Backpropagation in Sequential Deep Neural Networks

Galen Andrew
University of Washington
galen@cs.washington.edu

Jeff Bilmes
University of Washington
bilmes@ee.washington.edu

Abstract

Most previous work applying neural networks to problems in speech processing has combined the output of a static network trained over a sliding window of input with an HMM or CRF to model linear-chain dependencies in the output. The recently developed Sequential Deep Neural Network (SDNN) model allows sequential dependencies between internal hidden units, allowing them to potentially detect long-range phenomena. During the upward propagation phase, a binary forward-backward (Baum-Welch) computation is performed for each hidden unit, passing information along the extent of the chain. Then the gradient of the SDNN error can be computed exactly via a modified backpropagation algorithm that also includes an information-passing Baum-Welch-like step. The model introduces minimal computational overhead compared to other DNN approaches to sequential labeling, and achieves comparable performance with a much smaller model (in terms of number of parameters). Experiments on the TIMIT phone recognition dataset show that using sequential information at all layers improves accuracy over baseline models that use a sequence model only in the output.

1 Learning deep representations for sequential labeling

Sequential labeling problems occur in a variety of settings, including speech recognition, audio and video processing, natural language processing, and bioinformatics. The goal of sequential labeling is to map whole sequences of inputs over observations $\{x_1, x_2, \dots, x_T\}$ to sequences of labels $\{y_1, y_2, \dots, y_T\}$, where the length of the sequence T may vary from instance to instance, and where dependencies between observations or labels at different positions within a sequence are used to produce a better labeling than might be achieved by labeling each y_i independently.

In Conditional Random Fields (CRFs) [Lafferty et al., 2001], a Markov Random Field (MRF) is defined over the label sequence whose parameters depend on the input. Typically, the graphical model structure is a linear chain so that globally conditioned on the input, each label is conditionally independent of the other labels given its neighboring labels. CRFs allow features of the input to be defined at arbitrary distance from the associated label, but the user must consciously design such features to allow long-distance dependencies and, moreover, the feature functions may only be fixed length. Feature design is a difficult, task-specific problem, and it is especially difficult to hand-design effective long-range features for tasks such as speech recognition, where the input is a relatively low-level representation of the acoustic signal. On the other hand, results in speech science suggest that longer-range features (that is, longer than the typical 25ms frame width) may be useful for speech perception, particularly in noisy environments [Hermansky and Sharma, 2002].

Several methods have been proposed to introduce hidden variables to CRFs that might be capable of modeling regularities in the data that are not explicit in the features but nevertheless aid in classification. We focus here on approaches that were applied to phone recognition, which is a prototypical sequential labeling task, and the subject of our experiments. The hidden CRF (HCRF) appends a multinomial hidden state to each phone class and optimizes the marginal likelihood [Gunawardana et al., 2005], so that subclasses may be induced that are easier to recognize than the original classes. Another successful approach models each phone as a sequence of three subphones, the boundaries of which are latent [Sung and Jurafsky, 2009]. Other work uses a multi-layer CRF in which the data is mapped through various layers of multinomial sequences that may be either Markov order-1, or order-0 (conditionally independent given the input) [Yu et al., 2010]. All of these approaches are more effective than a single-layer CRF with no latent structure, but even better results have been obtained with a richer latent representation via Deep Neural Networks (DNNs).

Deep Neural Networks have emerged as an empirically very effective model for inducing rich feature representations of static (non-sequential) data [Hinton et al., 2006]. Each layer of latent representation is learned by training a Restricted Boltzmann Machine (RBM) to model the data distribution at the next lower layer, using Contrastive Divergence (CD) or some other update. The expected values of the hidden layer units conditioned on the input can then be used as the representation for further processing. Typically after training several stacked RBMs in sequence, a discriminative classifier is trained using the final layer as the input, and the parameters of the entire chain of feature transformations are then fine-tuned according to a discriminative training criterion.

Several researchers have employed DNNs to learn feature representations for use in phone recognition [Plahl et al., 2012, Hinton et al., 2012]. Mohamed et al. [2009] train a static DNN to classify phones (actually, subphones) which is then combined with an HMM bigram language model, using Viterbi decoding to produce a labeling of the sequence. In later work, a DNN phone classifier is trained jointly with a CRF taking the top hidden layer of the DNN as input [Mohamed et al., 2010]. Other recent work [Veselỳ et al., 2013] includes a model where gradients based on a sequence error in an HMM-based system are passed down through shared-parameter deep neural networks at each time frame — unlike this work, our model allows structures at every level in the deep hierarchy to interact in a manner that still allows for tractable backpropagation style training.

In the present work, we give full details on the tractable training procedure for our Sequential DNN (SDNN) model, which introduces true sequence models in each of the hidden layers of a DNN [Andrew and Bilmes, 2012].[1] Each layer is modeled with an MRF structure called a sequential RBM (SRBM) that allows dependencies between corresponding hidden units at adjacent time frames. Exact sampling of hidden structures given the input and computation of conditional expectations remains tractable in the SRBM—it involves only matrix multiplication and forward-backward computations—so CD training is still possible. As with RBMs, we can stack SRBMs and append a discriminative sequence classifier (in particular, a CRF) atop the final layer. Finally, using a backpropagation-like algorithm, we can discriminatively fine-tune all parameters. This way the model can learn to enforce smoothness in the hidden layers across timeframes, and to allow the hidden units to detect longer-range phenomena.

