A Primal-Dual Method for Training Recurrent Neural Networks Constrained by the Echo-State Property

Jianshu Chen
Department of Electrical Engineering
University of California
Los Angeles, CA 90034, USA
cjs09@ucla.edu

Li Deng
Machine Learning Group
Microsoft Research
Redmond, WA 98052, USA
deng@microsoft.com

Abstract

We present an architecture of a recurrent neural network (RNN) with a fully-connected deep neural network (DNN) as its feature extractor. The RNN is equipped with both causal temporal prediction and non-causal look-ahead, via auto-regression (AR) and moving-average (MA), respectively. The focus of this paper is a primal-dual training method that formulates the learning of the RNN as a formal optimization problem with an inequality constraint that provides a sufficient condition for the stability of the network dynamics. Experimental results demonstrate the effectiveness of this new method, which achieves 18.86% phone recognition error on the TIMIT benchmark for the core test set. The result approaches the best result of 17.7%, which was obtained by using RNN with long short-term memory (LSTM). The results also show that the proposed primal-dual training method produces lower recognition errors than the popular RNN methods developed earlier based on the carefully tuned threshold parameter that heuristically prevents the gradient from exploding.

1 Introduction

Considerable impact in speech recognition has been created in recent years using fully-connected deep neural networks (DNN) that drastically cut errors in large vocabulary speech recognition (Yu et al., 2010; Dahl et al., 2011; Seide et al., 2011; Dahl et al., 2012; Kingsbury et al., 2012; Hinton et al., 2012; Deng et al., 2013b,c; Yu et al., 2013; Dahl et al., 2013; Sainath et al., 2013). However, the well-known problem of the previous state-of-the-art approach based on Gaussian Mixture Model (GMM)-HMMs has not been addressed by the DNNs in a principled way: missing temporal correlation structure that is prevalent in the speech sequence data. Recurrent neural networks (RNN) have shown their potential to address this problem (Robinson, 1994; Graves et al., 2013), but the difficulty of learning RNNs due to vanishing or exploding gradients (Pascanu et al., 2013) or the complexity of the LSTM (long short-term memory) structure in RNNs (Graves et al., 2013) have so far slowed down the research progress in using RNNs to improve speech recognition and other sequence processing tasks.

In this paper, we propose an architecture of RNNs for supervised learning, where the input sequence to the RNN is computed by an independent feature extractor using a fully-connected DNN receiving its input from raw data sequences. We have formulated both autoregressive (AR) and autoregressive and moving-average (ARMA) versions of this RNN. A new learning method is developed in this work, which is successfully applied to both AR and ARMA versions of the RNN. The new method frames the RNN learning problem as a constrained optimization one, where cross entropy is maximized subject to having the infinity norm of the recurrent matrix of the RNN to be less than

a fixed value that provides a sufficient condition for the stability of RNN dynamics. A primal-dual technique is devised to solve the resulting constrained optimization problem in a principled way. Experimental results on phone recognition demonstrate: 1) the primal-dual technique is effective in learning RNNs, with satisfactory performance on the TIMIT benchmark; 2) The ARMA version of the RNN produces higher recognition accuracy than the traditional AR version; 3) The use of a DNN to compute high-level features of speech data to feed into the RNN gives much higher accuracy than without using the DNN; and 4) The accuracy drops progressively as the DNN features are extracted from higher to lower hidden layers of the DNN.

2 Related Work

The use of recurrent or temporally predictive forms of neural networks for speech recognition dated back to early 1990's (Robinson, 1994; Deng et al., 1994), with relatively low accuracy. Since deep learning became popular in recent years, much more research has been devoted to the RNN (Graves et al., 2006; Graves, 2012; Maas et al., 2012; Mikolov, 2012; Vinyals et al., 2012; Graves et al., 2013). Most work on RNNs made use of the method of Back Propagation Through Time (BPTT) to train the RNNs, and empirical tricks need to be exploited in order to make the training effective. The most notable trick is to truncate gradients when they become too large (Mikolov et al., 2011; Pascanu et al., 2013; Bengio et al., 2013). Likewise, another empirical method, based on a regularization term that represents a "preference for parameter values," was introduced to handle the gradient vanishing problem (Pascanu et al., 2013).

The method we propose for training RNN in this paper is based on optimization principles, capitalizing on the cause of difficulties in BPTT analyzed in (Pascanu et al., 2013). Rather than using empirical ways to scale up or scale down the gradients, we formulate the RNN learning problem as an optimization one with inequality constraints and we call for a natural, primal-dual method to solve the constrained optimization problem in a principled way (Polyak, 1987). Our work also differs from the earlier work reported in (Sutskever, 2013; Martens & Sutskever, 2011), which adopted a special way of initialization using echo state networks (Jaeger, 2001a) when carrying out the standard BPTT procedure, without using a formal constrained optimization framework.

This work is originally motivated by the echo state network (Jaeger, 2001a), which carefully hand-designed the recurrent weights making use the the echo-state property. The weights are then fixed and not learned, due to the difficulty in learning them. Instead, in this work, we devise a method of learning these recurrent weights by a formal optimization framework which naturally incorporates the constraint derived from the echo-state property. The echo-state property proposed in (Jaeger, 2001a) provides a sufficient condition for avoiding the exploding gradient problem. Although such a sufficient constraint looks restrictive, it allows the RNN to be trained in a relatively easy and principled way, and the trained RNN achieves satisfactory performance on the TIMIT benchmark.

As another main contribution of this paper, we build a "deep" version of the RNN by using an independent DNN to extract high-level features from the raw speech data, capitalizing on the well established evidence for the power of DNNs in carrying out automatic feature extraction (Hinton et al., 2012; Dahl et al., 2011; LeCun, 2012). Having the DNN as the feature extractor and the RNN as the classifier trained separately helps reduce overfitting in the training. This strategy of building a deep version of RNNs contrasts with other strategies in the literature, where the same RNN model is stacking up on top of each other and all layers of RNNs are trained jointly. Aside from using weight noise for regularization, the overfitting problem caused by joint training of many RNNs has been circumvented mainly by exploiting elaborate structures such as LSTM (Graves et al., 2013), making the complexity of the overall model higher and the results more difficult to reproduce than our simpler approach to building the deep version of the RNN reported in this paper.

Finally, our method exploits the DNN features in a very different way than the "tandem" architecture discussed in (Tüske et al., 2012), where posterior outputs at the top layer of the DNN is concatenated with the acoustic features as new inputs to an HMM system. We take the hidden layer of the DNN as the features, which are shown experimentally in our work to be much better than the top-layer posterior outputs as in the tandem method. Further, rather than using the GMM-HMM as a separate sequence classifier in (Tüske et al., 2012), we use the RNN as the sequence classifier.

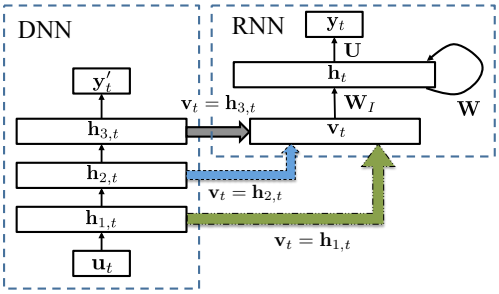

Figure 1: Architecture of the recurrent neural network with DNN features, where one of the hidden layers of the DNN will be used as input for the RNN. Arrows of different colors from the DNN to the RNN indicate alternative ways of providing the inputs to RNN.

3 The Recurrent Neural Network

3.1 Basic architecture

We consider a deep-RNN shown in Fig. 1. $\mathbf{y}'_t$ denotes the output of the DNN that is trained over randomly permuted input sequence, and $\mathbf{y}_t$ denotes the output of RNN corresponding to the input sequence of the original order. The network consists of two parts: (i) a lower-level DNN, and (ii) an upper-level RNN. The major task of the lower DNN is to extract useful features from the raw data and feed them to the upper RNN component, which replaces HMM as a sequence classifier. The RNN in the upper level captures the temporal contextual information in its hidden states and generates the outputs that predict the labels of the input data in the supervised sequential prediction task such as recognizing phones or words embedded in a sentence.

