RECURRENT NEURAL NETWORKS FOR VOICE ACTIVITY DETECTION

*Thad Hughes and Keir Mierle**

Google, Inc.
thadh@google.com, mierle@gmail.com

ABSTRACT

We present a novel recurrent neural network (RNN) model for voice activity detection. Our multi-layer RNN model, in which nodes compute quadratic polynomials, outperforms a much larger baseline system composed of Gaussian mixture models (GMMs) and a hand-tuned state machine (SM) for temporal smoothing. All parameters of our RNN model are optimized together, so that it properly weights its preference for temporal continuity against the acoustic features in each frame. Our RNN uses one tenth the parameters and outperforms the GMM+SM baseline system by 26% reduction in false alarms, reducing overall speech recognition computation time by 17% while reducing word error rate by 1% relative.

Index Terms— Voice activity detection (VAD), endpointing, recurrent neural networks (RNNs)

1. INTRODUCTION

Speech is a complex audio signal influenced by many factors, including speaker characteristics and environmental conditions. As a pre-processing step before automatic speech recognition (ASR), it is useful to determine which portions of audio contain speech, both to reduce ASR computation and to guide speech user interfaces. This classification, called voice activity detection (VAD), is difficult because of the wide variation of speech and non-speech signals.

Current VAD techniques typically use a classifier to make speech/non-speech predictions about each audio frame independently, together with a temporal smoothing scheme to reduce noise in the classifier's output. One way to build a VAD system involves two GMMs, one trained on speech frames and the other on non-speech frames, to predict the per-frame likelihood of speech, followed by an ergodic hidden Markov model (HMM) that penalizes transitions between speech and non-speech states to give temporal continuity to the prediction [1]. Recent work has investigated both different kinds of features and more powerful classifiers [2].

A problem inherent to many current VAD techniques is that the models used for frame classification and temporal smoothing cannot be easily optimized simultaneously. With HMM-based smoothing, the Markov assumption postulates conditional independence of all observed frames given their discrete hidden state. Further, common HMM training algorithms like Baum-Welch [3] cannot learn a useful state space from data; rather, they assume the state space is specified a priori. In ASR, for example, phonetic lexicons and triphone units [4] often help define the HMM's state space. HMM VAD systems typically have a small number of hidden states, often just one for speech and one for non-speech. Using more hidden states is possible, but unlike typical ASR training where the transcription explicitly specifies a state sequence, there is no straightforward way to decide which VAD HMM states to use or how those states should relate to the training data. This imposes two related limitations on the HMM VAD model: first, processing each frame independently fails to account for the lack of temporal conditional independence of speech frames, and second, the small, discrete HMM state space implies that the model cannot "remember" much about the past. As in ASR, these limitations can be mitigated by providing the classifier with several frames to give it more temporal context, but this increases the number of parameters and computational load.

RNNs address these limitations because they can be discriminatively optimized for frame classification while simultaneously learning a useful, factored, continuous state space and its non-linear temporal dynamics.

2. RECURRENT NEURAL NETWORK MODEL

RNNs are parameterizable models representing computation on data sequences. Like feed-forward neural networks (NNs), which model stateless functions over $\mathbb{R}^m \rightarrow \mathbb{R}^n$, an RNN's computation is factored into nodes, each of which evaluates a simple function mapping its input values to a single scalar output. Unlike NNs, RNN nodes can receive input from nodes at previous timesteps, which allows them to store and manipulate state as they iteratively process sequences of inputs and generate sequences of outputs. RNNs are thus closely related to digital infinite impulse response (IIR) filters.

2.1. Quadratic nodes

Instead of the traditional weighted sum and non-linear activation of multi-layer perceptrons (MLPs) [5], our RNN nodes

*The second author performed the work while at Google, Inc.

