Some Variations on Training of Recurrent Networks

Gary M. Kuhn and Norman P. Herzberg
Center for Communications Research - IDA

1 Introduction

We describe some variations on training of multi-layered recurrent networks which overcome the need for an externally-supplied target function, avoid back-propagation of error derivatives in time, reduce training time, and enhance generalization.

Applied to a speech recognition problem, these variations resulted in as low a number of training iterations and as high a performance, as those reported for cross-entropy trained, hidden Markov models. However, we find that our recurrent networks have *not* provided a large performance improvement over a competing non-recurrent network with a similar number of weights.

2 The Task

Following [1,2,3] we have $N = 768$ speech signals X_n, selected samples of which we have concatenated into one long signal. Each X_n belongs to one of $K = 4$ classes. Each class represents one of the English letter names "*b*", "*d*", "*e*" or "*v*". We parameterize the speech at each 10*ms* time-frame $t = 1, \ldots, T$ of the concatenated signal, yielding vector observations $x(t)$, $T = 18347$. The parameterization is based on locally normalized filter-bank channel energies $e_i(t)$, $i = 1, \ldots, 16$.

We also have N labels $m_n = (s_n, e_n, a_n)$ which indicate the start sample s_n, the end sample e_n, and the letter name a_n for each X_n. The X_n have different durations. Letter name a_n is spoken once somewhere between time samples s_n and e_n.

Originally appeared in Neural Networks: theory and applications, Richard J. Mammone and Yehoshua Y. Zeevi Eds. New York, Academic Press, ISBN 0-12-467050-4, p. 233-244, 1991.

We set up a network with sigmoid units and one hidden layer. We want to train the network to accept each *x(t)* in turn as input, and for each input, to produce a *K*-dimensioned vector *o(t)* as output. We want output unit *k* to turn on only while a pattern from class *k* is present at the input.

We believe that the response of the network to the current input should be a function of earlier inputs. We want the network to learn how to weight the current input relative to earlier inputs. We use a network architecture that can accomplish this learning. One version introduces time delays from lower layers to higher layers.[2,4] The other introduces time delays from one layer to any other, making the network potentially "recurrent".[3,5]

We now describe variations on network training which have led to solutions to the following problems: (1) how to overcome the need for an externally-supplied target function, (2) how to train the recurrent network without back-propagating the derivatives of error in time, and (3) how to reduce training time and enhance generalization.

3 Derive the Targets from an Input "Event" Parameter

This section presents an idea reported in [3]. We add a new speech parameter, a *temporal difference operator*, *r(t)*, defined over a time window of width $2\Delta t$ and filter-bank frequency channels $i = 1, \ldots, 16$, by

$$r(t) = \frac{\alpha}{16\Delta t} \sum_{\tau,i} max\,[0, e_i(t+\tau) - e_i(t+\tau-\Delta t)],$$

where $\tau = 0, \ldots, \Delta t - 1$, and $\Delta t = 10$ time frames. We use $\alpha = 2$, and clip at 1.

r(t) is a shape-dependent energy increase indicator, which turns on at those time frames where the vowel turns on *r(t)* can be thought of as an "event" signal: when it turns on, *something* is happening.

The *target function* is based on *r(t)*. Let lag *g* be the (positive) number of time frames from the appearance of a value of *r(t)* at the network input to its desired appearance on the output of output unit a_n. Then for each time frame *t* from s_n through e_n,

$$targ_{an}(t) = r(t - g),$$

and

$$targ_k(t) = 0,\ k \neq a_n\ .$$

During time frames s_n to $s_n + g - 1$, the target values are zero if $n = 1$, or are based on values of *r(t)* in the preceding X_n if $n > 1$.

The main point is that the network should learn to indicate which event happened by reproducing input parameter *r(t)* with a lag of *g* time frames on output unit a_n. To do so, it can use all of the input parameters, including *r(t)*.

4 Forward Propagate the Derivatives of the Potentials

This section presents an idea reported in several places. See [3,5-10] and the review [11].

To train network weight matrix *W*, we minimize *E*, the squared difference between observed and target outputs summed over all output units and time frames. Let weight *w*, from unit *a* to unit *b* with delay *d*, be denoted w_{abd}. We need equations for the gradient of *E* with respect to each weight w_{abd}.