3.2 Mathematical formulation of the RNN

In this section, we focus on the RNN part of the deep network, given the DNN outputs that provide input vectors $\mathbf{v}_t$'s to the RNN. A standard RNN is described by the following equations:

$$\mathbf{h}_t = \mathbf{f}\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{W}_I\mathbf{v}_t + \mathbf{b}\right) \tag{1}$$

$$\mathbf{y}_t = \mathbf{g}\left(\mathbf{U}\mathbf{h}_t\right) \tag{2}$$

where $\mathbf{h}_t \in \mathbb{R}^N$, $\mathbf{v}_t \in \mathbb{R}^{N_I}$ and $\mathbf{y}_t \in \mathbb{R}^{N_o}$ are vectors at discrete time t that represent the hidden states, the inputs, and the outputs, respectively, and the matrices $\mathbf{W} \in \mathbb{R}^{N\times N}$, $\mathbf{W}_I \in \mathbb{R}^{N\times N_I}$, $\mathbf{U} \in \mathbb{R}^{N_o\times N}$, and vector $\mathbf{b} \in \mathbb{R}^N$ collect the recurrent weights, input weights, the output weights and the bias. The function $\mathbf{f}(\mathbf{x})$ represents the nonlinearities that are applied to each entry of the vector $\mathbf{x}$, i.e., the nonlinear operation that is applied to the vector component-by-component:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f(x_1) & \cdots & f(x_N) \end{bmatrix}^T$$

where x_k denotes the k-entry of the vector $\mathbf{x}$, and each $f(x_k)$ can be sigmoid, $\tanh$ or rectified. (We will use sigmoid nonlinearity through out this work as an example.) And $\mathbf{g}(\mathbf{x})$ represents the operation applied at the output units. Typical choices are linear function $\mathbf{g}(\mathbf{x}) = \mathbf{x}$ or soft-max operation.

While the standard RNN model described by (1) updates the hidden states $\mathbf{h}_t$ from its summary of the history $\mathbf{h}_{t-1}$ and the current input $\mathbf{v}_t$, which we call the autoregressive (AR) version, we can have a more general ARMA version described by

$$\mathbf{h}_t = \mathbf{f}\left(\mathbf{W}\mathbf{h}_{t-1} + \sum_{\tau=-\Delta_1}^{\Delta_2} \mathbf{W}_{I,\tau}\mathbf{v}_{t-\tau} + \mathbf{b}\right) \tag{3}$$

where Δ_1 and Δ_2 denote the number of inputs that the network looks forward and backwards. If $\Delta_1 = 0$, then it only looks backwards into the past history and if $\Delta_2 = 0$, it only looks into future.

The models described by (1) and (3) are parallel to the vector AR and ARMA models, respectively, in time series analysis except that the classical AR and ARMA models are linear:

$$[\text{AR}]: \ \mathbf{h}_t = \mathbf{W}\mathbf{h}_{t-1} + \mathbf{W}_I\mathbf{v}_t \tag{4}$$

$$[\text{ARMA}]: \ \mathbf{h}_t = \mathbf{W}\mathbf{h}_{t-1} + \sum_{\tau=-\Delta_1}^{\Delta_2} \mathbf{W}_{I,\tau}\mathbf{v}_{t-\tau} \tag{5}$$

In the context of ARMA version of the RNN, $\mathbf{W}\mathbf{h}_{t-1}$ is called AR part, and the term $\sum_{\tau=-\Delta_1}^{\Delta_2} \mathbf{W}_{I,\tau}\mathbf{v}_{t-\tau}$ is called MA part. Depending on the nature of the time sequence, some tasks are easier to be modeled by AR, and some others by MA, or jointly ARMA. Just as AR model is a special case of ARMA model ($\Delta_1 = \Delta_2 = 0$), model (1) is a special case of (3). Due to its generality, the ARMA version of the RNN (3) is expected to be more powerful than the AR version (1). Indeed, our experimental results presented in Section 5 have confirmed this expectation.

A key observation we make is that in order to develop a unified learning method, (3) can be converted back to the form of (1) by defining some extra augmented variables. Let $\overline{\mathbf{v}}_t$ and $\overline{\mathbf{W}}_I$ be defined as

$$\overline{\mathbf{v}}_t \triangleq \begin{bmatrix}\mathbf{v}_{t-\Delta_2}^T & \cdots & \mathbf{v}_{t+\Delta_1}^T\end{bmatrix}^T \tag{6}$$

$$\overline{\mathbf{W}}_I \triangleq [\mathbf{W}_{I,-\Delta_2} \quad \cdots \quad \mathbf{W}_{I,\Delta_1}] \tag{7}$$

Then, (3) can be written in the following equivalent form:

$$\mathbf{h}_t = \mathbf{f}\left(\mathbf{W}\mathbf{h}_{t-1} + \overline{\mathbf{W}}_I\overline{\mathbf{v}}_t + \mathbf{b}\right) \tag{8}$$

In other words, the ARMA version of the RNN model in (3) can be implemented in an equivalent manner by having a context window that slides through several input samples and deliver them as an augmented input sample. From now on, we will proceed with the simple AR model (1)–(2) since the general ARMA version can be reduced to the AR version by the above conversion and hence the learning method will stay unchanged from the AR version to the ARMA version. Note that another approach to letting the model look into future is the bi-directional recurrent neural network(Graves et al., 2013), which uses information from future by letting its hidden layers depend on the hidden states at future. On the other hand, the ARMA is made to depend on future by letting its input include future. So these two methods uses information from future in two different ways and our ARMA method is relatively easier to implement and also effectively capture the future information. Furthermore, by incorporating past and future information into the inputs, the ARMA-RNN also provides a relative easy way of enhancing the amount of memory in the RNN.

3.3 Learning the RNN and its difficulties

The standard BPTT method for learning RNNs "unfolds" the network in time and propagates error signals backwards through time. Let $J(\Theta)$ denote the cost function that measures how good the RNN predicts the target, where $\Theta \triangleq \{\mathbf{W}, \mathbf{W}_I, \mathbf{U}, \mathbf{b}\}$ is the set of all the parameters to be trained. In general, the cost can be written as an average of costs over different time instants:

$$J(\Theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\mathbf{y}_t, \mathbf{d}_t) \tag{9}$$

where $\mathbf{d}_t$ denotes the desired signal (also called target) at time t, and $J_t(\mathbf{y}_t, \mathbf{d}_t)$ characterizes the cost at time t. The cost $J_t(\mathbf{y}_t, \mathbf{d}_t)$ depends on the model parameters Θ through $\mathbf{y}_t$, which, as shown in (1)–(2), further depends on $\{\mathbf{W}, \mathbf{W}_I, \mathbf{U}, \mathbf{b}\}$. Typical choices of $J_t(\mathbf{y}_t, \mathbf{d}_t)$ includes squared-error and cross entropy, etc. The gradient formula for $J(\Theta)$ with respect to $\Theta \triangleq \{\mathbf{W}, \mathbf{W}_I, \mathbf{U}, \mathbf{b}\}$ can be computed by back propagation through time (BPTT), which are given in sections B–C of the supplementary material.

It is well known that training RNNs is difficult because of the exploding and vanishing gradient problems (Pascanu et al., 2013). Let

$$\gamma \triangleq \|\mathbf{D}_f\| \tag{10}$$

where $\mathbf{D}_f$ is a diagonal matrix constructed from the element-wise derivative of $\mathbf{f}(\mathbf{x})$:

$$\mathbf{D}_f \triangleq \text{diag}\{\mathbf{f}'(\mathbf{p}_t)\} \tag{11}$$

and $\|\mathbf{X}\|$ denotes the $2-$norm (the largest singular value) of the matrix X. For sigmoid function $\mathbf{f}(\mathbf{x})$, $\gamma = 1/4$, and for $\tanh$, $\gamma = 1$. The argument in (Pascanu et al., 2013) pointed out that a sufficient condition for vanishing gradient problem to occur is

$$\|\mathbf{W}\| = \|\mathbf{W}^T\| < \frac{1}{\gamma} \tag{12}$$

and a necessary condition for exploding gradient to occur is that

$$\|\mathbf{W}\| = \|\mathbf{W}^T\| > \frac{1}{\gamma} \tag{13}$$

Therefore, the properties of the recurrent matrix $\mathbf{W}$ is essential for the performance of the RNN. In previous work (Pascanu et al., 2013; Mikolov et al., 2011), the way to solve the exploding gradient problem is to empirically clip the gradient if the norm of the gradient exceeds a certain threshold, and the way to avoid the vanishing gradient is also empirical — either to add a regularization term to push up the gradient or exploit the information about the curvature of the objective function (Maas et al., 2012). Below, we describe an alternative primal-dual approach for training RNN, which is based on the echo-state property, a sufficient condition for avoiding the exploding gradient problem.

