Bidirectional Truncated Recurrent Neural Networks for Efficient Speech Denoising

Philémon Brakel, Dirk Stroobandt, Benjamin Schrauwen

Department of Electronics and Information Systems, Ghent University, Ghent, Belgium
philemon.brakel@ugent.be, dirk.stroobandt@ugent.be, benjamin.schrauwen@ugent.be

Abstract

We propose a bidirectional truncated recurrent neural network architecture for speech denoising. Recent work showed that deep recurrent neural networks perform well at speech denoising tasks and outperform feed forward architectures [1]. However, recurrent neural networks are difficult to train and their simulation does not allow for much parallelization. Given the increasing availability of parallel computing architectures like GPUs this is disadvantageous. The architecture we propose aims to retain the positive properties of recurrent neural networks and deep learning while remaining highly parallelizable. Unlike a standard recurrent neural network, it processes information from both past and future time steps. We evaluate two variants of this architecture on the Aurora2 task for robust ASR where they show promising results. The models outperform the ETSI2 advanced front end and the SPLICE algorithm under matching noise conditions.

Index Terms: recurrent neural networks, deep learning, robust ASR

1. Introduction

One of the key features of a robust system for automatic speech recognition (ASR) is the ability to handle background noise. Methods for improving noise robustness either try to improve the quality of the features that are presented to the recognition system [2], or to improve the robustness of the recognition system itself [3]. In this paper we focus on the first of these two approaches.

To enhance the quality of features that have been derived from a speech signal, one can either use expert knowledge about the characteristics of human speech or resort to data-driven systems. As more data is becoming available, the latter approach seems to become increasingly viable.

A successful data driven approach to speech denoising is the Stereo-based Piecewise Linear Compensation for Environments (SPLICE) method [4], which models the joint distribution between clean and noisy utterances. In this approach, the clean signal is modelled as a piecewise linear function of the noisy signal. This function is constructed from models that have been separately trained on different types and levels of noise.

Neural networks are trainable non-linear functions that can learn from data to perform a mapping from some input pattern to a desired output pattern. What makes neural networks interesting is that they can be applied to large datasets and that they can process information relatively fast after they have been trained. This makes them good candidates for various speech processing tasks. When both noisy and clean versions of utterances are available, neural networks can be trained to perform speech denoising directly [6].

Recently, there has been a revival of interest in neural networks for speech processing due to the success of the so-called deep learning approach [7] on a variety of machine learning tasks. Most importantly, deep architectures have been shown to perform very well as acoustic models for automatic speech recognition [8]. The idea behind deep learning is to use neural networks with multiple layers of hidden units that learn increasingly complex representations of the input patterns.

A neural network architecture that is particularly interesting for speech processing is the Recurrent Neural Network (RNN). RNNs are able to process information over time by updating a set of state variables that are a function of both their previous state and the current input pattern. One of the first applications of RNNs in speech recognition was phoneme prediction [9]. Recently, it was shown that a combination of deep learning with RNNs [1] can outperform well-known noise reduction methods like the SPLICE algorithm [4] and the ETSI2 advanced front end [2]. It was also shown that RNNs can improve robust speech recognition in a Tandem setup [10].

As datasets are becoming increasingly large and parallel processing units like GPUs are getting more commonly available, it becomes increasingly beneficial for computational methods to perform computations in parallel. Unfortunately, RNNs are quite slow and the computations involved cannot be parallelized. Moreover, they are difficult to train because of the so called *vanishing gradients* problem [11]. The vanishing gradients problem occurs because the backpropagation of gradients through time includes a large number of multiplications which have a tendency to either drive the gradients towards zero so learning occurs very slowly, or to make them very large so learning can become unstable.

In this paper we propose two variants of a recurrent neural network architecture that inherits some of the properties of RNNs but allows for more parallelization. The models can also use information from future speech frames and seem to be easier to optimize because the number or required backpropagation iterations is smaller. The way the models process information is similar to the type of information processing in deep neural networks. We demonstrate their performance on the Aurora2 robust ASR task when they are trained with an advanced second order optimization method to remove noise from MFCC speech features. Under matched noise conditions, our models outperform the SPLICE method and the ETSI2 advanced front end.