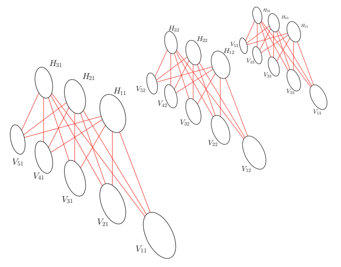
1.1 Notation

We employ the following notation in the sequel. If X is a matrix, the $(i,j)^{\text{th}}$ entry is X_{ij} and the j^{th} column is X_{*j}. The submatrix of columns j through k is $X_{*(j:k)}$. The matrix transpose is denoted X'. If X and Y are matrices (or vectors) of the same dimension, $\langle X, Y \rangle$ denotes $\operatorname{tr}(X'Y)$. If X and Y have the same number of rows, $[X|Y]$ denotes their concatenation.

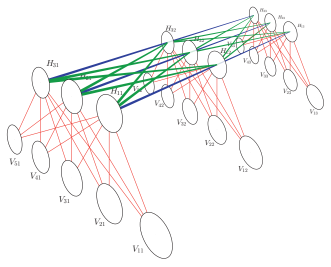
2 The Sequential Restricted Boltzmann Machine

An SRBM defines a joint distribution over two matrix-valued layers, a visible layer $V \in \mathbb{R}^{n_v \times T}$ and a hidden layer $H \in \mathbb{R}^{n_h \times T}$. Conditioned on the hidden layer, all variables of the visible layer are independent (just as in a standard RBM). Conditioned on the visible layer, all *rows* of the hidden layer are independent of each other, but we allow Markov interactions within each row (see Figure 1). While an RBM typically has dense connections between the visible and hidden layers, an SRBM has only edges that are local in time. Specifically, we use edges between V_{it} and $H_{j(t+\delta)}$ for all i, j, t and for $|\delta| \leq \delta_{\text{max}}$. The weights on the edges are summarized in the matrices $W_\delta \in \mathbb{R}^{n_v \times n_h}$, where

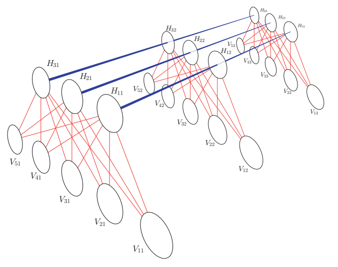
[1] In the cited work, the SDNN was referred to as "Sequential Deep Belief Network", but we feel the terminology used here is more appropriate.



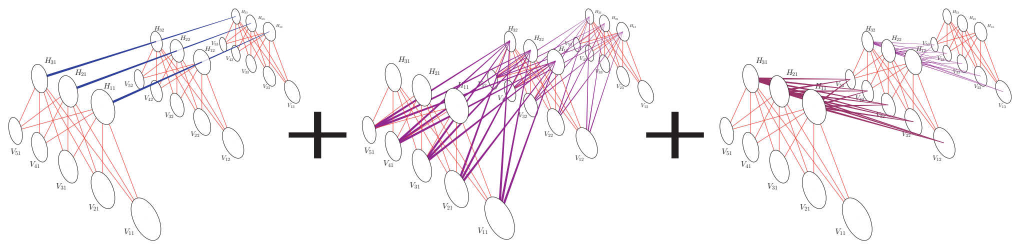
(a) Independent (non-sequential) RBMs



(b) Intractable



(c) SRBM with $\delta_{\text{max}} = 0$



(d) Purple edges in the center figure correspond to W_1, and those on the right, to W_{-1}. The union of the three graphs is an SRBM with $\delta_{\text{max}} = 1$.

Figure 1: Illustrations of SRBM with $T = 3$ time frames, $n_1 = 5$ input units and $n_2 = 3$ hidden units per frame. Fig. 1a shows a sequence of independent (non-sequential) RBMs. Fig. 1b shows a model with dense connections in the hidden layer, which may seem desirable from a modeling perspective, but including all green edges would render contrastive divergence training intractable. Fig. 1c shows an SRBM with $\delta_{\text{max}} = 0$. The red edges correspond to the weights of the matrix W_0, while the blue edges have weights given by θ. Fig. 1d shows the edges corresponding to $W_{\pm 1}$ when $\delta_{\text{max}} > 0$ in purple.

$(W_\delta)_{ij}$ is the weight on all edges $(V_{it}, H_{j(t+\delta)})$. The hidden layer of the SRBM also has a vector of transition parameters $\theta \in \mathbb{R}^{n_h}$ which govern the interactions between adjacent frames within each row of H, as we describe shortly. We intentionally disallow edges between observed units, in order to encourage the hidden layer to model any dependencies between time frames of the observations.

