Equilibrated Recurrent Neural Network: Neuronal Time-Delayed Self-Feedback Improves Accuracy and Stability

Zhang, Z.; Kag, A.; Sullivan, A.; Saligrama, V.

Abstract

We propose a novel Equilibrated Recurrent Neural Network (ERNN) to combat the issues of inaccuracy and instability in conventional RNNs. Drawing upon the concept of autapse in neuroscience, we propose augmenting an RNN with a time-delayed self-feedback loop. Our sole purpose is to modify the dynamics of each internal RNN state and, at any time, enforce it to evolve close to the equilibrium point associated with the input signal at that time. We show that such selffeedback helps stabilize the hidden state transitions leading to fast convergence during training while efficiently learning discriminative latent features that result in state-of-the-art results on several benchmark datasets at test-time. We propose a novel inexact Newton method to solve fixedpoint conditions given model parameters for generating the latent features at each hidden state. We prove that our inexact Newton method converges locally with linear rate (under mild conditions). We leverage this result for efficient training of ERNNs based on backpropagation.

arXiv

Equilibrated Recurrent Neural Network:
Neuronal Time-Delayed Self-Feedback Improves Accuracy and Stability

Ziming Zhang [* 1]   **Anil Kag** [* 2]   **Alan Sullivan** [1]   **Venkatesh Saligrama** [2]

Abstract

We propose a novel *Equilibrated Recurrent Neural Network* (ERNN) to combat the issues of inaccuracy and instability in conventional RNNs. Drawing upon the concept of autapse in neuroscience, we propose augmenting an RNN with a time-delayed self-feedback loop. Our sole purpose is to modify the dynamics of each internal RNN state and, at any time, enforce it to evolve close to the equilibrium point associated with the input signal at that time. We show that such self-feedback helps stabilize the hidden state transitions leading to fast convergence during training while efficiently learning discriminative latent features that result in state-of-the-art results on several benchmark datasets at test-time. We propose a novel inexact Newton method to solve fixed-point conditions given model parameters for generating the latent features at each hidden state. We prove that our inexact Newton method converges locally with linear rate (under mild conditions). We leverage this result for efficient training of ERNNs based on backpropagation.

1. Introduction

Recurrent neural networks (RNNs) are useful tools to analyze sequential data, and have been widely used in many applications such as natural language processing (Sutskever et al., 2014) and computer vision (Hori et al., 2017). It is well-known that training RNNs is particularly challenging because we often encounter both diverging as well as vanishing gradients (Pascanu et al., 2013). In this paper we propose an efficient training algorithm for RNNs that substantially improves convergence during training while achieving state-of-the-art generalization at test-time.

---

[*]Equal contribution [1]Mitsubishi Electric Research Laboratories (MERL), MA, USA [2]Department of Electrical and Computer Engineering, Boston University, MA, USA. Correspondence to: Ziming Zhang <zzhang@merl.com>.

Motivation: We draw our inspiration from works in Neuroscience (Seung et al., 2000), which describe a concept called *Autapse*. According to this concept, some neurons in the neocortex and hippocampus are found to enforce *time-delayed self-feedback* as a means to control, excite and stabilize neuronal behavior. In particular, researchers (*e.g.* (Qin et al., 2015)) have found that negative feedback tends to damp excitable neuronal behavior while positive feedback can excite quiescent neurons. Herrmann & Klaus (2004) have experimented with physical simulation based on artificial autapses, and more recently Fan et al. (2018) have demonstrated how autapses lead to enhanced synchronization resulting in better coordination and control of neuronal networks.
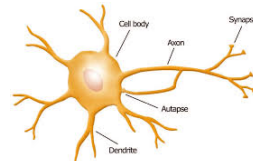

Figure 1. Illustration of autapse for a neuron (Herrmann & Klaus, 2004).

In this light we propose two modifications in the architecture and operation of RNNs as follows:

M1. Adding self-feedback to each RNN hidden state;
M2. Processing RNNs with self-feedback based on upsampled inputs interpolated with a constant filter so that each input sequence can achieve a *set-point* over time.