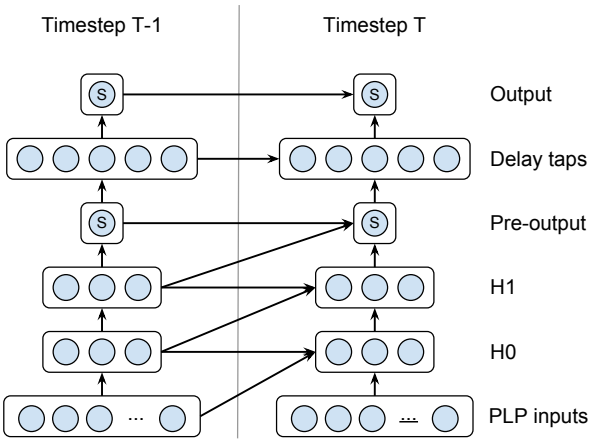

Fig. 1. Our architecture is a feed-forward NN with recurrence added at various points. Nodes marked with S have $\tanh$ non-linearities; others achieve non-linearity by evaluating quadratic polynomial functions of their inputs.

compute quadratic functions of their inputs, followed by an optional non-linearity:

$$V(x) = f\left(x^T W_Q x + w_L^T x + w_B\right) \quad (1)$$

A node computes its output value $V(x)$ from the vector x of its inputs using Eq. (1); W_Q is an upper-triangular sparse matrix with weights for quadratic terms, w_L is a vector of linear weights similar to those in MLPs, and w_B is a scalar bias. Motivating this approach is the idea that higher-order Taylor polynomials can reasonably approximate more functions than affine functions can. This representation can compute products, similar to the Multiplicative RNNs described in [6], and such nodes can also evaluate the multidimensional Gaussian density (and other radial basis functions), since $\mathcal{N}(x;\mu,\Sigma)$ can be written $exp(-x^T\Sigma^{-1}x + 2\mu^T\Sigma^{-1}x - \mu^T\Sigma^{-1}\mu + \ln(z))$, where z is the Gaussian normalization constant.

2.2. VAD RNN architecture and initialization

Our RNN VAD architecture, shown in Fig. 1, is different from the MLP-like RNNs others have applied to VAD [7, 8], speech de-noising [9], and ASR [10] because it uses quadratic nodes as described in section 2.1 and because the structure augments multi-layer feed-forward NNs with recurrent connections. The connectivity is depicted in Fig. 1, where an arrow pointing towards a node indicates that the node receives as inputs the values of the nodes at the arrow's tail. For example, if $H1[T]$ denotes a vector of the outputs of the nodes in layer $H1$ at timestep T, then for all nodes n in layer $H1$, the input vector $x_n[T]$ (when evaluating Eq. 1) is shown in Eq. (2).

The arrows in Fig. 1 describe the connectivity of the network but not its parameterization, which is defined by each node's weights: W_Q, w_L, and w_B . A dense W_Q can represent a more flexible network, but we choose to use sparsity based on our intuition that the product of a node's output at the current and previous timesteps would be most useful, since those values are highly correlated. The sparsity pattern of W_Q for all nodes in layer $H1$ is also shown in Eq. (2); other layers follow the same pattern:

$$x_n[T] = \begin{pmatrix} H0[T] \\ H0[T-1] \\ H1[T-1] \end{pmatrix}, \; sparsity(W_Q) = \begin{pmatrix} I_{3x3} & I_{3x3} & 0 \\ 0 & I_{3x3} & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad (2)$$

Thus each node has quadratic weights in W_Q for the squares of its vertical and diagonal input nodes, and the products of pairs of the same input nodes from the current (vertical) and previous (diagonal) timesteps. Each node also has linear weights w_L for all its inputs, a bias w_B, and a trainable parameter for its initial value at timestep $T-1$.

Our model is initialized as a feedforward NN, where the linear weights in w_L for vertical inputs are randomly initialized with zero mean and small variance, and horizontal and diagonal linear weights in w_L and all weights in W_Q and w_B are initialized to zero. However, the linear weights w_L associated with the nodes in the tapped delay layer are initialized with 0s and 1s so that these nodes function as a "shift register," as described in [11]. This allows the final output node to use 5 timesteps of recent history when computing its value, similar to the baseline GMM VAD's state machine smoothing. However, the network is free to adjust the tap weights (making it no longer a delay line) to improve the final error.

We compute the RNN's error at each timestep by running it on training data. For each timestep T, we set the RNN's input nodes with the input features, compute each node's output using Eq. (1), and compute the difference of the RNN's output node $N_{\text{Output}}[T]$ with a slightly delayed target output:

$$\text{Error}[T] = N_{\text{Output}}[T] - \text{Target}[T-\Delta] \quad (3)$$

The fixed delay Δ allows the RNN to process input frames up to $T+\Delta$ before outputting its decision about frame T, giving it both extra context and extra timesteps to compute. Unless otherwise noted, we use $\Delta = 10$.