Let *p(t)* represent the weighted sum, or "potential" into a unit at time *t*. We factor the derivative of the error at output unit *k* with respect to weight w_{abd} as

$$\frac{\partial E_k}{\partial w_{abd}} = \sum_t \frac{\partial E_k(t)}{\partial p_k(t)} \frac{\partial p_k(t)}{\partial w_{abd}}.$$

Note that in a multi-layer recurrent network, the error at time *t* depends on potentials at all earlier times, except for those potentials in lower layers of the network that are so recent that they have not yet affected the error-producing outputs of the network.

Two training variations for the recurrent network come immediately to mind. We might store all unit outputs on a forward pass and back-propagate the error derivative to $t = 1$, accumulating the negative derivative of the error with respect to the potential of unit *k*, at all time frames from *T* back to each *t*.[5] Or, on the forward pass, at each *t* we might compute all unit outputs and then propagate the error derivative back only to time frame $t - \tau$, yielding a truncated approximation to the gradient.[5]

Wouldn't it be nice if we could propagate the derivative of the unit potential forward, and multiply by the current error derivative at the same time as the network error is generated. Then we could avoid any backward pass.

This is, of course, possible. Carrying out the differentiation, we obtain a forward recursion at any unit *c*,

$$\frac{\partial p_c(t)}{\partial w_{abd}} = \sum_{x,l} \left(w_{xcl} \frac{\partial o_x(t-l)}{\partial p_x(t-l)} \frac{\partial p_x(t-l)}{\partial w_{abd}} + \begin{cases} o_x(t-l), & \text{if } (x,c,l) = (a,b,d); \\ 0, & \text{otherwise} \end{cases} \right). \quad (1)$$

From (1) we see that the difference between a feed-forward network and a recurrent network is that for the recurrent network, the derivative of a unit's potential can have other terms than just the output of the unit on the input end of the connection.

The forward recursion has the *disadvantage* that the number of partials to carry forward can get large. The number of partials can be controlled by limiting feed-back to so-called "context" units [12, 13], or by limiting the number of feedback connections, *e.g.* by using only self-loops.

For the recurrent networks which we report on here, feed-back was limited to self-loops, either on output units, or on the hidden units, and back-propagation in time was avoided using forward propagation of the derivative of the unit potentials.

5 Try Adaptive Training

To reduce training time and enhance generalization, we consider variations on (1) weight initialization, (2) frequency of weight updates, (3) adaptation of the step size, (4) adaptation of the search direction, (5) gradient evaluation during events, and (6) assignment of examples to training and cross-validation data subsets. A third and entirely separate dataset is, of course, kept for testing.

5.1 Initialization

Random initialization of the weights may produce a network which cannot be trained. One remedy is to *adjust* the initial weights, as follows. Evaluate E and $\partial E/\partial W$ over a small but representative number of training examples. Then, for each w_{abd}, see whether the sign of w_{abd} is the same as the sign of $\partial E/\partial w_{abd}$. If so, flip the sign of w_{abd} and re-evaluate E. If the new E is better than the previous E, keep the sign change and set E to the new E. Continue for all weights.

Applied to just 4 balanced sets of training examples, this weight adjustment takes no more time than a couple of training iterations, increases the probability that the network will be trainable, and reduces E by about a factor of 2.

5.2 Frequency of Weight Updates

In the early stages of training, a rough estimate of $\partial E/\partial W$ can be used to point the weight changes quickly in a good, if not a perfect, direction. One schedule for producing increasingly refined estimates of $\partial E/\partial W$ is the following [14]. On each iteration i through the training examples, set the size of the balanced block of examples over which $\partial E/\partial W$ is calculated and after which W is updated, to iK.

Unfortunately, for our examples, small blocks at the initial values of i produce large fluctuations in E. Also, we want to avoid increasing block size if training can continue faster at the current size.

As a result, we propose a modified schedule. Let the initial block size be the same as the number of examples over which W was initialized, and divide the training examples into training and cross-validation subsets. Here, we use a ratio of 4 to 1.

Then on each iteration, evaluate $\partial E/\partial W$ and update W on each block in the training subset. At the end of the iteration, evaluate E_v, the error on the cross-validation subset. Keep the current block size until the number of increases in E_v at the current block size equals n_{Ev}. Then, if possible, increase the number of balanced sets of examples per block by 1. We set $n_{Ev} = 2$, to tolerate one uptick but no oscillation.