These modifications lead to our novel *Equilibrated Recurrent Neural Network (ERNN)*, as illustrated in Fig. 2. Compared with conventional RNNs, our ERNNs introduce a self-feedback link to each hidden state in the unfolded networks. Using the same unfolding trick, we repeat each input word as an upsampled input sequence. This step essentially generates a time-delayed hidden state towards the equilibrium point. In summary, based on the self-feedback loops, our ERNNs can be considered as *RNN-in-RNN* networks where the outer loop accounts for timesteps whereas the inner loop accounts for time-delay internal state dynamics.

We further demonstrate that such self-feedback helps in:

H1. Stabilizing the system that allows for equilibration of the internal state evolution;
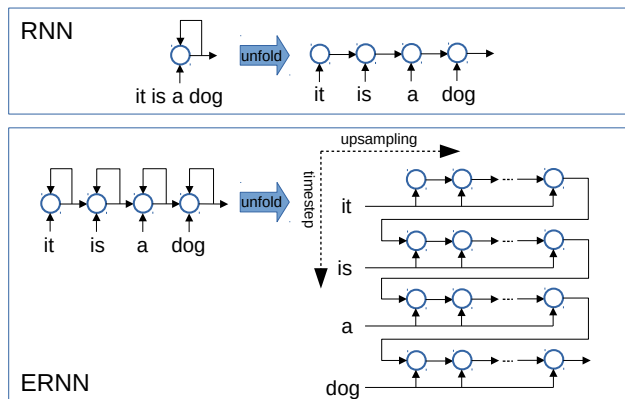H2. Learning discriminative latent features;

Figure 2. Comparison between **(top)** RNN and **(bottom)** ERNN.

H3. Accelerating convergence in training;

H4. Achieving good generalization in testing.

Validation of Proposed Method on Toy Data: We compare conventional RNN with our ERNN in Fig. 2. We define the dynamics in the RNN as follows:

$$\mathbf{h}_t = \tanh\left(\mathbf{V}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}\right), \tag{1}$$

where $\mathbf{x}_t, \mathbf{h}_t$ denote the input data and output latent feature at t-th timestep, respectively, and $\mathbf{V}, \mathbf{W}, \mathbf{b}$ are the model parameters. Similarly we consider a special case of ERNN for exposition:

$$\mathbf{h}_t = \tanh\left(\mathbf{h}_t + \mathbf{V}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}\right) \tag{2}$$

so that both models contain the same number of parameters. Notice that here $\mathbf{h}_t$ is a fixed point of the equation [1].

We generate a toy dataset to demonstrate the effectiveness of our approach. Our dataset is a collection of 2D random walks with Gaussian steps: $X_t \sim N\left(X_{t-1}, \sigma_i I\right) \in \mathbb{R}^2$ with $X_0 = \mathbf{0}, t \in [100]$, and $\sigma_0 = 0.1, \sigma_1 = 1$ corresponding to two classes. We generate 10^4 walks for each class and choose half the size for training and leave the rest for testing.

Both RNN and ERNN are endowed with a 10-dimensional hidden state. As is the convention, we utilize the state in the last timestep for classification. We tune the hyperparameters such as learning rate to achieve best model parameters for each network, where we train our ERNN using the proposed method in Sec. 3.

To validate (H1), we compare the Euclidean distances between the model after each epoch and the one after the final epoch, which we consider is convergent during training. Our results in Fig. 3(a) show that the change of ERNN per epoch

---

[1]While convergence to equilibrium is outside the scope of the paper, we point to control theorists who describe conditions (Barabanov & Prokhorov, 2002) for these types of recurrence.
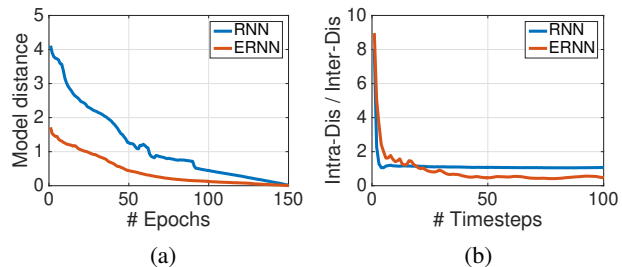

Figure 3. Validation of **(a)** H1 and **(b)** H2 on toy data.

is always smaller than RNN, which reveals that neuronal self-feedback indeed improves the stability in hidden states.