4 A New Algorithm for Learning the RNN

In this section, we first discuss an essential property for the model of the kind in (1) — the echo-state property, and provide a sufficient condition for it. Then, we formulate the problem of training the RNN that preserves the echo-state property as a constrained optimization problem. Finally, we develop a primal-dual method to solve the problem.

4.1 The echo-state property

We now show that conditions of the form (12)–(13) are closely related to whether the model (1)–(2) satisfies the echo-state property (Jaeger, 2002), which states that "if the network has been run for a very long time [from minus infinity time in the definition], the current network state is uniquely determined by the history of the input and the (teacher-forced) output". It is also shown in (Jaeger, 2001b) that this echo-state property is equivalent to the "state contracting" property. Since we do not consider the case with feedback from output in the model (1), we here describe the "state contracting" property slightly different from (Jaeger, 2001b):

Definition 1 (State contracting)**.** *The network is state contracting if for all right-infinite input sequences $\{\mathbf{v}_t\}$, where $t = 0, 1, 2, \ldots$, there exists a null sequence $(\epsilon_t)_{t\geq 0}$ such that for all starting states $\mathbf{h}_0$ and $\mathbf{h}'_0$ and for all $t > 0$ it holds that $\|\mathbf{h}_t - \mathbf{h}'_t\| < \epsilon_t$, where $\mathbf{h}_t$ [resp. $\mathbf{h}'_t$] is the hidden state vector at time t obtained when the network is driven by $\mathbf{v}_t$ up to time t after having been stated in $\mathbf{v}_0$, [resp. in $\mathbf{h}'_0$].*

It is shown in (Jaeger, 2001a) that a sufficient condition for the *non-existence*, i.e., a necessary condition for the *existence*, of the echo-state property is that the spectral radius of the recurrent matrix $\mathbf{W}$ is greater than one (resp. four) when $\tanh$ (resp. sigmoid) units are used. And it is further conjectured that for a weight matrix $\mathbf{W}$ that is randomly generated by sampling the weights over uniform distribution over $[-1, 1]$ and normalized such that $\rho(\mathbf{W}) = 1 - \delta$ would satisfy the echo-state property with high probability, where δ is a small positive number.

In echo-state networks, the reservoir or the recurrent weights are randomly generated and normalized according to the rule above and will be fixed over time in the training. The input weights are fixed as well. Instead, in this paper, *we learn the recurrent weights together with the input and output weights subject to the constraint that the network satisfies the echo-state property.* To this end, we propose the following sufficient condition for the echo-state property, which will be shown to be easily handled in the training procedure. The proof of the following proposition can be found in Appendix A.

Proposition 1 (Echo-state: a sufficient condition)**.** *Let $\gamma \triangleq \max_x |f'(x)|$. The network model* (1) *satisfies the echo-state property if*

$$\|\mathbf{W}\|_\infty < \frac{1}{\gamma} \tag{14}$$

where $\|\mathbf{W}\|_\infty$ *denote the* ∞*-norm of the matrix* $\mathbf{W}$ *(maximum absolute row sum), and* $\gamma = 1$ *for* $\tanh$ *units, and* $\gamma = 1/4$ *for sigmoid units.*

The echo state property is typically assumed to capture a short-term memory in the data. However, it is reasonable for many applications, such as speech recognition, using RNNs since each phone typically lasts for fewer than 60 frames, requiring relatively short memory. Suppose we have two RNNs that run on different input sequences up to some time t so that they have different hidden states at time t. Afterwards, if the input acoustic sequences to the two RNNs become identical, the echo state property requires that the hidden states of these two RNNs be close to each other in order to produce the same predicted labels. Another important consequence of condition (14) is that it also provides a sufficient condition to avoid the exploding gradient problem in a more principled manner. This can be shown by following the same argument as (Pascanu et al., 2013) except that the 2-norm is replaced by ∞-norm. Thus, if the condition (14) can be enforced in the training process, there is no need to clip the gradient. We will show in next section that the ∞-norm constraint on $\mathbf{W}$ is convenient to handle in optimization. Therefore, our proposed method below provides a relatively easy way to train RNN in a principled manner, especially on tasks that requires short memory. Learning RNN under refined characterization of its dynamics, such as (Manjunath & Jaeger, 2013), is left as future work.

4.2 Formal formulation of the learning problem

Based on the previous discussion, we can formulate the problem of training the RNN that preserves the echo-state property as the following constrained optimization problem:

$$\min_{\Theta} \quad J(\Theta) = J(\mathbf{W}, \mathbf{W}_I, \mathbf{U}, \mathbf{b}) \tag{15}$$

$$\text{s.t.} \quad \|\mathbf{W}\|_\infty \leq \frac{1}{\gamma} \tag{16}$$

In other words, we need to find the set of model parameters that best predict the target while preserving the echo-state property. Recall that $\|\mathbf{W}\|_\infty$ is defined as the maximum absolute row sum. Therefore, the above optimization problem is equivalent to the following constrained optimization problem (since $\max\{x_1, \ldots, x_N\} \leq 1/\gamma$ is equivalent to $x_i \leq 1/\gamma,\ i = 1, \ldots, N$):

$$\min_{\Theta} \quad J(\Theta) = J(\mathbf{W}, \mathbf{W}_I, \mathbf{U}, \mathbf{b}) \tag{17}$$

$$\text{s.t.} \quad \sum_{j=1}^{N} |W_{ij}| \leq \frac{1}{\gamma}, \quad i = 1, \ldots, N \tag{18}$$

where W_{ij} denotes the (i, j)-th entry of the matrix $\mathbf{W}$. Next, we will proceed to derive the learning algorithm that can achieve this objective.

4.3 Primal-dual method for optimizing RNN parameters

We solve the constrained optimization problem (17)–(18) by primal-dual method. The Lagrangian of the problem can be written as

$$\mathcal{L}(\Theta, \boldsymbol{\lambda}) = J(\mathbf{W}, \mathbf{W}_I, \mathbf{U}, \mathbf{b}) + \sum_{i=1}^{N} \lambda_i \left(\sum_{j=1}^{N} |W_{ij}| - \frac{1}{\gamma} \right) \tag{19}$$

where λ_i denotes the ith entry of the Lagrange vector $\boldsymbol{\lambda}$ (or dual variable) and is required to be non-negative. Let the dual function $q(\boldsymbol{\lambda})$ be defined as the following *unconstrained* optimization problem:

$$q(\boldsymbol{\lambda}) = \min_{\Theta} \mathcal{L}(\Theta, \boldsymbol{\lambda}) \tag{20}$$