5.3 Adaptation of the Search Direction

Rather than use ΔW (= $-\partial E/\partial W$ on the current block) with a heuristic momentum factor, the authors of [15] allowed search direction Z to be given by the conjugate gradient [16]

$$Z \leftarrow \Delta W + \gamma Z$$

with γ computed from the Polak-Ribiere formula

$$\gamma = (\Delta W - \Delta W_p) \cdot \Delta W \,/\, (\Delta W_p \cdot \Delta W_p).$$

We find that even if the previous negative gradient, ΔW_p, is computed on a different block, E falls much faster with a computed γ than with a constant value for γ.

5.4 Adaptation of the Step Size

Having computed some weight-change direction Z on the current block, we search along Z for the best weight matrix $\hat{W}$ satisfying

$$\hat{W} = W + s\eta Z,$$

with $\eta = .005$. We try the following sequence of values for s. Initially, s is set to the smallest s accepted over the last $\lambda = 6$ blocks. Then $E_{\hat{W}}$ is computed. Then $s \leftarrow \varphi^x s$, where $\varphi = 1.61$, and x is either +1 if $E_{\hat{W}}$ is an improvement or -1 if not. We accept s when $E_{\hat{W}}$ gets worse *and* either $s = 1$, or at least three values have been tried for s.

If three values have been tried, the accepted s may be moved under the minimum of a quadratic fitted to the last three (s, E) pairs. Finally, W is updated using the accepted s.

5.5 Gradient Evaluation during Events

Even though the target function is defined for all t, a threshold ρ can be established so that $\partial E/\partial W$ is evaluated only when $r(t - g) \geq \rho$ signals an event. This provides some speed-up and allows the network to learn completely from event data. Matrix ∂P/∂W is still computed at each t. Here, we experiment with $\rho = 0.25$.

5.6 Randomized Training and Cross-Validation Subsets

The network tends to overtrain, *i.e.*, to pass from an initial stage during which E_t (the error on the training subset) and E_v are both reduced, to a second stage during which only E_t is reduced. Also, we have barely enough examples to train the weights. We want a way to control overtraining while keeping all examples not in the test set *available* for the training subset.

We propose the following procedure. On each iteration, re-assign the training examples *randomly* to the training or cross-validation subsets. The ratio of their sizes is still 4 to 1.

With random re-assignment to the subsets, the instantaneous errors E_t and E_v now track each other closely. More importantly, neither error falls so far that it becomes a bad predictor of error on the *test* dataset. Do they fall as far as possible consistent with good prediction of error on the test dataset? We do not know.

6 Simulation Results

Figure 1 shows the value of several quantities as a function of training iteration. Quantities 1 and 2 are E_t and E_v, the training and cross-validation training errors. Quantity 3 is proportional to the number of blocks in the training subset. Quantity 4 is the root mean square of the weights, which seems to have stopped growing by iteration 100. Quantities 5 and 6 are the fraction of training and cross-validation examples correctly recognized. Because the training subset is much larger, quantity 5 looks less noisy.

A recognition score is defined for the network response to concatenated presentation of those X_n kept in a test dataset.[3] The score $E_{u|n}$ for each letter name u on each X_n is the sum over time frames in X_n, of the squared difference between observed and expected network outputs, under the hypothesis that X_n is an occurrence of letter name u. The expected outputs are the same as the target values that would have been used if X_n were a training example of letter name u. The network "recognizes" X_n if $argmin_u$ $E_{u|n}$ is a_n. We now redefine $E_{u|n}$ over frames where $r(t - g) \geq \rho$.

In [3], the letter-name discrimination task is attacked using a network with one hidden layer, 4 time delays from the input units to each of 8 hidden units, and 3 time delays from the hidden units to the output units. Delays to the hidden units are 1 time-frame apart. Delays to the output units are 2 time-frames apart.

A version of that network *without* self-loops on the output units was trained on 372 X_n for 1000 training iterations, and recognized 83.6% of 396 test X_n. Adding self-loops on the output units late in training produced a network that recognized 84.6% of the test X_n.

The same non-recurrent network now trains on 672 X_n in only 100 iterations, and each iteration takes half as much time. After 100 iterations of training the non-recurrent network recognizes 88.5% of 128 test X_n. Adding self-loops on the output units and training for another 10 iterations produces a recurrent network that recognizes 89.6% of the test X_n.

In an additional simulation, a single delay was used from hidden to output units, the number of hidden units was increased to 9, and self-loops were used only on the hidden units. For this network, the total number of weights was nearly identical to the number in the network with multiple hidden-to-output delays but no recurrence. Recognition results for this additional network only reached 86.5% after 100 iterations.