We assume the hidden variables are always binary, meaning $H \in \{\pm 1\}^{n_h \times T}$, and the observed variables are either binary ($V \in \{\pm 1\}^{n_v \times T}$) or real-valued Gaussian ($V \in \mathbb{R}^{n_v \times T}$). For $\delta_{\text{max}} = 1$, the energy of a configuration is defined in terms of the matrix $A^h \in \mathbb{R}^{n_h \times T}$:

$$A^h = \left[W'_{-1} V_{*(2:T)} \middle| \mathbf{0}\right] + W'_0 V + \left[\mathbf{0} \middle| W'_1 V_{*(1:T-1)}\right]. \tag{1}$$

In (1), the middle term $W'_0 V$ produces the matrix of inputs to each hidden unit coming from the visible units at the same time frame (the red edges in the figure). The other two terms add the influence of visible units at the preceding and subsequent frame (the purple edges). The generalization to $\delta_{\text{max}} > 1$ is straightforward.

If both layers are binary, the joint distribution is $\Pr(V, H) \propto \exp -E(V, H)$ where

$$E(V, H) = -\langle H, A^h \rangle - \sum_{t=1}^{T-1} \sum_{j=1}^{n_h} \theta_j H_{jt} H_{j(t+1)}. \tag{2}$$

An examination of (2) reveals the function of the θ parameters. If $\theta_j = 0$, there are no terms involving the products $H_{jt} H_{j(t+1)}$, so the hidden states are independent. If $\theta_j > 0$, the model prefers configurations where $H_{jt} = H_{j(t+1)}$, and in the unlikely case that $\theta_j < 0$, "flip-flopping" hidden state configurations would be preferred.

Defining

$$A^v = \left[\mathbf{0} \middle| W_{-1} H_{*(1:T-1)}\right] + W_0 H + \left[W_1 H_{*(2:T)} \middle| \mathbf{0}\right],$$

note that

$$\Pr(V|H) \propto \exp -E(V,H) \propto \exp\langle H, A^h\rangle = \exp\langle V, A^v\rangle,$$

so the V_{it} are independent given H, with $\Pr(V_{it}|H) \propto \exp A^v_{it} V_{it}$, or $\Pr(V_{it}|H) = \sigma(2V_{it}A^v_{it})$ where $\sigma(x) = (1+\exp -x)^{-1}$.

If the visible layer is Gaussian, then the joint density is $f(V,H) \propto \exp -E(V,H) - \frac{1}{2}\langle V, V\rangle$. Now $f(V|H) \propto \exp\left(\langle V, A^v\rangle - \frac{1}{2}\sum_{it} V_{it}^2\right)$, so the V_{it} are independent and Gaussian-distributed given H, with $V_{it}|H \sim \mathcal{N}(A^v_{it}, 1)$.

Regardless of the type of visible layer, $\Pr(H|V)$ factorizes into terms involving individual H_{jt} and terms involving $H_{jt}H_{j(t+1)}$:

$$\begin{aligned}\Pr(H|V) &\propto \exp\Bigg(\langle H, A^h\rangle + \sum_{t=1}^{T-1}\sum_{j=1}^{n_h}\theta_j H_{jt}H_{j(t+1)}\Bigg)\\ &= \prod_{j=1}^{n^h}\exp\Bigg(\langle H_{j*}, A^h_{j*}\rangle + \sum_{t=1}^{T-1}\theta_j H_{jt}H_{j(t+1)}\Bigg)\\ &= \prod_{j=1}^{n^h}\Bigg(\prod_{t=1}^{T}\exp H_{jt}A^h_{jt}\Bigg)\Bigg(\prod_{t=1}^{T-1}\exp\theta_j H_{jt}H_{j(t+1)}\Bigg). \qquad (3)\end{aligned}$$

So given V, the rows of H are independent Markov order-1 sequences with binary states. Therefore the Baum-Welch (or "forward-backward") algorithm can be used to sample from $\Pr(H|V)$ and to determine $\mathbb{E}\left[H|V\right]$, which we will need for training.

It is not hard to show that the gradient of the log-likelihood $\log\Pr(V=\hat{V})$ with respect to the W_δ has the following form, similar to a standard RBM:

$$\nabla_{W_0} = V\Big(\mathbb{E}\big[H' \mid V=\hat{V}\big] - \mathbb{E}\big[H'\big]\Big)$$

$$\text{and, e.g.,}\quad \nabla_{W_1} = V_{*(1:T-1)}\Big(\mathbb{E}\big[H'_{*(2:T)} \mid V=\hat{V}\big] - \mathbb{E}\big[H'_{*(2:T)}\big]\Big).$$

Also, the gradient with respect to θ_j is

$$\nabla_{\theta_j} = \sum_{t=1}^{T-1}\Big(\mathbb{E}\big[H_{jt}H_{j(t+1)} \mid V=\hat{V}\big] - \mathbb{E}\big[H_{jt}H_{j(t+1)}\big]\Big).$$

The positive terms (the conditional expectations) can all be computed exactly by first computing the values $\mathbb{E}[H_{jt}|\hat{V}]$ and $\mathbb{E}[H_{jt}H_{j(t+1)}|\hat{V}]$ with Baum-Welch. To approximate the negative terms, we sample $\tilde{V}$ by running two steps of blocked Gibbs sampling, from $\hat{V}$ to H and back, and then use the conditional expectations given $\tilde{V}$, which is analogous to CD training for an RBM.