It is shown that the dual function $q(\boldsymbol{\lambda})$ obtained from the above unconstrained optimization problem is always concave, even when the original cost $J(\Theta)$ is not convex (Boyd & Vandenberghe, 2004). And the dual function is always a lower bound of the original constrained optimization problem (17)–(18): $q(\boldsymbol{\lambda}) \leq J(\Theta^\star)$. Maximizing $q(\boldsymbol{\lambda})$ subject to the constraint $\lambda_i \geq 0$, $i = 1, \ldots, N$ will

be the best lower bound that can be obtained from the dual function (Boyd & Vandenberghe, 2004). This new problem is called the dual problem of the original optimization problem (17)–(18):

$$\max_{\boldsymbol{\lambda}} \quad q(\boldsymbol{\lambda}) \tag{21}$$

$$\text{s.t.} \quad \lambda_i \geq 0, \quad i = 1, \ldots, N \tag{22}$$

which is a convex optimization problem since we are maximizing a concave objective with linear inequality constraints. After solving $\boldsymbol{\lambda}^\star$ from (21)–(22), we can substitute the corresponding $\boldsymbol{\lambda}^\star$ into the Lagrangian (19) and then solve the correponding set of parameters $\Theta^o = \{\mathbf{W}^o, \mathbf{W}_I^o, \mathbf{U}^o, \mathbf{b}^o\}$ that minimizes $\mathcal{L}(\Theta, \boldsymbol{\lambda})$ for this given $\boldsymbol{\lambda}^\star$:

$$\Theta^o = \arg\min_{\Theta} \mathcal{L}(\Theta, \boldsymbol{\lambda}^\star) \tag{23}$$

Then, the obtained $\Theta^o = \{\mathbf{W}^o, \mathbf{W}_I^o, \mathbf{U}^o, \mathbf{b}^o\}$ will be an approximation to the optimal solutions. In convex optimization problems, this approximate solution will be the same global optimal solution under some mild conditions (Boyd & Vandenberghe, 2004). This property is called strong duality. However, in general non-convex problems, it will not be the exact solution. But since finding the globally optimal solution to the original problem (17)–(18) is not realistic, it would be satisfactory if it can provide a good approximation.

Back to the problem (21)–(22), we are indeed solving the following problem

$$\max_{\boldsymbol{\lambda} \succeq \mathbf{0}} \min_{\Theta} \mathcal{L}(\Theta, \boldsymbol{\lambda}) \tag{24}$$

where the notation $\boldsymbol{\lambda} \succeq \mathbf{0}$ means that each entry of the vector $\boldsymbol{\lambda}$ is greater than or equal to zero. *Now, to solve the problem, what we need to do is to minimize the Lagrangian $\mathcal{L}(\Theta, \boldsymbol{\lambda})$ with respect to Θ, and in the mean time, maximize the dual variable $\boldsymbol{\lambda}$ subjected to the constraint that $\boldsymbol{\lambda} \succeq \mathbf{0}$.* Therefore, as we will see soon, updating the RNN parameters consists of two steps: *primal update* (minimization of $\mathcal{L}$ with respect to Θ) and *dual update* (maximization of $\mathcal{L}$ with respect to $\boldsymbol{\lambda}$).

First, we provide the primal update rule. To minimize the Lagrangian $\mathcal{L}(\Theta, \boldsymbol{\lambda})$ with respect to Θ, we may apply gradient descent to $\mathcal{L}(\Theta, \boldsymbol{\lambda})$ with respect to Θ. However, note that $\mathcal{L}(\Theta, \boldsymbol{\lambda})$ consists of two parts: $J(\Theta)$ that measures the prediction quality, and the part that penalizes the violation of the constraint (18). Indeed, the second part is a sum of many ℓ_1 regularization terms on the rows of the matrix $\mathbf{W}$:

$$\sum_{j=1}^{N} |W_{ij}| = \|\mathbf{w}_i\|_1 \tag{25}$$

where $\mathbf{w}_i$ denotes the ith row vector of the matrix $\mathbf{W}$. With such observations, the Lagrangian in (19) can be written in the following equivalent form:

$$\mathcal{L}(\Theta, \boldsymbol{\lambda}) = J(\mathbf{W}, \mathbf{W}_I, \mathbf{U}, \mathbf{b}) + \sum_{i=1}^{N} \lambda_i \left(\|\mathbf{w}_i\|_1 - \frac{1}{\gamma} \right) \tag{26}$$

To Minimize $\mathcal{L}(\Theta, \boldsymbol{\lambda})$ of the above structure with respect to $\Theta = \{\mathbf{W}, \mathbf{W}_I, \mathbf{U}, \mathbf{b}\}$, we can apply an argument similar to the one made in (Beck & Teboulle, 2009) to derive the following iterative soft-thresholding algorithm for the primal update of $\mathbf{W}$:

$$\mathbf{W}_k = \mathcal{T}_{\boldsymbol{\lambda}\mu_k} \left\{ \mathbf{W}_{k-1} - \mu_k \frac{\partial J(\mathbf{W}_{k-1}, \mathbf{W}_{I,k-1}, \mathbf{U}_{k-1}, \mathbf{b}_{k-1})}{\partial \mathbf{W}} \right\} \tag{27}$$

where $\mathcal{T}_{\boldsymbol{\lambda}\mu_k}(\mathbf{X})$ denote a component-wise shrinkage (soft-thresholding) operator on a matrix $\mathbf{X}$, defined as

$$[\mathcal{T}_{\boldsymbol{\lambda}\mu_k}(\mathbf{X})]_{ij} = \begin{cases} X_{ij} - \lambda_i \mu_k & X_{ij} \geq \lambda_i \mu_k \\ X_{ij} + \lambda_i \mu_k & X_{ij} \leq -\lambda_i \mu_k \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

Note that the primal update for $\mathbf{W}$ is implemented by a standard (stochastic) gradient descent followed by a shrinkage operator. On the other hand, the primal updates for $\mathbf{W}_I$, $\mathbf{U}$ and $\mathbf{b}$ follow the standard (stochastic) gradient descent rule:

$$\mathbf{W}_{I,k} = \mathbf{W}_{I,k-1} - \mu_k \frac{\partial J(\mathbf{W}_{k-1}, \mathbf{W}_{I,k-1}, \mathbf{U}_{k-1}, \mathbf{b}_{k-1})}{\partial \mathbf{W}_I} \tag{29}$$

Table 1: Phone recognition error (percent) on the TIMIT core test set using DNN-top feature sequences as the input to the RNNs. The results are shown as a function of two hyperparameters: the size of the RNN's hidden layer and the moving-average order of the RNN.

HiddenSz	Order=0	Order=2	Order =4	Order=6	Order=8	Order=10	Order=12
100	19.85	19.83	19.80	19.65	19.50	19.44	19.42
200	19.86	19.72	19.65	19.60	19.45	19.35	19.31
300	20.02	19.72	19.60	19.56	19.40	19.23	19.16
500	20.00	19.73	19.56	19.44	19.34	19.06	18.91
1000	20.44	19.83	19.60	19.49	19.24	19.10	18.98
2000	20.70	20.34	20.10	19.65	19.45	19.30	19.12

Table 2: Phone recognition error as a function of the clipping threshold ((Mikolov et al., 2011); (Pascanu et al., 2013)). The lowest error 19.05% around threshold 1.0 is nevertheless higher than 18.91% obtained by the new primal-dual method without tuning parameters.

Threshold	1	2	0.5	9	1.0	1.1	1.5	2	10
Phone error (%)	19.65	19.50	19.25	19.10	19.05	19.08	19.15	19.54	20.5

$$\mathbf{U}_k = \mathbf{U}_{k-1} - \mu_k \frac{\partial J(\mathbf{W}_{k-1}, \mathbf{W}_{I,k-1}, \mathbf{U}_{k-1}, \mathbf{b}_{k-1})}{\partial \mathbf{U}} \tag{30}$$

$$\mathbf{b}_k = \mathbf{b}_{k-1} - \mu_k \frac{\partial J(\mathbf{W}_{k-1}, \mathbf{W}_{I,k-1}, \mathbf{U}_{k-1}, \mathbf{b}_{k-1})}{\partial \mathbf{b}} \tag{31}$$

In order to accelerate the convergence of the algorithm, we can, for example, add momentum or use Nesterov method (Sutskever et al., 2013) to replace the gradient descent steps in (27)–(30). In our experiments reported in Section 5, we adopted Nesterov method to accelerate the training process.