To validate (H2), we compare the feature discrimination over timesteps after training by computing the ratio of intra-class distance and inter-class distance. Our results in Fig. 3(b) show that the discrimination of ERNN gradually improves and after a period of transition is substantially superior to RNN, whose discrimination saturates very quickly.

To validate (H3), we show the training losses of each network over epochs in Fig. 4. As we see, ERNN converges to lower values. At test-time, RNN and ERNN achieve accuracy of 86.6% and 99.7%, respectively, validating (H4).
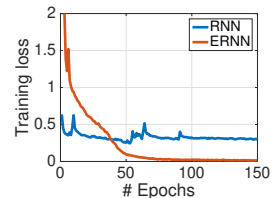

Figure 4. Validation of H3.

Vanishing and Exploding Gradients: In light of these results, we speculate how autapse could possibly combat this issue. While we have an incomplete understanding of ERNNs, we will provide some intuition. Fixed points can be iteratively approximated with our inexact Newton method. This method involves setting the next iterate to be equal to previous iterate and a weighted error term. We refer to Sec 3.3 for further details. In essence, this iteration implicitly encodes skip/residual connections, which have been shown recently to overcome vanishing and exploding gradient issue (Kusupati et al., 2018b).

A different argument is based on defining the equilibrium point for Eq. 2 in terms of an abstract map, Φ, that satisfies $\mathbf{h}_t = \Phi(\mathbf{V}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b})$. In this case, the fixed points are entry-wisely independent and thus we

Figure 5. Illustration of fixed-point and derivative of Φ *vs.* α

can consider each entry in the fixed points in isolation. As numerically illustrated in Fig. 5, we find that the equilibrium point of $h = \tanh(h + \alpha)$ is some function
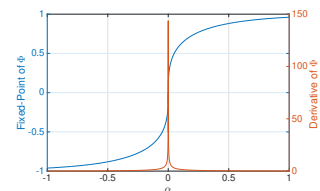
$h = \Phi(\alpha)$, where $\alpha = [\mathbf{V}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}]_j$ is the j-th entry. Theoretically the derivative of Φ at $\alpha = 0$ is infinity, *i.e.* unbounded. This observation violates the bounded conditions in (Pascanu et al., 2013) for the existence of vanishing gradients. We conjecture that this point lies at the root of why vanishing gradients in training does not arise.

While we do not completely understand the issue of exploding gradients, we conjecture that our method learns the parameters, *i.e.* $\mathbf{V}, \mathbf{W}, \mathbf{b}$ for Φ, such that the α equal to zero is not a local optima and in this way ERNN is able to handle exploding gradients as well. In fact, in our experiments we did not encounter exploding gradients as an issue.

Contributions: Our key contributions are as follows:

C1. We propose a novel Equilibrated Recurrent Neural Network (ERNN) as defined in Sec. 3.1 drawing upon the concept of autapse from the Neuroscience literature. Our method improves training stability of RNNs as well as the accuracy.

C2. We propose a novel inexact Newton method for fixed-point iteration given learned models in Sec. 3.2. We prove that under mild conditions our fixed-point iteration algorithm can converge locally at linear rate.

C3. We propose a principled way in Sec. 3.3 to convert ERNNs into trainable networks with residual connections using backpropagation. We demonstrate that FastRNN (Kusupati et al., 2018a) is a special ERNN.

C4. We present an exemplar ERNN in Sec. 3.4. In Sec. 4 we compare this model with other state-of-the-art RNNs empirically to show its superiority in terms of accuracy as well as training efficiency.