3 The Sequential Deep Neural Network

An L-layer SDNN is formed by stacking multiple layers of SRBMs. For $l = 1\ldots L-1$, the hidden layer at level l is a binary matrix $H^l \in \{\pm1\}^{n_l\times T}$ with weight matrices W^l_δ and transition parameters θ^l. We define $V^l \in \mathbb{R}^{n_l\times T}$ for $l = 0\ldots L-1$ to be a matrix of features at layer l. In case $l=0$ (the input), the features are assumed to be real values that are defined by the user in a task-specific way. For the hidden layers ($l = 1\ldots L-1$), we specify $V^l = \mathbb{E}\left[H^l\right]$, where $\Pr(H^l|V^{l-1})$ is defined as in Eq. (3), using the input matrix A^l of the lth layer as defined in Eq. (1).

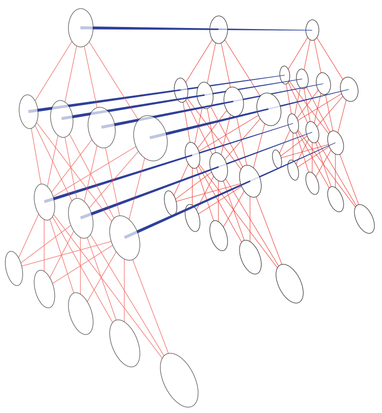


Figure 2: SDNN with $T=3$, $L=3$, $\delta_{\max}=0$.

The output $\{y_1 \dots y_T\}$ is assumed to be a sequence of integer labels, with $y_i \in \{1 \dots n_L\}$. Let $Y \in \mathbb{R}^{n_L \times T}$ be the matrix where $Y_{it} = 1$ if $y_t = i$, and 0 otherwise. We have weight matrices W_δ^L just as with the hidden layers, and the input matrix A^L is formed applying Eq. (1) to the features V^{L-1} of the deepest hidden layer. However now instead of a vector θ of transition parameters, we have a full matrix $U \in \mathbb{R}^{n_L \times n_L}$. The distribution is similar to Eq. 3:

$$\Pr(Y|V^{L-1}) \propto \exp\left(\langle Y, A^L\rangle + \sum_{t=1}^{T-1} Y'_{*t} U Y_{*(t+1)}\right) \tag{4}$$

only here instead of a set of independent binary Markov sequences, $\Pr(Y|V^{L-1})$ defines a single Markov sequence over multinomials with n_L values.

The temporal edges at internal layers of an SDNN can potentially offer distinct advantages in modeling capacity. Consider, for example, a CRF that utilizes features with a fixed temporal span over the input, for example, derived from a static DNN. The only hope to recognize patterns that occur over larger spans is via the temporal integration at the output CRF layer. A SDNN, by contrast, has the ability, starting at $l = 2$, for its hidden units to detect the presence of an arbitrarily long temporal pattern, or even properties of the entire sequence, owing to the earlier layers' Baum-Welch stages that can pass information over an arbitrary temporal extent.

4 Backpropagation algorithm

To fine tune the SDNN parameters, we use a procedure similar to error backpropagation in a static deep network to compute the gradient of the log-likelihood $\ell = \log \Pr(\hat{Y}|V^{L-1})$. The computation has algorithmic properties that favor efficient implementation. Matrix-matrix multiplication is used for both the upward and downward passes, enabling the use of fast matrix multiplication routines. In addition, both passes require Baum-Welch-like procedures that operate independently on rows of the matrix, and that can make efficient use of distributed processing or vectorized arithmetic.

The gradient of the log-likelihood with respect to U is

$$\begin{aligned}\nabla_U \ell &= \sum_{t=1}^{T-1} \hat{Y}_{*t}\hat{Y}'_{*(t+1)} - \mathbb{E}\Big[\sum_{t=1}^{T-1} Y_{*t}Y'_{*(t+1)}\Big] \\ &= \sum_{t=1}^{T-1}\Big(\hat{Y}_{*t}\hat{Y}'_{*(t+1)} - \mathbb{E}\Big[Y_{*t}Y'_{*(t+1)}\Big]\Big)\end{aligned} \tag{5}$$

For W_0^L, we have that

$$\nabla_{W_0^L}\ell = V^{L-1}\nabla_{A^L}\ell = V^{L-1}(\hat{Y}' - \mathbb{E}\big[Y'\big]) = V^{L-1}(D^L)', \tag{6}$$

where $D^L \triangleq \nabla_{A^L}\ell$, and the expressions for W_1^L and W_{-1}^L are similar. (If some other objective function ℓ' is used, just replace D^L with $\nabla_{A^L}\ell'$, and the rest of the derivation is unchanged.)