3. TRAINING PROCEDURE

We train our RNNs using supervised pairs of input and target output sequences. Unless otherwise noted, the inputs are 13-dimensional PLP features, without deltas or double-deltas. The target output at each timestep is a single value indicating whether the frame Δ timesteps ago is speech or non-speech, and is generated by forced alignment of the audio with a human transcription using a monophone acoustic model.

3.1. Ceres Solver

RNN training involves searching for parameter values that make the RNN's predicted outputs match the supervised target outputs for the given input sequences. This is a non-convex problem, but given a reasonable initialization, a gradient-based approach often finds a good solution. We use Ceres Solver [12], an open-source C++ non-linear least-squares minimizer to perform this optimization. Similar to [13], Ceres Solver is a Levenberg–Marquardt optimizer that only requires the Jacobian matrix containing the partial derivatives of each residual (error term) with respect to each parameter. Ceres attempts to minimize the sum of the squares of all errors from Eq. (3) over all utterances in the training set, plus a scaled L^2-norm of the RNN's parameters.

3.2. Automatic differentiation

In addition to the residuals, Ceres also requires a Jacobian matrix containing the first partial derivative of each residual with respect to each parameter. Eq. (2) is recurrent because node outputs are fed back as inputs, making Eq. (3) inconvenient and error-prone to differentiate symbolically, so we use forward automatic differentiation (autodiff) [14] to compute these derivatives. Forward autodiff is beyond the scope of this paper, but it allows us to re-use, literally in place, the same C++ code that calculates the residuals in Eq. (3) to also compute their exact first derivatives, which greatly simplifies experimentation. The computation is equivalent to back-propagation through time [5] (itself an instance of reverse autodiff), but forward autodiff is simpler to implement and more efficient when the number of parameters is much smaller than the number of residuals (which scales with the duration of the training data).

3.3. Two-stage training

We perform two stages of training. In the first stage, we fix all recurrent parameters and only train the feedforward parameters (those associated with vertical arrows in Fig. 1). In this stage, the RNN's only memory is provided by the tapped delay line's fixed parameters. In the second stage, we optimize all the parameters together, including the w_L weights controlling the tapped delay line. Like [10, 11], we found that this scheme prevents the recurrent parameters from settling into an irremediably poor local optimum early in the training.

4. EXPERIMENTS

We trained and evaluated several variations on our RNN architecture, described in Table 1. The best variation, called RNN$_A$, is exactly as described in Section 2.2. RNN$_B$ is identical to RNN$_A$, but trained with $\Delta = 5$, so RNN$_B$ outputs its decision after processing only 5 frames of future context instead of the default 10. RNN$_C$ is like RNN$_A$, except that

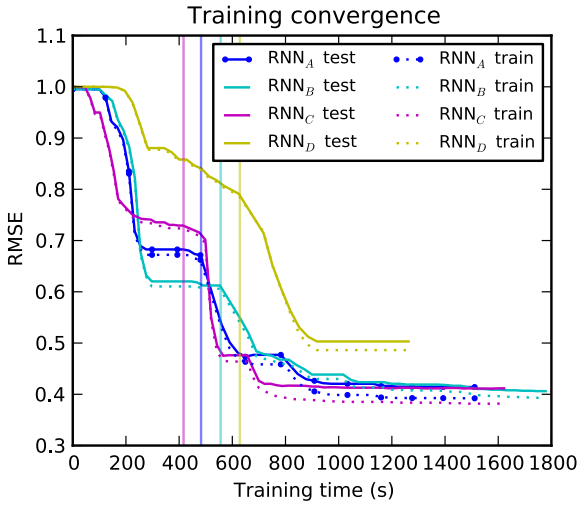

Fig. 2. RNNs A-C, with quadratic nodes and multiple layers, converge faster and better than RNN$_D$, with non-quadratic nodes (yellow). Vertical lines mark the transition from the first optimization stage (training only feed-forward non-recurrent parameters) to the second stage (training all parameters together), and show the large benefit provided by the recurrent parameters.

it omits the tapped delay line portion of the RNN, turning the pre-output node into the final output and thus reducing the number of parameters. RNN$_D$ uses a completely different, more traditional MLP-like RNN architecture. The input layer is expanded to 39 nodes by adding delta and double-delta features (since the hidden layer cannot compute them in an obvious way), and there are 9 hidden nodes in a single layer, each one computing an affine weighted sum of all current inputs and all hidden nodes at the previous timestep ($W_Q = 0$ for