Next, we describe the dual update, which aims to maximize $\mathcal{L}$ with respect to $\boldsymbol{\lambda}$ subject to the constraint that $\boldsymbol{\lambda} \succeq \mathbf{0}$. To this end, we use the following rule of gradient ascent with projection, which increases the function value of $\mathcal{L}$ while enforcing the constraint:

$$\lambda_{i,k} = \left[\lambda_{i,k-1} + \mu_k \left(\|\mathbf{w}_{i,k}\|_1 - \frac{1}{\gamma} \right) \right]_+ \tag{32}$$

where $[x]_+ \triangleq \max\{0, x\}$. Note that λ_i is indeed a regularization factor in $\mathcal{L}(\Theta, \boldsymbol{\lambda})$ that penalizes the violation of constraint (18) for the ith row of $\mathbf{W}$. The dual update can be interpreted as a rule to adjust the regularization factor in an adaptive manner. When the sum of the absolute values of the ith row of $\mathbf{W}$ exceeds $1/\gamma$, i.e., violating the constraint, the recursion (32) will increase the regularization factor $\lambda_{i,k}$ on the ith row in (19). On the other hand, if the constraint (18) for a certain i is not violated, i.e., $\|\mathbf{w}_i\|_1 < 1/\gamma$, then the dual update (32) will decrease the value of the corresponding λ_i so that $\|\mathbf{w}_i\|_1 - 1/\gamma$ is less penalized in (19). This process will repeat itself until the constraints (18) are satisfied. The projection operator $[x]_+$ makes sure that once the regularization factor λ_i is decreased below zero, it will be set to zero and the constraint for the ith row in (22) will not be penalized in (19). An alternative choice to enhance the constraint is to apply projection operator to $\mathbf{W}$ after each stochastic gradient descent update (29)–(31).

5 Experiments and Results

We use the TIMIT phone recognition task to evaluate the deep-RNN architecture and the primal-dual optimization method for training the RNN part of the network. The standard 462-speaker training set is used and all SA sentences are removed conforming to the standard protocol (Lee & Hon, 1989; Hinton et al., 2012; Deng et al., 2013a). A separate development set of 50 speakers is used for tuning all hyper parameters. Results are reported using the 24-speaker core test set, which has no overlap with the development set.

In our experiments, standard signal processing techniques are used for the raw speech waveforms, and 183 target class labels are used with three states for each of 61 phones. After decoding, the original 61 phone classes are mapped to a set of 39 classes for final scoring according to the standard

Table 3: Phone recognition error (percent) on the TIMIT core test set using DNN-middle and DNN-bottom features, as well as the raw filter-bank features.

Features	Order0	Order4	Order8	Order12
DNN-Middle	20.70	20.30	19.96	19.65
DNN-Bottom	23.10	22.65	22.00	21.50
Filter-Banks	30.50	30.15	29.40	28.15

evaluation protocol. In our experiments, a bi-gram language model over phones, estimated from the training set, is used in decoding. To prepare the DNN and RNN targets, a high-quality tri-phone HMM model is trained on the training data set, which is then used to generate state-level labels based on HMM forced alignment.

The DNN as part of the baseline described in (Deng et al., 2013a) is used in this work to extract the high-level features from raw speech features. That is, the DNN features are discriminatively learned. In our experiments, we use a DNN with three hidden layers, each having 2000 units. Each hidden layer's output vector, with a dimensionality of 2000, can be utilized as the DNN features. Thus, we have three sets of high-level features: DNN-top, DNN-middle, and DNN-bottom, indicated in Figure 1 with three separate colors of arrows pointing from DNN to RNN.

We first report the phone recognition error performance of the deep-RNN using the DNN-top features. In Table 1, the percent error results are shown as a function of the RNN hidden layer's size, varying from 100 to 2000, and also as a function of the moving average (MA) order in the RNN. Note when MA order is zero, the ARMA version of the RNN reverts to the traditional AR version.

The above results are obtained using the fixed insertion penalty of zero and the fixed bi-phone "language model" weight of one. When the language model's weight is tuned slightly over the development set, the core test set error further drops to 18.86%, which is close to the best numbers reported recently on this benchmark task in (Deng et al., 2013a) using deep convolutional neural networks with special design of the pooling strategy (18.70%) and in (Graves et al., 2013) using a bi-directional RNN with special design of the memory structure (17.7%). No bi-directionality and no special design on any structure are used in the RNN reported in this paper. The confusion matrix of this best result is shown in Section D of the supplementary document.

We next compare the new primal-dual training method with the classical BPTT using gradient clipping as described in (Pascanu et al., 2013). Table 2 shows the the phone recognition error of the classical BPTT with gradient clipping on the TIMIT benchmark. We found that the error rate is sensitive to the threshold value. The best phone error rate on the test set is found to be between 19.05%-20.5% over a wide range of the threshold values where the best tuned clipping threshold is around 1.0 which corresponds to the error rate of 19.05%. This is higher than the 18.91% from our primal-dual method (without tuning the language model weights). Thus, using the new method presented in the paper, we do not need to tune the hyper-parameter of clipping threshold while obtaining lower errors.

We finally show the phone recognition error (percent) results for the features of DNN-middle, DNN-bottom, and of raw filter-bank data. The size of the RNN's hidden layer is fixed at 500. And the results for four different MA orders are shown in Table 3.

Comparisons among the results in Tables 1 and 3 on phone recognition provide strong evidence that the high-level features extracted by DNNs are extremely useful for lowering recognition errors by the RNN. Further, the higher hidden layers in the DNN are more useful than the lower ones, given the same dimensionality of the features (fixed at 2000 as reported in this paper but we have the same conclusion for all other dimensions we have experimented). In addition, the introduction of MA components in the traditional AR version of the RNN also contributes significantly to reducing recognition errors.

6 Discussion and Conclusion

The main contribution of the work described in this paper is a formulation of the RNN that lends itself to effective learning using a formal optimization framework with a natural inequality constraint

that provides a sufficient condition to guarantee the stability of the RNN dynamics during learning. This new learning method overcomes the challenge of the earlier echo-state networks that typically fixed the recurrent weights due to the well-known difficulty in learning them. During the development of our new method, we propose a sufficient condition for the echo-state property, which is shown to be easily incorporated in the training procedure. We also make contributions to a new deep-RNN architecture, where the MA part is added to the original AR-only version of the RNN. The learning of both AR and ARMA versions of the RNN is unified after re-formulation of the model, and hence the same learning method developed can be applied to both versions.

The experimental contributions of this work are of four folds. First, we successfully apply the new learning method for the RNN to achieve close-to-record low error rates in phone recognition in the TIMIT benchmark. Second, we demonstrate the effectiveness of using DNNs to extract high-level features from the speech signal for providing the inputs to the RNN. With a combination of the DNN and RNN, we form a novel architecture of the deep-RNN, which, when trained separately, mitigates the overfitting problem. Third, on the same TIMIT benchmark task, we demonstrate clear superiority of the ARMA version of the RNN over the traditional AR version. The same efficient learning procedure is applied to both versions, with the only difference in the additional need to window the DNN outputs in the ARMA version of the RNN. Fourth, we show experimentally that the new training method motivated by optimization methodology achieves satisfactory performance as a sequence classifier on TIMIT benchmark task. Compared to the previous methods (Mikolov et al., 2011; Pascanu et al., 2013) of learning RNNs using heuristic rules of truncating gradients during the BPTT procedure, our new method reports slightly lower phone recognition errors on the TIMIT benchmark and no longer needs to tune the threshold parameter as in the previous methods.

References

Beck, A. and Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. Advances in optimizing recurrent networks. In *Proc. ICASSP*, Vancouver, Canada, May 2013.

Boyd, S. P. and Vandenberghe, L. *Convex Optimization*. Cambridge university press, 2004.

Dahl, G., Yu, D., Deng, L., and Acero, A. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Trans. on Audio, Speech and Language Processing*, 20(1):30–42, jan 2012.

Dahl, G. E., Yu, D., Deng, L., and Acero, A. Large vocabulary continuous speech recognition with context-dependent DBN-HMMs. In *Proc. IEEE ICASSP*, pp. 4688–4691, Prague, Czech, May 2011.

Dahl, G. E., Sainath, T. N., and Hinton, G. E. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Proc. ICASSP*, pp. 8609–8613, Vancouver, Canada, May 2013. IEEE.

Deng, L., Hassanein, K., and Elmasry, M. Analysis of the correlation structure for a neural predictive model with application to speech recognition. *Neural Networks*, 7(2):331–339, 1994.