By the chain rule, the derivative with respect to some value ρ at or below level l is

$$\frac{\partial \ell}{\partial \rho} = \sum_{i=1}^{n_l}\sum_{t=1}^{T} \frac{\partial \ell}{\partial V_{it}^l}\frac{\partial V_{it}^l}{\partial \rho}. \tag{7}$$

Define ϵ^l to be the matrix with $\epsilon_{it}^l = \frac{\partial \ell}{\partial V_{it}^l}$. Then

$$\epsilon^{L-1} = \big[\mathbf{0} \mid W_{-1}^L D_{*(1:T-1)}^L\big] + W_0^L D^L + \big[W_1^L D_{*(2:T)}^L \mid \mathbf{0}\big].$$

Because V^l is the conditional expected value of a log-linear distribution, it follows that

$$\frac{\partial V_{it}^l}{\partial \rho} = \mathbb{E}\Big[H_{it}^l\Big(\mathbb{E}\big[\frac{\partial}{\partial \rho}E(V^{l-1}, H^l)\big] - \mathbb{E}\big[\frac{\partial}{\partial \rho}E(V^{l-1}, H^l) \mid H_{it}^l\big]\Big)\Big]. \tag{8}$$

Now consider $\rho = (W_0^l)_{jk}$. Since $\frac{\partial}{\partial\rho}E(V^{l-1}, H^l) = -\sum_{\tau=1}^{T} V_{j\tau}^{l-1} H_{k\tau}^l$,

$$\begin{aligned}\frac{\partial V_{it}^l}{\partial \rho} &= \sum_{\tau=1}^{T} V_{j\tau}^{l-1}\mathbb{E}\Big[H_{it}^l\Big(\mathbb{E}\big[H_{k\tau}^l \mid H_{it}^l\big] - \mathbb{E}\big[H_{k\tau}^l\big]\Big)\Big] \\ &= \sum_{\tau=1}^{T} V_{j\tau}^{l-1}\Big(\mathbb{E}\big[H_{it}^l H_{k\tau}^l\big] - \mathbb{E}\big[H_{it}^l\big]\big[H_{k\tau}^l\big]\Big) \\ &= \sum_{\tau=1}^{T} V_{j\tau}^{l-1}\operatorname{Cov}(H_{it}^l, H_{k\tau}^l),\end{aligned}$$

which is zero when $k \neq i$ because the rows of H are independent. Plugging this into Eq. (7) only the $i = k$ terms remain, yielding

$$\frac{\partial \ell}{\partial (W_0^l)_{jk}} = \sum_{t=1}^{T}\epsilon_{kt}^l \sum_{\tau=1}^{T} V_{j\tau}^{l-1}\operatorname{Cov}(H_{kt}^l, H_{k\tau}^l) = \sum_{\tau=1}^{T} V_{j\tau}^{l-1}\sum_{t=1}^{T}\epsilon_{kt}^l \operatorname{Cov}(H_{kt}^l, H_{k\tau}^l).$$

Defining $D_{i\tau}^l \triangleq \sum_t \epsilon_{it}^l \operatorname{Cov}(H_{it}^l, H_{i\tau}^l)$ we can write the gradient with respect to the entire matrix W_0^l compactly as $\nabla_{W_0^l}\ell = V^{l-1}(D^l)'$ (exactly as (6)). The gradients with respect to W_δ^l can also be expressed in terms of D^l.

The case of θ_i^l is similar. Since $\frac{\partial}{\partial\theta_i^l}E(V^{l-1}, H^l) = -\sum_{\tau=1}^{T-1} H_{i\tau}^l H_{i(\tau+1)}^l$, from (8)

$$\begin{aligned}\frac{\partial V_{it}^l}{\partial \theta_i^l} &= \mathbb{E}\Big[H_{it}^l\Big(\mathbb{E}\big[\textstyle\sum_{\tau=1}^{T-1} H_{i\tau}^l H_{i(\tau+1)}^l \mid H_{i\tau}^l\big] - \mathbb{E}\big[\textstyle\sum_{\tau=1}^{T-1} H_{i\tau}^l H_{i(\tau+1)}^l\big]\Big)\Big] \\ &= \sum_{\tau=1}^{T-1}\Big(\mathbb{E}\big[H_{it}^l H_{i\tau}^l H_{i(\tau+1)}^l\big] - \mathbb{E}\big[H_{it}^l\big]E\big[H_{i\tau}^l H_{i(\tau+1)}^l\big]\Big) \\ &= \sum_{\tau=1}^{T-1}\operatorname{Cov}(H_{it}^l, H_{i\tau}^l H_{i(\tau+1)}^l),\end{aligned}$$

so

$$\nabla_{\theta_i^l}\ell = \sum_{\tau=1}^{T-1}\sum_{t=1}^{T}\epsilon_{it}^l \operatorname{Cov}(H_{it}^l, H_{i\tau}^l H_{i(\tau+1)}^l) \triangleq \sum_{\tau=1}^{T-1} F_{i\tau}.$$

Finally, by setting $\rho = V_{it}^{l-1}$ in (7), we can also derive

$$\epsilon^{l-1} = \Big[\mathbf{0}\Big|W_{-1}^l D_{*(1:T-1)}^l\Big] + W_0^l D^l + \Big[W_1^l D_{*(2:T)}^l\Big|\mathbf{0}\Big]$$

which allows us to recursively compute the derivatives with respect to lower-level parameters.