The number of iterations needed to train our best recurrent network is now similar to the number needed to train hidden Markov models to cross-entropy criteria.[17] Also, the performance of our best recurrent network, 89.6%, is now as good as the 89% estimated for cross-entropy trained, hidden Markov models.[1,2]

A recently reported "scanning" feedforward time-delay neural network has apparently obtained connected recognition performance of 90.9% on this dataset.[18] Unfortunately, the recurrent, the hidden Markov, and the scanning system all perform significantly below the human level of 95%. The goal remains to reach the human level of performance, which is even 97% on our full database of 9-letter names: "*b, c, d, e, g, p, t, v*" and "*z*".

7 Conclusions

The network training variations described above reduced the number of training iterations by a factor of 10, reduced training time by a factor of 20, and contributed to a reduction in recognition errors by 33%.

Applied to a speech recognition problem, these variations resulted in as low a number of training iterations and as high a performance, as those reported for cross-entropy trained, hidden Markov models. However, we find that our recurrent networks have *not* provided a large performance improvement over a competing non-recurrent network with a similar number of weights.

8 Acknowledgements

This is an expanded version of paper TP6.1 given at the 24th annual Conference on Information Sciences and Systems, Princeton University, March 22, 1990. The authors thank L. Bottou, P. Brown, P. Haffner, B. Ladendorf, K. Lang, E. Ojamaa and R. Watrous for help with various aspects of this work.

Bibliography

[1] Brown, P.F. *The acoustic-modeling problem in automatic speech recognition*, IBM Computer Science Tech. Report RC 12750, p. 1-119, 1987.

[2] Lang, K.J. and G. Hinton. *The development of the TimeDelay Neural Network Architecture for speech recognition*, Tech. Report CMU-CS-88-152, Carnegie-Mellon University, p. 1-30, 1988.

[3] Kuhn, G., R.L. Watrous and B. Ladendorf. *Connected recognition with a recurrent network*, Speech Communication, Vol. 9, p. 41-49, 1990.

[4] Waibel, A., T. Hanazawa, G. Hinton, K. Shikano and K. Lang. *Phoneme Recognition using time-delay neural networks*, IEEE Trans. ASSP, Vol 37, p. 328-339, 1989.

[5] Watrous, R.L., B. Ladendorf and G. Kuhn. *Complete gradient optimization of a recurrent network applied to |b|, |d|, |g| discrimination*, J. Acoust. Soc. Amer., Vol. 87, p. 1301-1309, 1990.

[6] Kuhn, G. *A first look at phonetic discrimination using a connectionist network with recurrent links*, CCRP - IDA SCIMP Working Paper No. 4/87, p. 1-41, 1987.

[7] Robinson, A.J., F. Fallside. *Static and dynamic error propagation networks with application to speech coding*, in D.Z. Anderson (ed.), Neural Information Processing Systems, New York, Amer. Inst. Physics, p. 632-641, 1987.

[8] Gori, M., Y. Bengio and R. De Mori. *BPS: A learning algorithm for capturing the dynamic nature of speech*, Proc. IntI. Joint Conf. on Neural Networks, Washington, D.C., Vol. II, p. 417-423, 1989.

[9] Gherrity, M. *A learning algorithm for analog, fully recurrent neural networks*, Proc. Intl. Joint ConL on Neural Networks, Washington, D.C., Vol. I, p. 643-644, 1989.

[10] Williams, R.J., and D. Zipser. *A learning algorithm for continually running fully recurrent neural networks*, Neural Computation, Vol. 1, p. 270-280, 1989.

[11] Pearlmutter, B. *Two new learning procedures for recurrent networks*, Neural Network Review, Vol. 3, p. 99-101, 1990.

[12] Elman, J. *Finding structure in time*, CRL Tech. Report 8801, Univ. of California at San Diego, p. 1-29, 1988.

[13] Zipser, D. *A subgrouping strategy that reduces complexity and speeds up learning in recurrent networks*, ICS Tech. Report 8902, Univ. of California at San Diego, p. 1-5, 1989.

[14] Haffner, P., A. Waibel, H. Sawai and K. Shikano. *Fast backpropagation learning methods for large phonemic neural networks*, Proc. Eurospeech '89, Vol. II, p. 553-556, 1989.

[15] Webb, A., D. Lowe and M. Bedworth. *A comparison of nonlinear optimisation strategies for feed-forward adaptive layered networks*, Royal Signals and Radar Establishment memorandum 4157, Malvern, England, p. 1-33, 1988.