Deng, L., Abdel-Hamid, O., and Yu, D. A deep convolutional neural network using heterogeneous pooling for trading acoustic invariance with phonetic confusion. In *Proc. IEEE ICASSP*, Vancouver, Canada, May 2013a.

Deng, L., Hinton, G., and Kingsbury, B. New types of deep neural network learning for speech recognition and related applications: An overview. In *Proc. IEEE ICASSP*, Vancouver, Canada, May 2013b.

Deng, L., Li, J., Huang, J.-T., Yao, K., Yu, D., Seide, F., Seltzer, M., Zweig, G., He, X., Williams, J., Gong, Y., and Acero, A. Recent advances in deep learning for speech research at microsoft. In *Proc. ICASSP*, Vancouver, Canada, 2013c.

Graves, A. Sequence transduction with recurrent neural networks. In *Representation Learning Workshp, ICML*, 2012.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proc. ICML*, pp. 369–376, Pittsburgh, PA, June 2006. ACM.

Graves, A., Mohamed, A., and Hinton, G. Speech recognition with deep recurrent neural networks. In *Proc. ICASSP*, Vancouver, Canada, May 2013.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012.

Jaeger, H. *The "echo state" approach to analysing and training recurrent neural networks*. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001a.

Jaeger, H. *Short term memory in echo state networks*. GMD Report 152, GMD - German National Research Institute for Computer Science, 2001b.

Jaeger, H. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*. GMD Report 159, GMD - German National Research Institute for Computer Science, 2002.

Kingsbury, B., Sainath, T. N., and Soltau, H. Scalable minimum bayes risk training of deep neural network acoustic models using distributed hessian-free optimization. In *Proc. INTERSPEECH*, Portland, OR, September 2012.

LeCun, Yann. Learning invariant feature hierarchies. In *Proc. ECCV*, pp. 496–505, Firenze, Italy, October 2012. Springer.

Lee, K.-F. and Hon, H.-W. Speaker-independent phone recognition using hidden Markov models. *IEEE Transactions on Acoustics, Speech and Signal Processing,*, 37(11):1641–1648, November 1989.

Maas, A. L., Le, Q., O'Neil, T. M., Vinyals, O., Nguyen, P., and Ng, A. Y. Recurrent Neural Networks for Noise Reduction in Robust ASR. In *Proc. INTERSPEECH*, Portland, OR, September 2012.

Manjunath, G. and Jaeger, H. Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks. *Neural computation*, 25(3):671–696, 2013.

Martens, J. and Sutskever, I. Learning recurrent neural networks with hessian-free optimization. In *Proc. ICML*, pp. 1033–1040, Bellevue, WA, June 2011.

Mikolov, T. *Statistical Language Models Based on Neural Networks*. PhD thesis, Ph. D. thesis, Brno University of Technology, 2012.

Mikolov, T., Deoras, A., Povey, D., Burget, L., and Cernocky, J. Strategies for training large scale neural network language models. In *Proc. IEEE ASRU*, pp. 196–201, Honolulu, HI, December 2011. IEEE.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *Proc. ICML*, Atlanta, GA, June 2013.

Polyak, B. *Introduction to Optimization*. Optimization Software, NY, 1987.

Robinson, A. J. An application of recurrent nets to phone probability estimation. *IEEE Transactions on Neural Networks*, 5(2):298–305, August 1994.

Sainath, T.N., Kingsbury, B., Soltau, H., and Ramabhadran, B. Optimization techniques to improve training speed of deep neural networks for large speech tasks. *IEEE Transactions on Audio, Speech, and Language Processing*, 21(11):2267–2276, November 2013.

Seide, F., Li, G., and Yu, D. Conversational speech transcription using context-dependent deep neural networks. In *Proc. INTERSPEECH*, pp. 437–440, Florence, Italy, August 2011.

Sutskever, I. *Training Recurrent Neural Networks*. PhD thesis, Ph. D. thesis, University of Toronto, 2013.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. E. On the importance of initialization and momentum in deep learning. In *Proc. ICML*, Atlanta, GA, June 2013.

Tüske, Z., Sundermeyer, M., Schlüter, R., and Ney, H. Context-Dependent MLPs for LVCSR: TANDEM, Hybrid or Both? In *Proc. Interspeech*, Portland, OR, September 2012.

Vinyals, Oriol, Ravuri, Suman V, and Povey, Daniel. Revisiting recurrent neural networks for robust ASR. In *Proc. ICASSP*, pp. 4085–4088, Kyoto, Japan, March 2012. IEEE.

Yu, D., Deng, L., and Dahl, G. Roles of pre-training and fine-tuning in context-dependent DBN-HMMs for real-world speech recognition. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

Yu, D., Deng, L., and Seide, F. The deep tensor neural network with applications to large vocabulary speech recognition. *IEEE Trans. on Audio, Speech and Language Processing*, 21(2):388 –396, 2013.

Supplementary Material

A Proof of a sufficient condition for echo-state property

First, we prove the following property regarding $\mathbf{f}(\mathbf{x})$:

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\|_\infty \leq \gamma \cdot \|\mathbf{x} - \mathbf{y}\|_\infty \tag{33}$$

where $\gamma \triangleq \max f'(x)$. Let x_k and y_k denote the k-th entries of the $N \times 1$ vectors $\mathbf{x}$ and $\mathbf{y}$, respectively. Then,

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\|_\infty = \max_{1 \leq k \leq N} |f(x_k) - f(y_k)| \tag{34}$$

By mean value theorem, we have

$$f(x_k) = f(y_k) + \int_0^1 f'\big(y_k + t(x_k - y_k)\big) dt (x_k - y_k) \tag{35}$$

so that

$$\begin{aligned}
&|f(x_k) - f(y_k)| \\
&\leq \left| \int_0^1 f'\big(y_k + t(x_k - y_k)\big) dt (x_k - y_k) \right| \\
&\leq \int_0^1 \left| f'\big(y_k + t(x_k - y_k)\big) \right| dt \cdot |x_k - y_k| \\
&\leq \int_0^1 \gamma \cdot dt \cdot |x_k - y_k| \\
&\leq \gamma \cdot |x_k - y_k|
\end{aligned} \tag{36}$$

Substituting the above result into (34), we get

$$\begin{aligned}
\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{y})\|_\infty &\leq \gamma \cdot \max_{1 \leq k \leq N} |x_k - y_k| \\
&= \gamma \cdot \|\mathbf{x} - \mathbf{y}\|_\infty
\end{aligned} \tag{37}$$

From the expressions of $\tanh$ and sigmoid functions, we can easily verify that $\gamma = 1$ for $\tanh$ function and $\gamma = 1/4$ for sigmoid function.

Let $\mathbf{h}_t$ and $\mathbf{h}'_t$ denote the hidden states of the recurrent neural network (1) that starts at initial conditions $\mathbf{h}_0$ and $\mathbf{h}'_0$, respectively:

$$\mathbf{h}_t = \mathbf{f}\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{W}_I \mathbf{v}_t + \mathbf{b}\right) \tag{38}$$

$$\mathbf{h}'_t = \mathbf{f}\left(\mathbf{W}\mathbf{h}'_{t-1} + \mathbf{W}_I \mathbf{v}_t + \mathbf{b}\right) \tag{39}$$

Subtracting (39) from (38), we obtain

$$\begin{aligned}
\mathbf{h}_t - \mathbf{h}'_t = &\mathbf{f}\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{W}_I \mathbf{v}_t + \mathbf{b}\right) \\
&- \mathbf{f}\left(\mathbf{W}\mathbf{h}'_{t-1} + \mathbf{W}_I \mathbf{v}_t + \mathbf{b}\right)
\end{aligned} \tag{40}$$

Taking ∞-norm of both sides of the above expression and using (37), we get

$$\begin{aligned}
&\|\mathbf{h}_t - \mathbf{h}'_t\|_\infty \\
&= \big\|\mathbf{f}\left(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{W}_I \mathbf{v}_t + \mathbf{b}\right) \\
&\quad - \mathbf{f}\left(\mathbf{W}\mathbf{h}'_{t-1} + \mathbf{W}_I \mathbf{v}_t + \mathbf{b}\right) \big\|_\infty \\
&\leq \gamma \cdot \big\|\mathbf{W}(\mathbf{h}_{t-1} - \mathbf{h}'_{t-1})\big\|_\infty \\
&\leq \gamma \cdot \|\mathbf{W}\|_\infty \cdot \|\mathbf{h}_{t-1} - \mathbf{h}'_{t-1}\|_\infty \\
&\leq \big(\gamma \cdot \|\mathbf{W}\|_\infty\big)^t \cdot \|\mathbf{h}_0 - \mathbf{h}'_0\|_\infty
\end{aligned} \tag{41}$$

Let $\epsilon_t \triangleq (\gamma \cdot \|\mathbf{W}\|_\infty)^t \cdot \|\mathbf{h}_0 - \mathbf{h}'_0\|_\infty$. Obviously, as long as $\gamma \cdot \|\mathbf{W}\|_\infty < 1$ or, equivalently, $\|\mathbf{W}\|_\infty < 1/\gamma$, we have $\epsilon_t \to 0$ and we can conclude the network (1) is state contracting and thus has echo-state property.