At first glance, it may appear that it requires $\mathcal{O}(n_l T^2)$ operations to compute D^l and F^l, since each entry is a sum over T weighted covariances. In fact, it is possible to compute all entries of D^l and F^l in time linear in T with an algorithm that bears a striking resemblance to Baum-Welch. Define

$$\alpha_{i\tau}^l = \sum_{t=1}^{\tau-1}\epsilon_{it}^l\frac{\operatorname{Cov}(H_{it}^l, H_{i\tau}^l)}{\operatorname{Var} H_{i\tau}^l} \quad \text{and} \quad \beta_{i\tau}^l = \sum_{t=\tau+1}^{T}\epsilon_{it}^l\frac{\operatorname{Cov}(H_{i\tau}^l, H_{it}^l)}{\operatorname{Var} H_{i\tau}^l},$$

so that $D_{i\tau}^l = \operatorname{Var}(H_{i\tau}^l)(\alpha_{i\tau}^l + \epsilon_{i\tau}^l + \beta_{i\tau}^l)$. For $F_{i\tau}^l$ we have

$$F_{i\tau}^l = \sum_{t=1}^{\tau-1}\epsilon_{it}^l \operatorname{Cov}(H_{it}^l, H_{i\tau}^l H_{i(\tau+1)}^l) + \sum_{t=\tau}^{\tau+1}\epsilon_{it}^l \operatorname{Cov}(H_{it}^l, H_{i\tau}^l H_{i(\tau+1)}^l) + \sum_{t=\tau+2}^{T}\epsilon_{it}^l \operatorname{Cov}(H_{it}^l, H_{i\tau}^l H_{i(\tau+1)}^l).$$

Considering the first term,[2]

$$\begin{aligned}\sum_{t=1}^{\tau-1}\epsilon_{it}^l \operatorname{Cov}(H_{it}^l, H_{i\tau}^l H_{i(\tau+1)}^l) &= \sum_{t=1}^{\tau-1}\epsilon_{it}^l\frac{\operatorname{Cov}(H_{it}^l, H_{i\tau}^l)\operatorname{Cov}(H_{i\tau}^l, H_{i\tau}^l H_{i(\tau+1)}^l)}{\operatorname{Var} H_{i\tau}^l} \\ &= \operatorname{Cov}(H_{i\tau}^l, H_{i\tau}^l H_{i(\tau+1)}^l)\alpha_{i\tau}^l,\end{aligned}$$

[2]Here and in the recursion for α below, use the identity $\operatorname{Cov}(A, C) = \frac{\operatorname{Cov}(A,B)\operatorname{Cov}(B,C)}{\operatorname{Var} B}$ which holds for ± 1-valued variables where A is independent of C given B.

so that

$$F_{i\tau}^l = \mathrm{Cov}(H_{i\tau}^l, H_{i\tau}^l H_{i(\tau+1)}^l)(\alpha_{i\tau}^l + \epsilon_{i\tau}^l) + \mathrm{Cov}(H_{i\tau}^l H_{i(\tau+1)}^l, H_{i(\tau+1)}^l)(\beta_{i(\tau+1)}^l + \epsilon_{i(\tau+1)}^l).$$

To compute α_{i1}^l, we can set $\alpha_{i1}^l = 0$, and recursively apply

$$\begin{aligned}
\alpha_{i(\tau+1)}^l &= \frac{1}{\mathrm{Var}\, H_{i(\tau+1)}^l} \left(\sum_{t=1}^{\tau-1} \epsilon_{it}^l \,\mathrm{Cov}(H_{it}^l, H_{i(\tau+1)}^l) + \epsilon_{i\tau}^l \,\mathrm{Cov}(H_{i\tau}^l, H_{i(\tau+1)}^l) \right) \\
&= \frac{1}{\mathrm{Var}\, H_{i(\tau+1)}^l} \left(\sum_{t=1}^{\tau-1} \epsilon_{it}^l \frac{\mathrm{Cov}(H_{it}^l, H_{i\tau}^l)\,\mathrm{Cov}(H_{i\tau}^l, H_{i(\tau+1)}^l)}{\mathrm{Var}\, H_{i\tau}} + \epsilon_{i\tau}^l \,\mathrm{Cov}(H_{i\tau}^l, H_{i(\tau+1)}^l) \right) \\
&= \frac{\mathrm{Cov}(H_{i\tau}^l, H_{i(\tau+1)}^l)}{\mathrm{Var}\, H_{i(\tau+1)}^l} \left(\alpha_{i\tau}^l + \epsilon_{i\tau}^l\right) \\
&= \frac{1}{2} \left(\mathbb{E}[H_{i\tau}^l | H_{i(\tau+1)}^l = 1] - \mathbb{E}[H_{i\tau}^l | H_{i(\tau+1)}^l = -1] \right) \left(\alpha_{i\tau}^l + \epsilon_{i\tau}^l\right),
\end{aligned}$$

and symmetrically

$$\beta_{i(\tau-1)}^l = \frac{1}{2} \left(\mathbb{E}[H_{i\tau}^l | H_{i(\tau-1)}^l = 1] - \mathbb{E}[H_{i\tau}^l | H_{i(\tau-1)}^l = -1] \right) \left(\beta_{i\tau}^l + \epsilon_{i\tau}^l\right),$$

yielding a numerically stable dynamic program for computing α^l and β^l (and therefore D^l and F^l) in linear time.