[16] Vapnik, V. *Estimation of Dependencies Based on Empirical Data*, Addendum A, §2. Springer-Verlag, 1982.

[17] Gopalakrishnan, J.S., D. Kanevsky, A. Nadas, D. Nahamoo, M.A. Picheny. *Decoder selection based on cross-entropies*, Proc. IEEE IntI. ConL on Acoustics, Speech and Signal Processing, New York, Vol. I, p. 20-23, 1988.

[18] Lang, K.J., A.H. Waibel, and G.E.Hinton. *A Time-Delay Neural Network Architecture for Isolated Word Recognition*, Neural Networks, Vol. 3, p. 23-43, 1990.

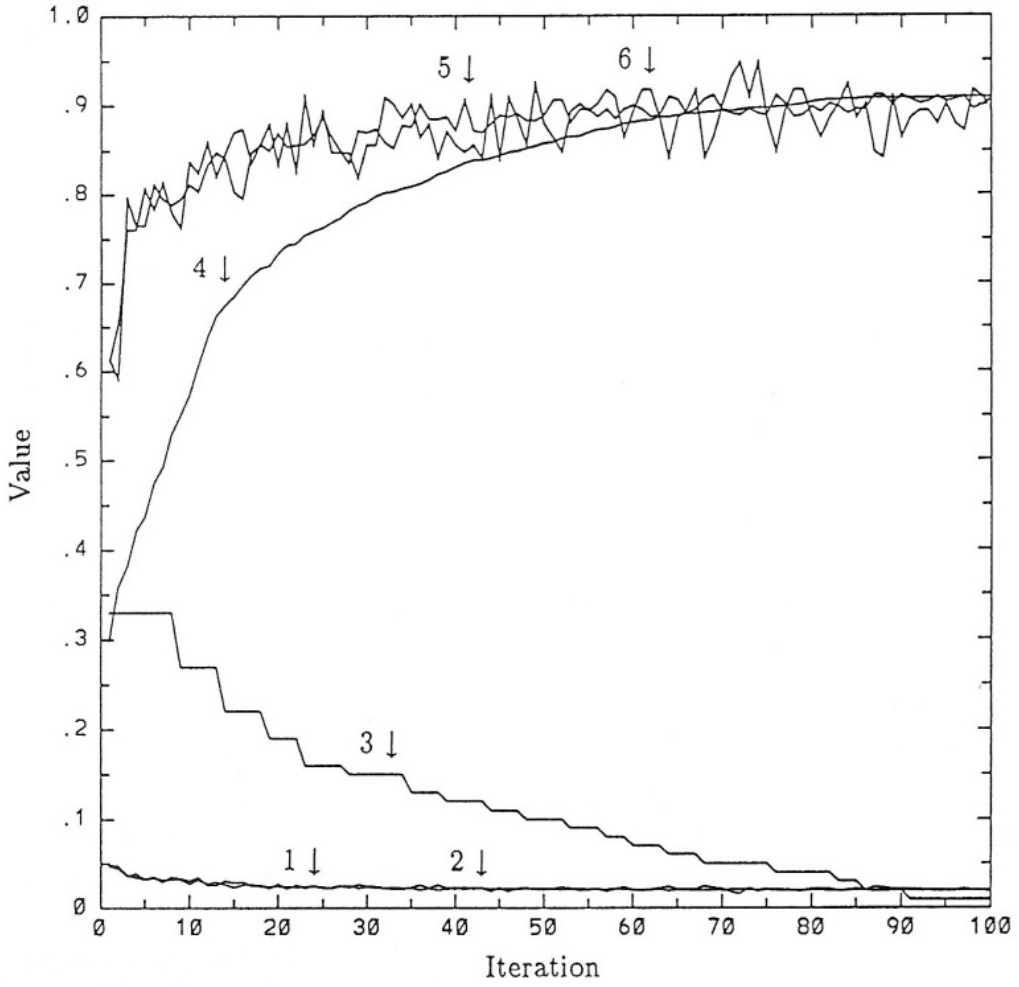

Figure 1: Plot of several quantities as a function of training iteration:

1. Training error E_t

2. Cross-validation error E_v

3. Number of blocks in training subset, divided by 100

4. Root mean square of the weights

5. Fraction of training examples correctly recognized

6. Fraction of cross-validation examples correctly recognized