B Gradient formula for BPTT

Specifically, let $\mathbf{p}_t$ be a $N \times 1$ vector that collects the input to the hidden units:

$$\mathbf{p}_t \triangleq \mathbf{W}\mathbf{h}_t + \mathbf{W}_I\mathbf{v}_t + \mathbf{b} \tag{42}$$

Let $\boldsymbol{\delta}_t$ denote the error signal to be propagated:

$$\boldsymbol{\delta}_t \triangleq T \cdot \frac{\partial J}{\partial \mathbf{p}_t} = \frac{\partial}{\partial \mathbf{p}_t} \sum_{t=1}^{T} J_t(\mathbf{y}_t, \mathbf{d}_t) \tag{43}$$

Then, by chain rule (see Section C of the supplementary document for details), $\boldsymbol{\delta}_t$ is propagated through time according to the following backward recursion:

$$\boldsymbol{\delta}_t = \mathbf{D}_f \mathbf{W}^T \cdot \boldsymbol{\delta}_{t+1} + \frac{\partial J_t}{\partial \mathbf{p}_t}, \quad t = T, T-1, \ldots, 0 \tag{44}$$

with initialization $\boldsymbol{\delta}_{T+1} = \mathbf{0}$, where $\mathbf{D}_f$ is a diagonal matrix constructed from the element-wise derivative of $\mathbf{f}(\mathbf{x})$:

$$\mathbf{D}_f \triangleq \mathrm{diag}\{\mathbf{f}'(\mathbf{p}_t)\} \tag{45}$$

The form of $\frac{\partial J_t}{\partial \mathbf{p}_t}$ depends on the choice of cost functions J_t and the output units $\mathbf{g}(\mathbf{x})$. For example, for linear output units and square-error cost, i.e.,

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} \tag{46}$$

$$J_t(\mathbf{y}_t, \mathbf{d}_t) = \|\mathbf{y}_t - \mathbf{d}_t\|^2 \tag{47}$$

the expression for $\frac{\partial J_t}{\partial \mathbf{p}_t}$ is given by

$$\frac{\partial J_t}{\partial \mathbf{p}_t} = -2\mathbf{U}^T(\mathbf{d}_t - \mathbf{y}_t) \tag{48}$$

On the other hand, if soft-max output units and cross-entropy cost are used, i.e.,

$$\mathbf{g}(\mathbf{x})_n = \frac{e^{x_n}}{\sum_{n=1}^{N_o} e^{x_n}} \tag{49}$$

$$J_t(\mathbf{y}_t, \mathbf{d}_t) = -\sum_{n=1}^{N_o} d_{n,t} \ln y_{n,t} \tag{50}$$

where $d_{n,t}$ and $y_{n,t}$ denote the nth entry of the vectors $\mathbf{d}_t$ and $\mathbf{y}_t$, respectively, the expression for $\frac{\partial J_t}{\partial \mathbf{p}_t}$ will be given by

$$\frac{\partial J_t}{\partial \mathbf{p}_t} = -(\mathbf{d}_t - \mathbf{y}_t) \tag{51}$$

And the gradients of $J(\theta)$ with respect to the parameters $\mathbf{W}$, $\mathbf{W}_I$, $\mathbf{U}$ and $\mathbf{b}$ are given by the following expressions:

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\delta}_t \mathbf{h}_{t-1}^T \tag{52}$$

$$\frac{\partial J}{\partial \mathbf{W}_I} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\delta}_t \mathbf{v}_t^T \tag{53}$$

$$\frac{\partial J}{\partial \mathbf{U}} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial J_t}{\partial \mathbf{U}} \tag{54}$$

$$\frac{\partial J}{\partial \mathbf{b}} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{\delta}_t \tag{55}$$

For soft-max output units with cross-entropy cost, which we use in this work, $\frac{\partial J_t}{\partial \mathbf{U}}$ is given by

$$\frac{\partial J_t}{\partial \mathbf{U}} = -(\mathbf{d}_t - \mathbf{y}_t)\mathbf{h}_t^T \tag{56}$$

C Derivation of the back propagation recursion

From the definition of the error signal in (43), we can write $\boldsymbol{\delta}_t$ as

$$\begin{aligned}\boldsymbol{\delta}_t &= \frac{\partial}{\partial \mathbf{p}_t} \sum_{u=1}^{T} J_u \\ &= \sum_{u=1}^{T} \frac{\partial J_u}{\partial \mathbf{p}_t}\end{aligned} \tag{57}$$

In RNN, only J_u with $u \geq t$ depend on $\mathbf{p}_t$, so we have $\frac{\partial J_u}{\partial \mathbf{p}_t} = 0$ for $u < t$ and hence

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{p}_t} &= \sum_{u=t}^{T} \frac{\partial J_u}{\partial \mathbf{p}_t} \\ &= \sum_{u=t+1}^{T} \frac{\partial J_u}{\partial \mathbf{p}_t} + \frac{\partial J_t}{\partial \mathbf{p}_t} \\ &= \sum_{u=t+1}^{T} \frac{\partial \mathbf{p}_t^T}{\partial \mathbf{p}_{t+1}} \cdot \frac{\partial J_u}{\partial \mathbf{p}_{t+1}} + \frac{\partial J_t}{\partial \mathbf{p}_t} \\ &= \frac{\partial \mathbf{p}_t^T}{\partial \mathbf{p}_{t+1}} \cdot \sum_{u=t+1}^{T} \frac{\partial J_u}{\partial \mathbf{p}_{t+1}} + \frac{\partial J_t}{\partial \mathbf{p}_t} \\ &= \frac{\partial \mathbf{p}_t^T}{\partial \mathbf{p}_{t+1}} \cdot \sum_{u=1}^{T} \frac{\partial J_u}{\partial \mathbf{p}_{t+1}} + \frac{\partial J_t}{\partial \mathbf{p}_t} \\ &= \frac{\partial \mathbf{p}_t^T}{\partial \mathbf{p}_{t+1}} \cdot \boldsymbol{\delta}_{t+1} + \frac{\partial J_t}{\partial \mathbf{p}_t}\end{aligned} \tag{58}$$

Now we evaluate $\frac{\partial \mathbf{p}_t^T}{\partial \mathbf{p}_{t+1}}$, which by chain rule, can be written as

$$\frac{\partial \mathbf{p}_t^T}{\partial \mathbf{p}_{t+1}} = \frac{\partial \mathbf{h}_t^T}{\partial \mathbf{p}_t} \cdot \frac{\partial \mathbf{p}_{t+1}^T}{\partial \mathbf{h}_t} \tag{59}$$