2. Related Work

We summarize related work from the following aspects:

Optimizers: Backpropagation through time (BPTT) (Werbos, 1990) is a generalization of backpropagation that can compute gradients in RNNs but suffer from large storage of hidden states in long sequences. Its truncated counterpart, truncated backpropagation through time (TBPTT) (Jaeger, 2002), is widely used in practice but suffers from learning long-term dependencies due to the truncation bias. The Real Time Recurrent Learning algorithm (RTRL) (Williams & Zipser, 1989) addresses this issue at the cost of high computational requirement. Recently several papers address the computational issue in the RNN optimizers, for instance, from the perspective of effectiveness (Martens & Sutskever, 2011; Cho et al., 2015), storage (Gruslys et al., 2016; Tallec & Ollivier, 2017; Mujika et al., 2018; MacKay et al., 2018; Liao et al., 2018) or parallelization (Bradbury et al., 2016; Lei et al., 2017; Martin & Cundy, 2018).

Feedforward vs. Recurrent: The popularity of recurrent models stems from the fact that they are particularly well-suited for sequential data, which exhibits long-term temporal dependencies. Nevertheless, an emerging line of research has highlighted several shortcomings of RNNs. In particular, apart from the well-known issues of exploding and vanishing gradients, several authors (van den Oord et al., 2016; Gehring et al., 2017; Vaswani et al., 2017; Dauphin et al., 2017) have observed that sequential processing leads to large training and inference costs because RNNs are inherently not parallelizable. To overcome these shortcomings they have proposed methods that replace recurrent models with parallelizable feedforward models which truncate the receptive field, and such feedforward structures have begun to show promise in a number of applications. Motivated by these works, Miller & Hardt (2018) have attempted to theoretically justify these findings. In particular, their work shows that under strong assumptions, namely, the class of so-called "stable" RNN models, recurrent models can be well-approximated by feedforward structures with a relatively small receptive field. Nevertheless, the assumption that RNNs are stable appears to be too strong as we have seen in a number of our experiments and so it does not appear possible to justify the usage of limited receptive field and feedforward networks in a straightforward manner.

In contrast, our paper does not attempt to a priori limit the receptive field. Indeed, the equilibrium states derived here are necessarily functions of both the input and the prior state and they influence the equilibrium solution since we are dealing with an underlying non-linear dynamical system. Instead, we show that ERNNs operating near equilibrium lend themselves to an inexact Newton method, whose sole purpose is to force the state towards equilibrium solution. In summary, although, there are parallels between our work and the related feedforward literature, this similarity appears to be coincidental and superficial.

Architectures: Our work can also be related to a number of related works that attempt to modify RNNs to improve the issues arising from exploding and vanishing gradients.

Long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) is widely used in RNNs to model long-term dependency in sequential data. Gated recurrent unit (GRU) (Cho et al., 2014) is another gating mechanism that has been demonstrated to achieve similar performance of LSTM with fewer parameters. Unitary RNNs (Arjovsky et al., 2016; Jing et al., 2017) is another family of RNNs that consist of well-conditioned state transition matrices.

Recently residual connections have been applied to RNNs with remarkable improvement of accuracy and efficiency for learning long-term dependency. For instance, Chang et al. (2017) proposed dilated recurrent skip connections. Campos et al. (2017) proposed Skip RNN to learn to skip state updates. Kusupati et al. (2018b) proposed FastRNN by adding

a residual connection to handle inaccurate training and in-efficient prediction. They further proposed FastGRNN by extending the residual connection to a gate, which involves low-rank approximation, sparsity, and quantization (LSQ) as well to reduce model size.

In contrast to these works, our work is based on enforcing RNNs through time-delayed self-feedback. Our work leverages an inexact Newton method for efficient training of ERNNs. Our method generalizes FastRNN in (Kusupati et al., 2018b) in this context.