Name	Params	FA%	Description
RNN$_A$	354	11.2	See Section 2.2.
RNN$_B$	354	12.5	Same as RNN$_A$, but with $\Delta = 5$.
RNN$_C$	302	12.2	Same as RNN$_A$, but without tapped delay line.
RNN$_D$	402	29	Traditional non-quadratic RNN; single hidden layer of 9 recurrent nodes with $\tanh$ non-linearity; no delay line. 39 PLP+delta+double-delta inputs.
GMM +SM	4740	15.1	2x30-component diagonal GMMs with 39 PLP+delta+double-delta features, plus SM with 15-40 timesteps of memory.

Table 1. RNN architectures, with number of parameters and false accept rate (lower is better) when false reject is 2%.

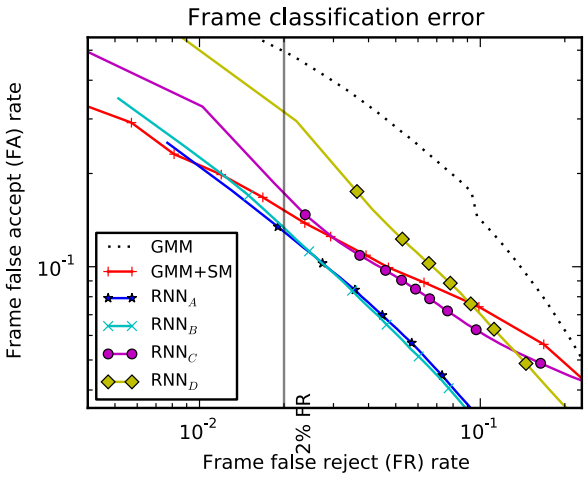

Fig. 3. Quadratic, multi-layer RNNs A-C beat the GMM+SM baseline (red) at our typical operating point of 2% FR (points closer to the origin are better).

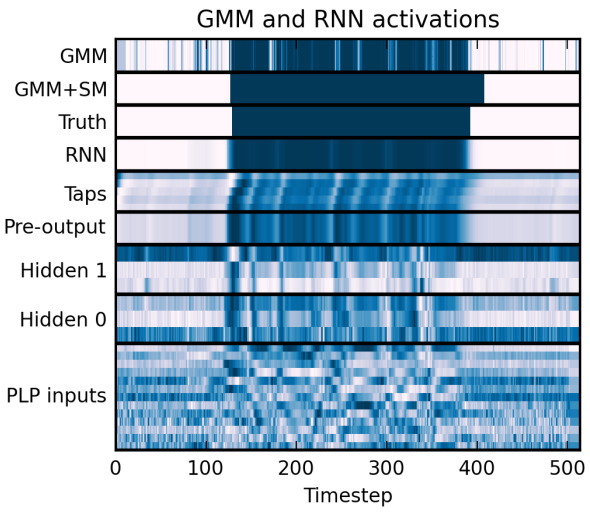

Fig. 4. GMM vs. RNN_A activations for an utterance containing the speech "hello world" show that RNN_A's output at higher levels becomes more stable and accurate than the GMM's speech posterior.

all nodes). Fig. 2 shows the training convergence for each of these models when trained on 1000 utterances and tested on another 1000 utterances, where approximately half the frames are labeled as speech. The multithreaded training was done using a single 12-core 2.67GHz CPU.

We compare the performance of the RNN systems against a strong baseline: our GMM+SM system, which uses 13-dimensional PLP features and their deltas and double-deltas, combined with a hand-tuned state machine (SM) that performs temporal smoothing by identifying regions where many frames exceed a threshold, and emitting its decision with a 19 frame delay. The GMM portion of the baseline uses two 30-component diagonal-covariance GMMs, trained using maximum likelihood estimation on speech and non-speech frames. Fig. 3 shows that most of the RNN systems dominate the GMM+SM baseline at almost all operating points. Since an ASR system can recover more easily from false accept (FA) errors than false reject (FR) errors, we target FR rate to around 2%.

Interestingly, RNN_D, the traditional MLP-like RNN similar to [7], performs conspicuously more poorly than the other RNNs, which have multiple layers and quadratic nodes, suggesting that these features may help the performance of RNNs.