2. Bidirectional truncated Recurrent Neural Networks

Let $\mathbf{V}$ be a noisy input sequence of column vectors $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N\}$, where N is the length of the sequence, and let $\mathbf{Y}$ be the corresponding clean sequence we want to predict. The goal is to learn a function $\mathbf{V} \mapsto \hat{\mathbf{Y}}$ with a minimal difference between the predicted sequence $\hat{\mathbf{Y}}$ and the true clean sequence $\mathbf{Y}$ as measured in the squared error given by

$$\|\hat{\mathbf{Y}} - \mathbf{Y}\|^2, \tag{1}$$

where $\|\cdot\|$ is the Frobenius norm.

The architecture we propose processes information in two directions through time; unlike a standard recurrent neural network, it also uses information about future frames. We call the architecture 'truncated', because the number of iterations that is carried out to process information is much smaller than the length of the input sequence.

Like a multilayer perceptron, the bidirectional truncated recurrent neural network computes a sequence of hidden unit activations $\mathbf{H}$ and maps these activations to a prediction sequence $\hat{\mathbf{Y}}$. The activation of a hidden unit is computed as a weighted sum of the activations of the units it is connected to, followed by a non-linear transformation (in this case the hyperbolic tangent function).

The temporal component is introduced in the computation of these hidden unit activations. For a predetermined number of iterations K, the hidden units are updated with the information they receive from both their neighbours and the input pattern $\mathbf{V}$. At the start of an iteration, the odd hidden state vectors $\{\mathbf{h}_m | m \in 2\mathbb{N}+1\}$ receive a weighted sum of the activations of their neighbours with an even index and the current input frame. Subsequently, the even units are updated based on the activations of the odd ones. Algorithm 1 shows this procedure in detail. The trainable parameters of the model are the input connection weights $\mathbf{W}_{\text{in}}$, the recurrent weights $\mathbf{W}_{\text{rec}}$, the output weights $\mathbf{W}_{\text{out}}$ and the bias parameters for the recurrent and output layers $\mathbf{b}_{\text{rec}}$ and $\mathbf{b}_{\text{out}}$.

The two inner for-loops in Algorithm 1 can both be replaced by two matrix multiplications. On parallel computing architectures, this modification can lead to significant speedups. This kind of parallelization is not possible for standard recurrent architectures where $N-1$ matrix-vector products are required. The architecture of this first variant of the model is shown in Fig. 1a. From now on we will refer to this model as the bidirectional truncated recurrent neural network (BTRNN).

Algorithm 1 *Predict clean sequence $\hat{\mathbf{Y}}$ given noisy sequence* $\mathbf{V}$

```
Initialize $\mathbf{H} \leftarrow \mathbf{0}$
Define $\mathbf{h}_0 = \mathbf{h}_{N+1} = \mathbf{0}$
$\mathbf{A} \leftarrow \mathbf{W}_{\text{in}}\mathbf{V} + \mathbf{b}_{\text{rec}}\mathbf{1}^\mathsf{T}$
for $k = 1$ to $K$ do
    for $j = 1$ to $N$ step 2 do
        $\mathbf{h}_j \leftarrow \tanh(\mathbf{W}_{\text{rec}}\mathbf{h}_{j-1} + \mathbf{W}_{\text{rec}}^\mathsf{T}\mathbf{h}_{j+1} + \mathbf{a}_j)$
    end for
    for $j = 2$ to $N$ step 2 do
        $\mathbf{h}_j \leftarrow \tanh(\mathbf{W}_{\text{rec}}\mathbf{h}_{j-1} + \mathbf{W}_{\text{rec}}^\mathsf{T}\mathbf{h}_{j+1} + \mathbf{a}_j)$
    end for
end for
$\hat{\mathbf{Y}} \leftarrow \mathbf{W}_{\text{out}}\mathbf{H} + \mathbf{b}_{\text{out}}\mathbf{1}^\mathsf{T}$
```

Updating the odd and even units separately was motivated by the mean-field algorithm for Markov Random Fields [12]. If the hidden units would be stochastic variables, this updating scheme would be equivalent to a mean-field approximation of their posterior distribution given the data.

