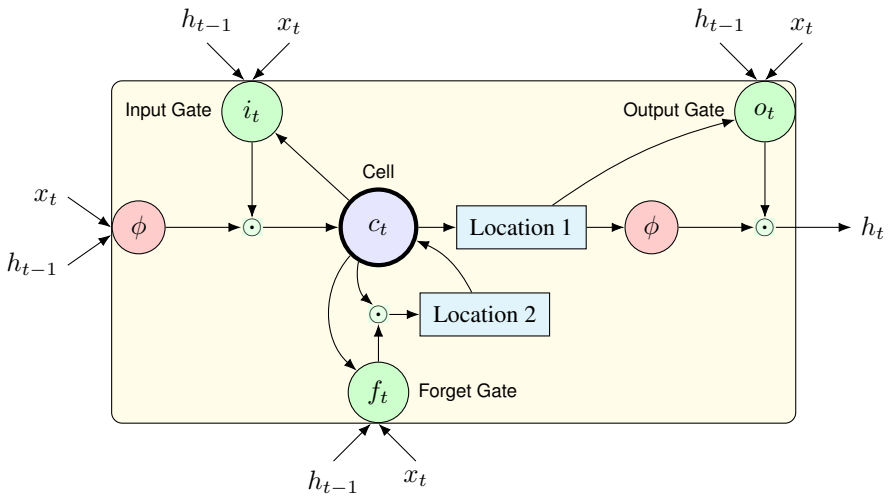

Figure 4: LSTM Memory cell with recurrent dropout locations. Location 1 corresponds to dropout location for RNNDrop, Location 2 corresponds to dropout location for recurrent dropout without memory loss.

Table 7: Experiments with dropout on recurrent connections. Postfix step and sequence indicates whether dropout mask is sampled every time step or every sequence. Recurrent dropout without memory loss is abbreviated as no memory loss (NML) dropout.

System	%WER (*dev-clean*)
Baseline (9-fold max perturbation + stacked/strided features)	10.72
RNNDrop-step	9.07
NML-step	8.55
NML-sequence	8.45

6.3 Combining feedforward and recurrent dropout

In the previous sections we have shown that both forward and recurrent dropout mechanisms improve system performance to a similar degree. The next logical extension is to combine both forward and recurrent dropout in a single model and explore if these performance gains are complementary.

6.3.1 Naïve dropout combination

The simplest approach to combine dropout is to apply both forward and recurrent dropout concurrently during network training; we refer to this combined dropout approach as *naïve* dropout combination. Indeed this is the approach that has been traditionally taken while combining dropout [24, 23].

To be exhaustive we ran experiments with all combinations of the three recurrent dropout variants and the two forward dropout variants, albeit with the same fixed dropout rate of 0.2. Table 8 summarizes these models and their corresponding WERs. The RNNDrop combination variants, while showing better WER performance than the RNNDrop alone system, are worse, or at best similar, in performance to the forward dropout alone models. On the other hand, the NML combination variants all show performance improvements over the NML or forward dropout alone models. We note that with these results, the LSTM-CTC system is at parity with the Kaldi Librispeech 100hr hybrid DNN system in Table 1.

6.3.2 Stochastic and cascade dropout combination

During our experimentation, it was clear that different dropout approaches had very different training profiles in how token accuracy improved on training and cross-validation and the corresponding difference in metrics. One conjecture is that the different dropout approaches explore a different part

Table 8: Experiments with Naïve dropout combination.

System	%WER (*dev-clean*)
RNNDrop-step + Forward-step	8.60
RNNDrop-step + Forward-sequence	8.85
NML-step + Forward-step	8.08
NML-step + Forward-sequence	7.76
NML-sequence + Forward-step	7.72
NML-sequence + Forward-sequence	7.97

Table 9: Experiments with Stochastic dropout combination.

System	%WER (Stochastic)	%WER (Naïve)
NML-step + Forward-step	8.76	8.08
NML-step + Forward-sequence	8.02	7.76
NML-sequence + Forward-step	7.86	7.72
NML-sequence + Forward-sequence	7.44	7.97

of the parameter space. If this is the case, we can direct this exploration more intelligently than a naïve dropout combination.

One combination approach is to apply forward or recurrent dropout singly rather than concurrently. The exact approach we implemented is that for each minibatch, we pick from an equiprobable Bernoulli distribution to decide between forward or recurrent dropout, and then apply the appropriate dropout for that minibatch. Note that there is nothing special in our choice of distribution or decision choice, an equally valid implementation could bias towards a particular dropout or introduce an additional decision choice to pick between forward and/or recurrent dropout types. We refer to this general approach of distribution based choice to determine dropout combination as *stochastic* dropout combination.