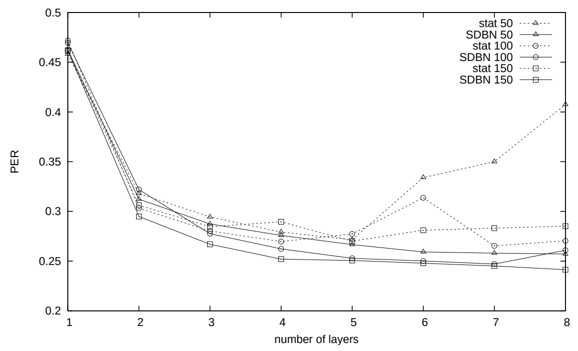
The entire SDNN backpropagation procedure is summarized as follows.

1. (Upward pass.) Given V^0, for $l = 1 \ldots L-1$ compute the inputs A^l from the previous layer's activations V^{l-1}, and then V^l from A^l using Baum-Welch on each row. Compute A^L from V^{L-1} and the label marginals $\mathbb{E}[Y_{it}]$ and $\mathbb{E}[Y_{it} Y_{j(t+1)}]$ with Baum-Welch over the multinomial sequence.
2. (Downward pass.) Construct D^L from the labels $\hat{Y}$ and marginals, and update the weights at the output layer L. Then for $l = L-1 \ldots 1$, backpropagate the error ϵ^l from D^{l+1}, compute D^l and F^l from ϵ^l with the Baum-Welch-like algorithm above, and finally update the weights at layer l.

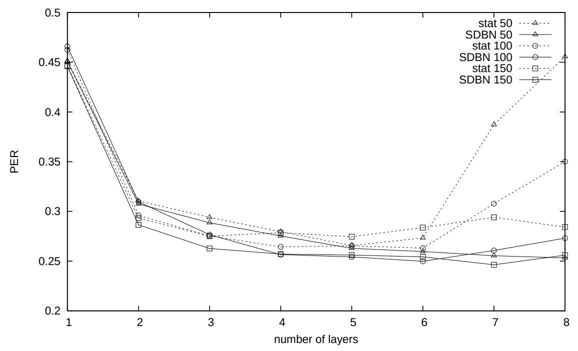
5 Experiments

We tested the SDNN on the TIMIT phone recognition dataset. We use the standard train/test split for phone recognition experiments: removing all SA records from training, and testing on the core test set of 24 speakers. The dataset has 3696 train utterances and 192 test utterances, with an average of 304 frames per utterance. The input features were 12th order MFCCs and energy over 25 ms windows, plus the first-order temporal differences, giving 26 total features per 10 ms frame. The outputs are sequences of the standard 39-phone set of Lee and Hon [1989]. In order to model repeated phones and get some of the modeling power of subphones, we divide each phone into two states, and constrain the model to require traversal through each substate of each phone. The boundaries between subphones are kept latent, that is, we follow the gradient of $\log \sum_{Y_{\text{sub}} \,:\, \text{phones}(Y_{\text{sub}}) = \hat{Y}} \Pr(Y_{\text{sub}})$.

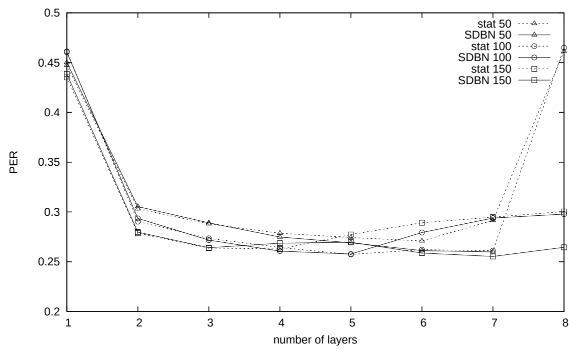
To establish the value of the primary innovation of the SDNN—that it makes use of sequence models at all layers—we compared the complete SDNN to a baseline model that uses a sequence classifier at the top level, but no sequential model at any hidden layer, exactly as if θ^l were constrained to be zero for $l < L$. We compare the models over a range of configurations: we vary the model depth from one layer to eight, the half-width δ_{max} of the input window (at all layers) from one to four, and the number of hidden units per frame (in each layer) from 50 to 150. Each stage of training (that is, pre-training each layer with CD and also joint training of all parameters with BP) continued until the training criterion (squared reconstruction error for CD, log-likelihood for BP) failed to improve over five epochs, at which point the learning rate was annealed linearly to zero over another five epochs. Weight decay is applied after each update, and the amount of decay is scaled proportionally with T. The initial learning rates, weight decays and momentum parameters were estimated using random



(a) $\delta_{\text{max}} = 1$



(b) $\delta_{\text{max}} = 2$



(c) $\delta_{\text{max}} = 4$

Figure 3: Average test set PER of the SDNN and baseline model over a range of number of layers, n_h and input window width δ_{max}.

search to maximize phone error rate (PER) on a randomly selected 10% the training set, which was added back before training the final model for test results.