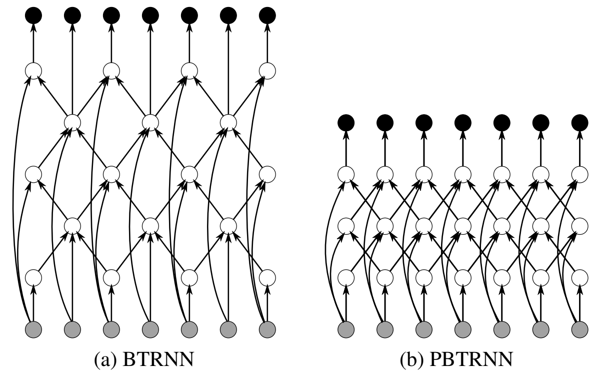

Figure 1: *The two variants of the bidirectional truncated recurrent neural network for two iterations of hidden unit updates. Each circle represents a layer of neural network units. Grey shading indicates that a unit is part of the input pattern. Black shading signifies output units. Time goes from the left to the right so each input unit represents an MFCC vector at a different time step.*

A nice property of this equivalency is that the updates are guaranteed to converge to a fixed point that is a local optimum of an optimization problem that aims to approximate the distribution over the hidden units given the data. To see if this alternating updating is actually necessary, we also look at a simpler version of the model in which all hidden units are updated in parallel. Updating the hidden units in parallel is computationally more efficient but theoretically this could lead to oscillations. However, since the number of iterations we intend to use is quite small, this might not be a problem. The architecture of this second variant of the model is shown in Fig. 1b. We will refer to this model as the parallel bidirectional truncated recurrent neural network (PBTRNN).

Note that the way our models process information from both previous and future frames is not the same as was done in previous work where the term bidirectional RNN was used [13, 14]. In these approaches, two different hidden states captured past and future information separately while our models directly integrate future and past information information into a single state representation.

The number of previous and future frames that are taken into account is determined by the number of iterations K. The prediction for a frame receives information about $2K+1$ frames from the input sequence. The number of iterations to use presents a trade-off between context size and computational efficiency.

It is also clear that as the models run for a greater number of iterations, the number of times that information flows through a non-linear activation unit increases. At every iteration, the information can be said to pass through an additional layer of a deep neural network with tied weights and a sparse connection structure. This is similar to the way convolutional neural networks process information.

3. Experiments

To evaluate our models for speech denoising, we used the Aurora2 dataset [15]. This is a well-known benchmark for robust ASR. The dataset is a collection of connected digits from the TIDigits corpus. The train set contains $8,440$ clean sentences of which duplicates exist that have been corrupted by four noise types at seven different levels of signal to noise ratio (clean, 20dB, 15dB, 10dB, 5dB, 0dB, -5dB). The test set consists of $56,056$ corrupted and clean sentences and is divided in three subsets. Test set A contains the same four noise types as the train data. Test sets B and C contain different types or noise and can be used to evaluate the performance of a model under mismatched training and testing conditions. All systems were trained to map the noisy train sentences to their clean counterparts.

Since the train data only contains a small number of noise types, we do not expect models that are trained on this data to perform very well on new noise types in general. For this reason we will focus on the results on test set A for which the noise types are identical to those on in the train set.

A recognition system was trained on the clean train data only. Subsequently, the noisy utterances from the test set were processed by the various denoising models we compare and these denoised features were presented to the recognizer during testing.