Another combination approach is to train the model with one type, combination or parameterization of dropout, and then switch to a different type of dropout, combination or parameterization, at an opportune time. Given that we cascade different dropout combinations during training we refer to this general approach as *cascade* dropout combination. The dropout schedule approach described in Cheng et. al. [26] is a specific example of cascade dropout combination. We should note that cascade and stochastic dropout combination are orthogonal approaches and can be applied at the same time.

Given our limited compute resources, we were unable to systematically explore the many permutations of dropout combination in a reasonable time frame. Table 9 summarizes our experiments with stochastic dropout combination and compares the results with naïve dropout combination. We find that while not all stochastic dropout combination results show better performance over naïve dropout combination, in the case of sequence based NML/Forward dropout combination we see an additional 6.6% relative reduction in WER over naïve dropout combination.

Table 10 illustrates one example of cascade dropout combination where we see a small improvement in WER over the individual systems alone.

7 Conclusions and future directions

In this paper, we have described our efforts to eliminate the observed performance gap between hybrid DNN CE and LSTM-CTC ASR systems on smaller domains. We show that by combining max perturbation and stacked/strided features with feedforward and dropout we can build an LSTM-CTC system that exceeds the performance of strong hybrid DNN CE system on a domain with 100 hours of training data. In this process, at least for the Librispeech 100hr corpus, we have

- introduced max perturbation, a modified data augmentation paradigm which outperforms prior approaches.

Table 10: Experiments with Cascade dropout combination.

System	%WER (*dev-clean*)
NML-sequence + Forward-step (1)	7.72
NML-sequence + Forward-sequence (2)	7.97
Cascade $(1) \rightsquigarrow (2)$	7.51

- shown that the combination of max perturbation, stacked/strided features, forward and recurrent dropout significantly improve LSTM-CTC ASR system performance.

Furthermore, none of the approaches in this paper explicitly make assumptions on or accommodations for corpus size, and should work in a similar manner when trained with larger datasets.

One criticism of this work is that both data augmentation and dropout would also improve the corresponding Kaldi baseline; this is most certainly the case, but even if we take the best reported results with data augmentation [15] and dropout [26] and assume that they are additive, the LSTM-CTC system would be within 10% relative WER of this Kaldi baseline[2], a significant leap over our initial starting point where the LSTM-CTC system was around 30-35% worse in WER.

Another criticism of this work is that we have omitted well known approaches to improve system performance such as discriminative training (e.g. [27]) and recent advances in combining different types of layers such as time delay neural network (TDNN) and convolutional layers (e.g. [28]) from our Kaldi baseline and thus comparing to an inferior baseline. However, evidence indicates that these approaches benefit LSTM-CTC based systems as well (e.g., [2, 4, 29]); in [27] sequence discriminative training on a hybrid DNN ASR system resulted in an 8% relative reduction in WER whereas in [4] sequence discriminative training on a LSTM-CTC ASR system resulted in an 10% relative reduction in WER albeit on a different task; in [28], TDNN layers are shown to result in 6.9% relative WER reduction on the Librispeech corpus, whereas in [29] a CLDNN (for convolution + LSTM + DNN) CE ASR system showed between 4-6% relative WER reduction, though in later work [30], when trained with CTC, performance was slightly worse. On the other hand, Amodei et. al. [2] use convolutional layers coupled with gated recurrent unit (GRU) layers but do not indicate the relative WER change from using only recurrent layers to the combination of convolutional and recurrent layers.

Another issue is that we have not validated these approaches and results on other datasets to confirm that they are broadly applicable principles rather than a Librispeech corpus specific idiosyncrasy. Unfortunately, given our limited resources, we were restricted to publicly available corpora.

All said, considering the EESEN system trained on the original phoneme set as the starting point, with a 11.81% WER, we report improvements that reduce error rates to 7.44% WER, eliminating and improving on the performance gap with our target strong Kaldi based Hybrid DNN baseline with 7.91% WER. While we have introduced several new approaches to improve LSTM-CTC system performance, given our limited compute pool, we have been unable to fully explore the parameter space of these improvements, i.e., we have not swept the parameters for optimal dropout rate, model size, learning rate or explored the many permutations of dropout combination. It is likely that a more comprehensive exploration will uncover additional performance improvements.

References

[1] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1764–1772, 2014.