3. Equilibrated Recurrent Neural Network

3.1. Problem Definition

Without loss of generality, we consider our ERNNs in the context of supervised learning. That is,

$$\min_{\phi \in \Phi, \omega \in \Omega} \sum_i \ell(\mathbf{h}_{i,T}, y_i; \omega), \tag{3}$$

$$\text{s.t.} \quad \mathbf{h}_{i,t} = \phi(\mathbf{h}_{i,t}, \mathbf{h}_{i,t-1}, \mathbf{x}_{i,t}), \forall i, \forall t \in [T].$$

Here $\{(x_i, y_i)\}$ denotes a collection of training data where $x_i = \{\mathbf{x}_{i,t}\}_{t=1}^T \subseteq \mathbb{R}^d, \forall i$ denotes the i-th training sample consisting of T timesteps and y_i denotes its associated label. $\phi : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^d \rightarrow \mathbb{R}^n$ denotes a (probably nonconvex) differentiable state-transition mapping function learned from a feasible solution space Φ, $\mathbf{h}_{i,t} \in \mathbb{R}^n, \forall i, \forall t$ denotes a hidden state for sample i at time t, and ℓ denotes a loss function parameterized by ω that is also learned from a feasible solution space Ω. For notational simplicity we assume hidden states all have the same dimension, though our approach can easily be generalized.

3.1.1. TWO-DIMENSIONAL SPACE-TIME RNNS

To solve this problem, we view the proposed method as two recurrences, one in space and the other in time. We introduce a space variable k and consider the following recursion at any fixed time t:

$$\mathbf{h}_{i,t}^{(k)} = \phi\left(\mathbf{h}_{i,t}^{(k-1)}, \mathbf{h}_{i,t-1}, \mathbf{x}_{i,t}\right), k = [K], \mathbf{h}_{i,t}^{(0)} = \mathbf{h}_{i,t-1}. \tag{4}$$

We then have a second RNN in time by moving to the next timestep with $\mathbf{h}_{i,t} = \mathbf{h}_{i,t}^{(K)}$ and then repeating Eq. 4.

Nevertheless, in order to implement this approach we need to account for two issues. First, the space recursion as written may not converge. To deal with this we consider Newton's method and suitably modify our recursion in the sequel. Second, we need to transform the updates into a form that lends itself to updates using backpropagation. We discuss as well in the sequel.

3.2. Inexact Newton Method for Fixed-Point Iteration

3.2.1. ALGORITHM

Newton's method is a classic algorithm for solving the system of (nonlinear) equations, say $f(\mathbf{z}) = \mathbf{0}$, as follows:

$$\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} + \mathbf{s}^{(k)}, f'\left(\mathbf{z}^{(k)}\right)\mathbf{s}^{(k)} = -f\left(\mathbf{z}^{(k)}\right), \tag{5}$$

where f' denotes the gradient of function f. The large number of unknowns in the equation system and the nonexistence of feasible solutions, however, may lead to very expensive updates in Newton's method.

Inexact Newton methods, instead, refer to a family of algorithms that aims to solve the equation system $f(\mathbf{z}) = \mathbf{0}$ approximately at each iteration using the following rule:

$$\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} + \mathbf{s}^{(k)}, f'\left(\mathbf{z}^{(k)}\right)\mathbf{s}^{(k)} = -f\left(\mathbf{z}^{(k)}\right) + \mathbf{r}^{(k)}, \tag{6}$$

where $\mathbf{r}^{(k)}$ denotes the error at the k-th iteration between $f\left(\mathbf{z}^{(k)}\right)$ and $\mathbf{0}$. Such algorithms provide the potential of updating the solutions efficiently with (local) convergence guarantee under certain conditions (Dembo et al., 1982).

Inspired by inexact Newton methods and the Barzilai-Borwein method (Barzilai & Borwein, 1988), we propose a new inexact Newton method to solve $f(\mathbf{z}) = \mathbf{0}$ as follows:

$$\mathbf{z}^{(k+1)} = \mathbf{z}^{(k)} + \eta^{(k)}f\left(\mathbf{z}^{(k)}\right), \eta_t \in \mathbb{R}, \tag{7}$$

where we intentionally set $\mathbf{s}^{(k)} = \eta^{(k)}f\left(\mathbf{z}^{(k)}\right)$, $\mathbf{r}^{(k)} = \left[\mathbf{I} + \eta^{(k)}f'\left(\mathbf{z}^{(k)}\right)\right]f\left(\mathbf{z}^{(k)}\right)$ and $\mathbf{I}$ is an identity matrix.