To generate MFCC features, we used the standard HTK scripts that were supplied with the Aurora2 dataset [15]. The models were trained on 13 dimensional vectors that contained 12 MFCC features and the log energy of the speech signal. First and second order derivatives (delta and delta-delta features) were appended after the denoising process during testing. To keep our results comparable with those in [15] we did not apply cepstral mean normalization to the signals. The recognizer was also trained and evaluated with the standard Aurora2 scripts for HTK. This means that whole-word HMMs with 16 states and for each state a Gaussian mixture with 3 components were used.

3.1. Models

The BTRNN and PBTRNN both had 500 hidden units. We set the number of iterations at which the hidden units were updated to 6. This gave each model an input context size of 13 frames and 263513 trainable parameters.

We compare our models with a deep recurrent denoising auto-encoder (DRDAE), an model which has been shown to work well for this task [1]. This model is a neural network with three layers of hidden units of which the second layer of hidden units has recurrent connections. Like in the paper by Maas et al. [1], we set the number of units in each hidden layer to 500 and presented the network with small time windows of three frames: the current time frame and its two neighbours. The number of trainable parameters for this model was 777513.

To investigate the importance of the recurrent data processing in our networks, we also trained a standard multi-layer perceptron with roughly the same number of trainable parameters. This network had a single layer of 1450 hidden units. To make the comparison as fair as possible, we presented this network with time-windows of 13 frames so that it received the same amount of context information as the truncated recurrent neural networks at each time step. The number of trainable parameters of this model was 265363.

Finally, we compared our models with the features generated by the ETSI2 advanced front end (AFE) and the SPLICE algorithm. [2].

3.2. Training

The neural network models were trained to minimize the mean squared error (MSE) between their predictions and the actual clean utterances in the train set. All utterances were normalized by subtracting the mean and dividing by the square root of the variances of the noisy utterances from the train set.

Table 1: *Mean squared error for the neural network models on the train and validation data. The scores are averaged over all different levels and types of noise.*

Model	Train	Validation
DRDAE	17.30	17.55
MLP	16.35	17.62
BTRNN	15.38	16.39
PBTRNN	15.18	16.89

To train the models we used Hessian-Free optimization [16]. This is a second order optimization method which has been shown to be effective for training deep neural networks and RNNs [17]. Earlier results about Hessian-Free optimization also suggest that it is not that important anymore to use any form of unsupervised pre-training when this optimization method is used [16]. Because there doesn't seem to be a straightforward way to apply pre-training to our models we find this property of the optimization algorithm to be particularly appealing.

The Hessian-Free optimizer is a truncated Newton method and needs the gradient of the error objective with respect to the parameters of a model. It also needs a function that provides it with the product of the Gauss-Newton matrix of the model with an arbitrary vector. If the objective is twice differentiable these gradients and matrix vector products can easily be obtained by means of automated differentiation. We used the Theano toolbox [18] for python for this and to simulate the more parallelizable models with a GPU. We used the standard settings for the Hessian-Free algorithm as described in the original paper by James Martens [16]. In preliminary experiments we found that this algorithm outperformed LBFGS and stochastic gradient descent.

The dampening parameters λ was initialized at 100. The number of sequences to use for computing the matrix vector products was set to 300. All weight matrices where initialized with values sampled from a zero mean Gaussian with variance .01. All bias parameters were initialized at 0.

We reserved 20% of the train data for validation. We applied early stopping and selected the parameters that achieved the lowest MSE scores on the validation data after 100 iterations of training for denoising the MFCCs of the test set. We did not apply any form of weight decay during training.

4. Results

Table 1 displays the MSE scores for the models on both the sentences from the train set and those that we left for validation. The BTRNN and PBTRNN have the lowest validation scores. The train error of the PBTRNN is slightly lower than that of the BTRNN but it performs slightly worse on the validation set. Interestingly, the MLP suffers more from early overfitting than the DRDAE model as its train error is quite a bit lower but its validation error is slightly higher. Table 2 shows the word error rates (WER) scores for test set A. This test set had the same types of noise as the train set. Except for the clean utterances, where the AFE got the best performance, the BTRNN and the PBTRNN outperformed all other methods for all noise conditions. The PBTRNN performed slightly better under moderate noise conditions while the BTRNN a performed a little bit better under heavier noise conditions.