Finally, we evaluated the real-life performance of RNN_A by embedding it in the ASR system described in [15] and testing it on 27,000 utterances averaging 4.4 seconds in duration. The RNN's lower computational load and lower false accept rate reduce overall ASR computation time by 17% (relative) while modestly improving the word error rate (WER) by 1% (relative).

Fig. 4 reveals how the RNN improves on the GMM+SM baseline. The GMM speech posterior, shown at the top, contains many spurious peaks that should not be labeled as speech, whereas the RNN is able to eliminate many of them. Note that the RNN's pre-output still shows some of these errors, but they are removed by the tapped delay line. In this case the GMM+SM was also able to recover from the GMM's errors, but in other cases it cannot.

5. CONCLUSION

We have shown that our RNN architecture can outperform considerably larger GMM-based systems on VAD tasks, reducing the per-frame false alarm rate by 26% and thus increasing overall recognition speed by 17%, with a modest 1% relative decrease in the word error rate. Our RNN architecture, with multiple layers and quadratic nodes, also seems to outperform traditional MLP-like RNNs, which suggests applying it to other ASR-related tasks such as feature computation and acoustic modeling. The non-convexity of RNN optimization also leaves open the possibility that combining gradient-based optimization with less localized search techniques such as genetic algorithms may find even better solutions.

6. ACKNOWLEDGEMENTS

Thad Hughes would like to thank these people: Christopher Manning and Daniel Jurafsky for introducing me to ASR and factored graphical models, Stephen Boyd for teaching me dynamical systems and convex optimization, my colleagues at Google, especially Sameer Agarwal, and my wife Carolina Tropini for inspiring me every day!

7. REFERENCES

[1] J. Sohn, N.S. Kim, and W. Sung, "A Statistical Model-based Voice Activity Detection," *Signal Processing Letters, IEEE*, vol. 6, no. 1, pp. 1–3, 1999.

[2] Ananya Misra, "Speech/Nonspeech Segmentation in Web Videos," in *Proceedings of InterSpeech 2012*, 2012.

[3] L.E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains," *The annals of mathematical statistics*, pp. 164–171, 1970.

[4] Steve J. Young, J.J. Odell, and P.C. Woodland, "Tree-based state tying for high accuracy acoustic modelling," in *Proceedings of the workshop on Human Language Technology*. Association for Computational Linguistics, 1994, pp. 307–312.

[5] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 1, pp. 213, 2002.

[6] Ilya Sutskever, James Martens, and Geoffrey Hinton, "Generating Text with Recurrent Neural Networks," in *28th International Conference on Machine Learning (ICML)*, 2011.

[7] R. Gemello, F. Mana, and R. De Mori, "Non-linear estimation of voice activity to improve automatic recognition of noisy speech," in *Proceedings of Interspeech 2005*, 2005.

[8] G.D. Wu and C.T. Lin, "A recurrent neural fuzzy network for word boundary detection in variable noise-level environments," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 31, no. 1, pp. 84–97, 2001.

[9] A. Maas, Q. Le, T. O'Neil, O. Vinyals, P. Nguyen, and A. Ng, "Recurrent Neural Networks for Noise Reduction in Robust ASR," in *Proceedings of INTERSPEECH*, 2012.

[10] O. Vinyals, S.V. Ravuri, and D. Povey, "Revisiting Recurrent Neural Networks for Robust ASR," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. IEEE, 2012, pp. 4085–4088.

[11] Oliver Obst and Martin Riedmiller, "Taming the Reservoir: Feedforward Training for Recurrent Neural Networks," in *Accepted at IJCNN 2012*, 2012.

[12] Sameer Agarwal and Keir Mierle, *Ceres Solver: Tutorial & Reference*, Google Inc.

[13] James Martens and Ilya Sutskever, "Learning Recurrent Neural Networks with Hessian-Free Optimization," in *28th International Conference on Machine Learning (ICML)*, 2011.

[14] L.B. Rall, "Automatic differentiation: Techniques and applications," 1981.

[15] Navdeep Jaitly, Patrick Nguyen, Andrew Senior, and Vincent Vanhoucke, "Application Of Pretrained Deep Neural Networks To Large Vocabulary Speech Recognition," in *Proceedings of Interspeech 2012*, 2012.