[2] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse Engel, Linxi Fan, Christopher Fougner, Awni Y. Hannun, Billy Jun, Tony Han, Patrick LeGresley, Xiangang Li, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Sheng Qian, Jonathan Raiman, Sanjeev Satheesh,

[2] Ko et. al. [15] show between 6.7% to 0.32% relative WER reduction using speed perturbation approach to data augmentation, Cheng et. al. [26] show 3-7% relative WER reduction using dropout; our best system with 7.44% WER is 5.9% better than the Kaldi system.

David Seetapun, Shubho Sengupta, Chong Wang, Yi Wang, Zhiqian Wang, Bo Xiao, Yan Xie, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2 : End-to-end speech recognition in english and mandarin. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 173–182, 2016.

[3] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[4] Hasim Sak, Andrew W. Senior, Kanishka Rao, and Françoise Beaufays. Fast and accurate recurrent neural network acoustic models for speech recognition. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 1468–1472, 2015.

[5] Hagen Soltau, Hank Liao, and Hasim Sak. Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition. *CoRR*, abs/1610.09975, 2016.

[6] Alex Graves, Santiago Fernández, Faustino J. Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, pages 369–376, 2006.

[7] Daniel Povey, Vijayaditya Peddinti, Daniel Galvez, Pegah Ghahremani, Vimal Manohar, Xingyu Na, Yiming Wang, and Sanjeev Khudanpur. Purely sequence-trained neural networks for ASR based on lattice-free MMI. In *Interspeech 2016, 17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, pages 2751–2755, 2016.

[8] Yajie Miao, Mohammad Gowayyed, and Florian Metze. EESEN: end-to-end speech recognition using deep RNN models and wfst-based decoding. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015, Scottsdale, AZ, USA, December 13-17, 2015*, pages 167–174, 2015.

[9] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[10] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An ASR corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*, pages 5206–5210, 2015.

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

[12] Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

[13] Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2342–2350, 2015.

[14] Yajie Miao, Mohammad Gowayyed, Xingyu Na, Tom Ko, Florian Metze, and Alexander H. Waibel. An empirical exploration of CTC acoustic models. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 2623–2627, 2016.

[15] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 3586–3589, 2015.

[16] Navdeep Jaitly and G. E. Hinton. Vocal tract length perturbation (vtlp) improves speech recognition. In *Proceedings of the International Conference on Machine Learning (ICML) 2013 Workshop on Deep Learning for Audio, Speech and Language Processing, Atlanta, Georgia, USA, June 16-21, 2013*, 2013.

[17] Xiaodong Cui, Vaibhava Goel, and Brian Kingsbury. Data augmentation for deep convolutional neural network acoustic modeling. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*, pages 4545–4549, 2015.

[18] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[19] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pages 338–342, 2014.

[20] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[21] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.

[22] Taesup Moon, Heeyoul Choi, Hoshik Lee, and Inchul Song. RNNDROP: A novel dropout for RNNS in ASR. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015, Scottsdale, AZ, USA, December 13-17, 2015*, pages 65–70, 2015.

[23] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. In *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, pages 1757–1766, 2016.

[24] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1019–1027, 2016.

[25] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *14th International Conference on Frontiers in Handwriting Recognition, ICFHR 2014, Crete, Greece, September 1-4, 2014*, pages 285–290, 2014.

[26] Gaofeng Cheng, Vijayaditya Peddinti, Daniel Povey, Vimal Manohar, Sanjeev Khudanpur, and Yonghong Yan. An exploration of dropout with lstms. In *INTERSPEECH 2017, Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, pages xxxx–xxxx, 2017.

[27] Karel Veselý, Arnab Ghoshal, Lukás Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In Frédéric Bimbot, Christophe Cerisara, Cécile Fougeron, Guillaume Gravier, Lori Lamel, François Pellegrino, and Pascal Perrier, editors, *INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association, Lyon, France, August 25-29, 2013*, pages 2345–2349. ISCA, 2013.

[28] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*, pages 3214–3218. ISCA, 2015.

[29] Tara N. Sainath, Oriol Vinyals, Andrew W. Senior, and Hasim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2015, South Brisbane, Queensland, Australia, April 19-24, 2015*, pages 4580–4584. IEEE, 2015.

[30] Andrew W. Senior, Hasim Sak, Felix de Chaumont Quitry, Tara N. Sainath, and Kanishka Rao. Acoustic modelling with CD-CTC-SMBR LSTM RNNS. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015, Scottsdale, AZ, USA, December 13-17, 2015*, pages 604–609, 2015.