Table 2: *Word error rate scores for the neural models and the ETSI2 advanced front end on test set A. The results are averaged over the four different noise types.*

SNR	MFCC	AFE	DRDAE	MLP	TRNN	PBTRNN
Clean	1.06	**0.77**	1.36	4.14	1.37	1.41
20dB	5.02	1.70	1.85	3.04	1.82	**1.68**
15dB	13.11	3.08	2.93	3.77	2.36	**2.19**
10dB	32.81	6.45	5.14	6.08	**3.91**	4.03
5dB	60.74	14.16	12.49	14.00	**9.87**	10.31
0dB	82.98	35.92	35.45	38.59	**29.98**	31.97
-5dB	91.62	68.70	73.70	72.79	**65.37**	67.42

It is surprising how badly the MLP performed on the clean data. This indicates that either recurrent or deep processing, or a combination of them, is important for good neural denoising models. The performance of the DRDAE was worse than the BTRNN and the PBTRNN but the rate at which its performance decreases as a function of increasing levels of noise seems similar. We suspect that it should be able to achieve similar performance to our models but is more difficult to optimize.

We also averaged across noise conditions while leaving out the clean and -5dB scores as described in the ETSI standard. The BTRNN now achieves a WER of 9.59% and the PBTRNN a WER of 10.03%. These results are better than the WER rates reported for the AFE [2] and the SPLICE algorithm (as reported by Droppo et al. [19]) which are 12.26% and 11.67%, respectively. The DRDAE gives a WER of 11.57%, which is higher than the result reported by Maas et al. [1] of 10.85%, which is still higher than the WER scores of both our models. Finally, the MLP gives an average WER of 13.09%, performing worse than the AFE baseline.

While it was not our intention to construct models for mismatched noise conditions, we also report the averaged results for test set B. As expected and like in earlier work [1], the results of the neural models are a lot worse for the mismatched noise types. The BTRNN and PBTRNN achieved WER scores of 25.50% and 24.66%, respectively. This is worse than the SPLICE algorithm which gives a WER of 12.25% on this test set. The MLP and DRDAE gave WER scores of 19.84% and 18.29%, respectively. All models still improved on the raw MFCC features which lead to a WER of 44.42%. It seems that the models that performed worse on the matching test set suffer less from the mismatching conditions but are still worse than the SPLICE baseline. This indicates that the BTRNN and PBTRNN learned to represent the noise in the train data very well but did not have enough noise data available to generalize well to other noise types. Note that the ability of the SPLICE algorithm to perform well on mismatched noisy data may be caused by the fact that separate models are used for each type of noise and noise condition.

4.1. Computational Efficiency

It is difficult to perform a fair assessment of the computational efficiency of the methods because this is implementation dependent. Nonetheless, we measured the time it took for the models to process the first 1000 utterances from the train set. The simulations were done on a system with an Intel i7 quadcore CPU and a Geforce GTX 480 GPU card installed. For all models, we report the computation time for both the GPU and the CPU.

Table 3: *Time in seconds it took for each model to process the first 1000 utterances of the train set using either the CPU or a GPU to perform computations.*

Model	CPU	GPU
DRDAE	15.4	20.6
MLP	7.7	1.2
BTRNN	56.0	10.2
PBTRNN	25.1	5.1

Computations on the CPU made use of the Intel MKL BLAS library with multi-threading enabled.

Since all methods were implemented in the Theano toolbox, their efficiency depends on the optimizations this software is able to perform for the various computational graphs. These implementations could probably be made more efficient by optimizing them manually.

As Table 3 shows, the standard BTRNN does not perform much faster than the DRDAE but the PBTRNN is about three times faster. This is a significant speed-up that will allow the PBTRNN to be applied to larger datasets.