3.2.2. CONVERGENCE ANALYSIS

We analyze the convergence of our proposed inexact Newton method in Eq. 7. We prove that under certain condition our method can converge locally with linear convergence rate.

Lemma 1 ((Dembo et al., 1982)). *Assume that $\frac{\|\mathbf{r}^{(k)}\|}{\|f(\mathbf{z}^{(k)})\|} \leq \tau < 1, \forall k$ where $\|\cdot\|$ denotes an arbitrary norm and the induced operator norm. There exists $\varepsilon > 0$ such that, if $\|\mathbf{z}^{(0)} - \mathbf{z}^*\| \leq \varepsilon$, then the sequence of inexact Newton iterates $\{\mathbf{z}^{(k)}\}$ converges to $\mathbf{z}^*$. Moreover, the convergence is linear in the sense that $\|\mathbf{z}^{(k+1)} - \mathbf{z}^*\|_* \leq \tau \|\mathbf{z}^{(k)} - \mathbf{z}^*\|_*$, where $\|\mathbf{y}\|_* = \|f'(\mathbf{z}^*)\mathbf{y}\|$.*

Theorem 1. *Assume that $\|\mathbf{I} + \eta^{(k)}f'\left(\mathbf{z}^{(k)}\right)\| < 1, \forall k$. There exists $\varepsilon > 0$ such that, if $\|\mathbf{z}^{(0)} - \mathbf{z}^*\| \leq \varepsilon$, then the sequence $\{\mathbf{z}^{(k)}\}$ generated using Eq. 7 converges to $\mathbf{z}^*$ locally with linear convergence rate.*

Proof. Based on Eq. 7 and the assumption, we have

$$\frac{\|\mathbf{r}^{(k)}\|}{\|f\left(\mathbf{z}^{(k)}\right)\|} \leq \frac{\|\mathbf{I} + \eta^{(k)}f'\left(\mathbf{z}^{(k)}\right)\|\|f\left(\mathbf{z}^{(k)}\right)\|}{\|f\left(\mathbf{z}^{(k)}\right)\|} < 1. \tag{8}$$
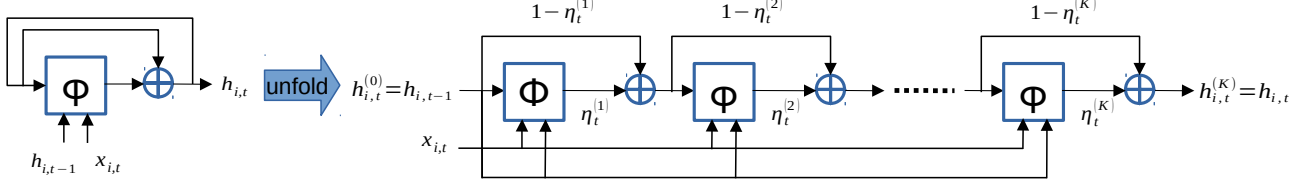
Figure 6. Illustration of networks for computing $\mathbf{h}_{i,t}$ using Eq. 9.

Further based on Lemma 1, we complete our proof. □

Discussion: The condition of $\left\|\mathbf{I} + \eta^{(k)} f'\left(\mathbf{z}^{(k)}\right)\right\| < 1, \forall k$ in Thm. 1 suggests that $\eta^{(k)}$ can be determined dependently (and probably differently) over the iterations. Also the linear convergence rate in Thm. 1 indicates that the top iterations in Eq. 7 are more important for convergence, as the difference between the current solution and the optimum decreases exponentially. This is the main reason that we can control the number of iterations in our algorithm (even just once) so that the convergence behavior can still be preserved.