The results of the experiment are summarized in Figure 3. Comparing the complete SDNN to the baseline model, it is apparent that using full sequence information at all layers is beneficial across nearly all configurations, and the gains are more significant as the number of hidden layers increases. Interestingly, the results also indicate that the use of temporal hidden units may make a very wide input window unnecessary: our best results are obtained with $\delta_{\text{max}} = 1$, whereas δ_{max} from 5 to as high as 11 is more common with MLPs or DNNs for phone recognition [Dahl et al., 2010, Hinton et al., 2012].

Our best performing configuration (150 units/frame, 8 layers, $\delta_{\text{max}} = 1$) achieved a PER of 24.2, which surpasses many systems that are highly tailored to the phone-recognition task [Keshet et al., 2006, Crammer, 2006, Cheng et al., 2009, Morris and Fosler-Lussier, 2008, Keshet et al., 2011, Sung and Jurafsky, 2009, Sha and Saul, 2007]. Other recent developments using deep networks have improved somewhat on those scores [Dahl et al., 2010, Hinton et al., 2012, Plahl et al., 2012, Tóth, 2013], and in future work it would be interesting to try combining some of those successful techniques with the SDNN model, in particular, the use of triphone states, dropout training and larger numbers of hidden units per frame. It is significant that the SDNN performs so well with only 150 hidden units per frame, over an order of magnitude fewer than state-of-the-art neural systems (1500 in Dahl et al. [2010], 2000 in Tóth [2013] and 4000 in Hinton et al. [2012]). However our goal in the present work is not to build a state-of-the-art system but to demonstrate that the use of temporal connections between hidden units improves upon a competitive baseline model that does not.

References

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.

H. Hermansky and S. Sharma. Temporal patterns (TRAPS) in ASR of noisy speech. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 289–292. IEEE, 2002. ISBN 0780350413.

A. Gunawardana, M. Mahajan, A. Acero, and J. C. Platt. Hidden conditional random fields for phone classification. In *Interspeech*, 2005.

Y.-H. Sung and D. Jurafsky. Hidden conditional random fields for phone recognition. In *Automatic Speech Recognition and Understanding*, 2009.

D. Yu, S. Wang, and L. Deng. Sequential labeling using deep-structured conditional random fields. *IEEE Journal of Selected Topics in Signal Processing*, 4(6), 2010.

G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18, 2006.

Christian Plahl, Tara N Sainath, Bhuvana Ramabhadran, and David Nahamoo. Improved pre-training of deep belief networks using sparse encoding symmetric machines. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4165–4168. IEEE, 2012.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

A. Mohamed, G. Dahl, and G. Hinton. Deep belief networks for phone recognition. In *Advances in Neural Information Processing Systems 22 (Workshops)*, 2009.

A. Mohamed, D. Yu, and L. Deng. Investigation of full-sequence training of deep belief networks. In *Interspeech*, 2010.

Karel Veselỳ, Arnab Ghoshal, Lukáš Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. In *Interspeech*, Lyon, France, 2013.

Galen Andrew and Jeff Bilmes. Sequential deep belief networks. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4265–4268. IEEE, 2012.

K.F. Lee and H.-W. Hon. Speaker-independent phone recognition using hidden markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37, 1989.

G. E. Dahl, M. Ranzato, A. Mohamed, and G. Hinton. Phone recognition with the mean-covariance restricted boltzmann machine. In *Advances in Neural Information Processing Systems 23*, 2010.

J. Keshet, S. Shalev-Shwartz, S. Bengio, Y. Singer, and D. Chazan. Discriminative kernel-based phoneme sequence recognition. In *Interspeech*, 2006.

K. Crammer. Efficient online learning with individual learning-rates for phoneme sequence recognition. *Journal of Machine Learning Research*, 7, 2006.

C.-C. Cheng, F. Sha, and L. K. Saul. A fast online algorithm for large margin training of continuous-density hidden markov models. In *Interspeech*, 2009.

J. Morris and E. Fosler-Lussier. Conditional random fields for integrating local discriminative classifiers. *IEEE Trans. on Acoustics, Speech, and Language Processing*, 16(3), 2008.

J. Keshet, D. McAllester, and T. Hazan. Pac-bayesian approach for minimization of phoneme error rate. In *International Conference on Acoustics Speech and Signal Processing*, 2011.

F. Sha and L. K. Saul. Comparison of large margin training to other discriminative methods for phonetic recognition by hidden markov models. In *International Conference on Acoustics Speech and Signal Processing*, 2007.

László Tóth. Convolutional deep rectifier neural nets for phone recognition. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2013.