5. Discussion

We introduced two variants of a bidirectional truncated recurrent neural network architecture for speech denoising that shares properties with deep learning and convolutional neural networks. The two models outperform similar neural architectures and the ETSI2 advanced front end for noise types that were present in the training data. Since the deep recurrent denoising auto-encoder we compared our models with can use more context information, we suspect that our models performed better because they are easier to optimize.

In contrast to more common recurrent architectures that only have access to information about the past, our models also use information about future frames. The models could still be applied in a setup where the speech frames arrive one at the time but in that case it would probably be best to re-update the hidden units after more information has become available.

The fact that the models were overfitting on the specific noise types suggests that their generalization properties could be improved by extending the training data with more types of noise at the potential cost of performing slightly worse on test set A. It should be straightforward to construct such a dataset. We also showed that our models are more computationally efficient than similar architectures. This increases their potential to scale to larger datasets.

Future work should investigate the applicability of our models on larger datasets for which the training data has been augmented with artificially generated noise as has been done in other work on robust speech recognition [20]. It would also be interesting to combine our approach with other techniques for noise robust ASR.

It was beyond the scope of this work to investigate the influence of the number of iterations for which the hidden units are updated. The value we chose was based on prior intuitions about the task and the influence of this parameter on the performance of the models should be investigated in more detail.

6. Acknowledgements

This article was written under partial support by the FWO Flanders project G.0088.09.

7. References

[1] A. L. Maas, Q. V. Le, T. M. O'Neil, O. Vinyals, P. Nguyen, and A. Y. Ng, "Recurrent neural networks for noise reduction in robust asr," in *INTERSPEECH*. ISCA, 2012.

[2] "Advanced front-end feature extraction algorihtm," Tech. Rep. ETSI ES 202 050, 2007.

[3] M. J. F. Gales, "Model-based techniques, for noise robust speech recognition," Ph.D. dissertation, University of Cambridge, 1995.

[4] L. Deng, A. Acero, J. L. Droppo, and J. X. Huang, "High-performance robust speech recognition using stereo training data," in *Proc. ICASSP*, Salt Lake City, UT, May 2001, pp. 301–304.

[5] H. Hermansky, D. P. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional hmm systems," in *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, vol. 3. IEEE, 2000, pp. 1635–1638.

[6] S. Tamura and A. Waibel, "Noise reduction using connectionist models," in *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*. IEEE, 1988, pp. 553–556.

[7] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[8] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

[9] T. Robinson, M. Hochberg, and S. Renals, "Ipa: Improved phone modelling with recurrent neural networks," in *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, vol. 1. IEEE, 1994, pp. I–37.

[10] O. Vinyals, S. Ravuri, and D. Povey, "Revisiting Recurrent Neural Networks for Robust ASR," in *ICASSP*, 2012.

[11] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[12] M. J. Wainwright and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Found. Trends Mach. Learn.*, vol. 1, no. 1-2, pp. 1–305, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1561/2200000001

[13] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *Signal Processing, IEEE Transactions on*, vol. 45, no. 11, pp. 2673–2681, 1997.

[14] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[15] D. Pearce, H. gnter Hirsch, and E. E. D. Gmbh, "The aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions," in *in ISCA ITRW ASR2000*, 2000, pp. 29–32.

[16] J. Martens, "Deep learning via hessian-free optimization," in *ICML*, J. Fürnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 735–742.

[17] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proc. 28th Int. Conf. on Machine Learning*, 2011.

[18] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math compiler in python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 3 – 10.

[19] J. Droppo, L. Deng, and A. Acero, "Evaluation of splice on the aurora 2 and 3 tasks," in *International Conference on Spoken Language Processing*, 2002, pp. 29–32.

[20] J. F. Gemmeke and T. Virtanen, "Artificial and online acquired noise dictionaries for noise robust ASR," in *INTERSPEECH*, 2010, pp. 2082–2085.