Notice that if $\|\cdot\|$ in Thm. 1 denotes ℓ_2 norm, the condition of $\left\|\mathbf{I} + \eta^{(k)} f'\left(\mathbf{z}^{(k)}\right)\right\| < 1, \forall k$ essentially defines a lower and upper bounds for the eigenvalue of the matrix $\mathbf{I} + \eta^{(k)} f'\left(\mathbf{z}^{(k)}\right)$. As we show in our experiments later, $\eta^{(k)}$ usually is very small, indicating that the range of the spectrum of matrix $f'\left(\mathbf{z}^{(k)}\right)$ is allowed to be quite large. This observation significantly increases the probability of the condition being feasible in practice.

3.3. Approximating Fixed-Point Iteration with Residual Connections

Let us consider solving $f(\mathbf{h}_{i,t}) \equiv \phi\left(\mathbf{h}_{i,t}, \mathbf{h}_{i,t-1}, \mathbf{x}_{i,t}\right) - \mathbf{h}_{i,t} = \mathbf{0}$ based on Eq. 7. By substitute $f(\mathbf{h}_{i,t})$ into Eq. 7, we have the following update rule:

$$\mathbf{h}_{i,t}^{(k)} = \mathbf{h}_{i,t}^{(k-1)} + \eta_t^{(k)}\left[\phi\left(\mathbf{h}_{i,t}^{(k-1)}, \mathbf{h}_{i,t-1}, \mathbf{x}_{i,t}\right) - \mathbf{h}_{i,t}^{(k-1)}\right]$$
$$= \left(1 - \eta_t^{(k)}\right) \mathbf{h}_{i,t}^{(k-1)} + \eta_t^{(k)} \phi\left(\mathbf{h}_{i,t}^{(k-1)}, \mathbf{h}_{i,t-1}, \mathbf{x}_{i,t}\right). \tag{9}$$

This update rule can be efficiently implemented using residual connections in networks, as illustrated in Fig. 6, where $\oplus$ denotes the entry-wise plus operator, numbers associated with arrow lines denote the weights for linear combination, each blue box denotes a same sub-network for representing function ϕ that accounts for the residual.

During training, we learn the parameters in ϕ as well as all the η's for linear combination so that the learned features are good for supervision. To do so, we predefine the number of K for each time t, and concatenate such networks together

with the supervision signals. Then we can apply backpropagation to minimize the total loss, same as conventional feedforward neural networks.

3.4. Example: Linear Dynamical Systems and Beyond

Now let us consider the linear dynamic systems for modelling ϕ in Eq. 3, defined as follows:

$$\phi(\mathbf{h}_{i,t}, \mathbf{h}_{i,t-1}, \mathbf{x}_{i,t}) = \mathbf{U}\mathbf{h}_{i,t} + \mathbf{V}\mathbf{h}_{i,t-1} + \mathbf{W}\mathbf{x}_{i,t} + \mathbf{b}, \tag{10}$$

where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times n}, \mathbf{W} \in \mathbb{R}^{n \times d}, \mathbf{b} \in \mathbb{R}^n$ are the parameters that need to be learned. By plugging Eq. 10 into Fig. 6, we can compute $\mathbf{h}_{i,t}$, but at the cost of high computation.

Suppose that the matrix $(\mathbf{I} - \mathbf{U}) \in \mathbb{R}^{n \times n}$ is invertible. We then have a close-form fixed-point solution for Eq. 10, *i.e.* $\mathbf{h}_{i,t} = (\mathbf{I} - \mathbf{U})^{-1}\left(\mathbf{V}\mathbf{h}_{i,t-1} + \mathbf{W}\mathbf{x}_{i,t} + \mathbf{b}\right)$. Computing $(\mathbf{I} - \mathbf{U})^{-1}$ using neural networks is challenging. Instead we approximate it as $(\mathbf{I} - \mathbf{U})^{-1} = \sum_{k=0}^{\infty} \mathbf{U}^k \approx \mathbf{I} + \mathbf{U}$. Accordingly, we have an analytic solution for $\mathbf{h}_{i,t}$ as