By the expression of $\mathbf{p}_t$ in (42), we have

$$\begin{aligned}\frac{\partial \mathbf{p}_{t+1}^T}{\partial \mathbf{h}_t} &= \frac{\partial}{\partial \mathbf{h}_t} \left\{\mathbf{W}\mathbf{h}_t + \mathbf{W}_I \mathbf{v}_t + \mathbf{b}\right\}^T \\ &= \mathbf{W}^T\end{aligned} \tag{60}$$

and by (1), we have

$$\begin{aligned}\frac{\partial \mathbf{h}_t^T}{\partial \mathbf{p}_t} &= \frac{\partial}{\partial \mathbf{p}_t} \left\{\mathbf{f}(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{W}_I \mathbf{v}_t + \mathbf{b})\right\}^T \\ &= \frac{\partial}{\partial \mathbf{p}_t} \left\{\mathbf{f}(\mathbf{p}_t)\right\}^T \\ &= \mathbf{f}'(\mathbf{p}_t) \\ &= \mathbf{D}_f\end{aligned} \tag{61}$$

Substituting (60)–(61) into (59), we get

$$\frac{\partial \mathbf{p}_t^T}{\partial \mathbf{p}_{t+1}} = \mathbf{D}_f \mathbf{W}^T \tag{62}$$

Substituting (62) into (58), we conclude our proof of (44). Next, we derive (52)–(54). We will only show the proof of (52). By chain rule, we have

$$\frac{\partial J}{\partial W_{ij}} = \frac{1}{T} \cdot \frac{\partial}{\partial W_{ij}} \left(\sum_{u=1}^{T} J_u \right)$$

$$= \frac{1}{T} \cdot \sum_{t=1}^{T} \frac{\partial \mathbf{p}_t^T}{\partial W_{ij}} \cdot \frac{\partial}{\partial \mathbf{p}_t} \left(\sum_{u=1}^{T} J_u \right)$$
$$= \frac{1}{T} \sum_{t=1}^{T} \frac{\partial \mathbf{p}_t^T}{\partial W_{ij}} \boldsymbol{\delta}_t \tag{63}$$

By (42), we have

$$\left[\frac{\partial \mathbf{p}_t^T}{\partial W_{ij}} \right]_n = \begin{cases} [\mathbf{h}_t]_j & n = i \\ 0 & n \neq i \end{cases} \tag{64}$$

where the notatin $[\mathbf{x}]_i$ denotes the ith entry of the vector. Therefore,

$$\frac{\partial J}{\partial W_{ij}} = \frac{1}{T} \sum_{t=1}^{T} [\mathbf{h}_t]_j [\boldsymbol{\delta}_t]_i \tag{65}$$

which implies that, in matrix form,

$$\frac{\partial J}{\partial \mathbf{W}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{\delta}_t \mathbf{h}_t^T \tag{66}$$

D Confusion matrix of phone recognition

See the figure in next page.

WORD: %Corr=83.31, Acc=81.14 [H=6109, D=395, S=829, I=159, N=7333]

---------- Confusion Matrix ----------

	a a	a e	a h	a w	a y	b	c h	d	d h	d x	e h	e r	e y	f	g	h h	h i	i y	i h	j	k	l	m	n	n g	o w	o y	p	p a u	r	s	s h	t	t h	u h	u w	v	w	y	z	Del [%c / %e]
aa	183	2	15	3	8	0	0	0	0	1	2	1	0	0	0	1	0	0	0	0	0	0	0	0	2	1	0	0	1	1	0	0	0	0	0	0	0	0	7	[82.8/0.5]
ae	2	66	3	0	0	0	0	0	0	0	19	0	1	0	0	0	3	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	8	[68.0/0.4]
ah	7	0	226	0	3	0	0	2	0	0	4	8	0	0	0	0	33	0	0	0	0	0	0	0	2	0	1	1	1	0	0	0	0	2	0	0	0	0	31	[77.9/0.9]
aw	0	2	0	25	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	[83.3/0.1]
ay	5	0	4	1	71	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	5	[84.5/0.2]
b	0	0	0	0	0	112	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	7	1	0	0	0	0	0	0	0	1	0	0	8	[90.3/0.2]
ch	0	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	0	0	0	0	[87.5/0.1]
d	0	0	0	0	0	1	0	78	4	4	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	1	0	0	0	0	0	9	[75.0/0.4]
dh	0	0	0	0	0	0	0	5	97	2	0	0	0	0	0	1	0	0	0	1	3	1	0	0	0	0	0	0	0	0	0	2	3	0	0	2	0	0	4	7	[80.2/0.3]
dx	0	0	0	0	0	0	0	4	3	74	0	0	0	0	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	[87.1/0.2]
eh	0	8	23	0	2	0	0	0	0	0	117	1	0	0	0	0	24	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	12	[66.1/0.8]
er	1	0	1	0	0	0	0	0	0	0	1	193	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	1	18	0	0	0	0	1	0	0	0	0	7	[85.0/0.5]
ey	0	0	0	0	1	0	0	0	0	0	2	0	98	0	0	0	6	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	[88.3/0.2]
f	0	0	0	0	0	2	0	0	0	0	1	0	0	118	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	3	0	0	3	0	1	[90.8/0.2]
g	0	0	0	0	0	1	0	3	0	0	0	0	0	0	56	0	0	0	0	3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	[87.5/0.1]
hh	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	59	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	14	[93.7/0.1]
ih	0	0	33	0	0	0	0	0	0	0	16	11	6	0	0	1	444	19	0	0	0	0	0	0	2	0	0	2	0	1	0	0	0	4	4	0	0	0	37	[81.8/1.4]
iy	0	1	0	0	0	0	0	0	0	0	0	0	3	0	0	0	18	211	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	2	5	[88.7/0.4]
jh	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	[85.4/0.1]
k	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	148	0	0	0	0	0	0	1	0	0	0	0	2	0	0	0	0	0	0	10	[95.5/0.1]
l	5	0	3	2	0	1	0	0	3	0	0	0	0	0	0	0	0	0	0	0	233	3	0	0	9	0	1	0	4	0	1	1	0	2	0	2	1	0	20	[86.0/0.5]
m	0	1	2	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	178	7	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	9	[91.3/0.2]
n	1	0	2	0	0	0	0	1	2	9	0	1	0	0	0	0	3	0	0	0	1	7	319	6	0	0	1	4	0	1	0	1	0	0	0	0	0	0	26	[88.9/0.5]
ng	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	8	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	[79.6/0.1]
ow	6	0	3	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	5	0	0	0	65	1	0	0	0	0	0	0	0	0	1	1	0	0	4	[76.5/0.3]
oy	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	0	0	0	1	0	0	0	0	11	0	0	1	0	0	0	0	0	0	0	0	0	0	[68.8/0.1]
p	0	0	0	0	0	5	0	1	1	0	0	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0	116	0	0	0	0	3	0	0	0	1	0	0	6	[89.2/0.2]
pau	0	0	3	0	0	0	0	0	0	2	3	0	2	0	0	0	1	1	0	1	0	0	1	1	2	0	0	1479	0	1	0	0	0	0	0	2	0	0	80	[98.7/0.3]
r	0	0	0	1	1	0	0	0	0	0	1	9	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	217	0	0	0	0	0	0	2	0	0	36	[92.7/0.2]
s	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	302	0	2	0	0	0	0	0	8	4	[95.3/0.2]
sh	0	0	1	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	2	66	1	0	0	0	0	0	0	0	[85.7/0.2]
t	0	0	0	0	0	0	2	6	2	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	163	3	0	0	0	0	0	8	[91.1/0.2]
th	0	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2	28	0	0	0	0	2	1	[75.7/0.1]
uh	0	0	6	0	0	0	0	0	0	0	0	1	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	1	0	0	0	3	[53.8/0.2]
uw	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	8	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	60	0	0	0	0	[81.1/0.2]
v	0	0	1	0	0	3	0	0	6	0	0	0	0	1	0	0	1	1	0	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	68	0	0	1	6	[78.2/0.3]
w	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	127	0	0	9	[94.1/0.1]
y	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	38	0	6	[86.4/0.1]
z	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	33	0	0	0	1	1	0	0	0	140	2	[78.2/0.5]
Ins	8	1	13	0	3	2	1	8	5	3	3	3	2	1	2	4	14	1	0	6	2	2	2	2	3	0	3	39	9	3	0	2	0	1	3	1	3	4	0		