$$\mathbf{h}_{i,t} = (\mathbf{I} + \mathbf{U})\left(\mathbf{V}\mathbf{h}_{i,t-1} + \mathbf{W}\mathbf{x}_{i,t} + \mathbf{b}\right), \tag{11}$$

which can be computed efficiently based on the network in Fig. 7(a). This linear recursion leads to a final hidden state for classification, which is a linear time invariant convolution of input timesteps. The discrimination of linear models for classification is very limited. To improve it, we propose a complex embedded nonlinear function for modelling ϕ based on linear dynamical systems that can be easily realized using the networks in Fig. 7(b), where σ denotes a nonlinear activation function such as $\tanh$ or ReLU. Mathematically this embedding networks for supervised learning aims to minimize (approximately) the following objective:

$$\min_{\mathbf{U},\mathbf{V},\mathbf{W},\mathbf{b},\omega \in \Omega} \sum_i \ell(\mathbf{h}_{i,T}, y_i; \omega), \tag{12}$$

$$\text{s.t. } \mathbf{g}_{i,t}^{(k)} = \mathbf{U}\mathbf{g}_{i,t}^{(k)} + \mathbf{V}\mathbf{h}_{i,t}^{(k-1)} + \mathbf{W}\mathbf{x}_{i,t} + \mathbf{b},$$
$$\mathbf{h}_{i,t}^{(k)} = \sigma\left(\mathbf{g}_{i,t}^{(k)}\right),$$
$$\mathbf{h}_{i,t}^{(0)} = \mathbf{h}_{i,t-1}, \mathbf{h}_{i,t} = \mathbf{h}_{i,t}^{(K)}, \forall i, \forall t \in [T].$$

The conditions above are essentially a special case of classic Jordan networks (Jordan, 1997).
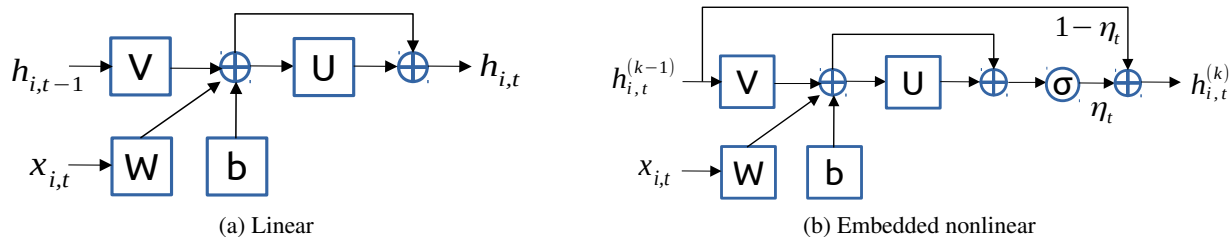
(a) Linear

(b) Embedded nonlinear

Figure 7. Illustration of networks for computing linear RNNs and the embedded nonlinear.

Discussion: From the perspective of autapse, here the matrix $\mathbf{U}$ can be considered to mimics the functionality of excitation and inhibition. From the perspective of learnable features, the matrix $(\mathbf{I} + \mathbf{U})$ is to perform the *alignment* in the hidden state space to reduce the variance among the features. Similar ideas have been explored in other domains. For instance, PointNet (Qi et al., 2017) is a powerful network for 3D point cloud classification and segmentation, where there exist so-called T-Nets (transformation networks) that transform input features to be robust to noise.

It is worth of mentioning that from the perspective of formulation, FastRNN (Kusupati et al., 2018a) can be considered as an approximate solver with $K = 1$ for the same minimization problem in Eq. 12 with a fixed $\mathbf{U} = \mathbf{0}$. Therefore, all the proofs in (Kusupati et al., 2018a) for FastRNN hold as well for ours in this special case.

Claim 1. *In passing, while we do not formally show this here, we point out that solving Eq. 12 using our inexact Newton method with Eq. 11 and $K = 1$, the bounds of our generalization error and convergence can be verified to be no larger than that of FastRNN using the same techniques as in that paper (Kusupati et al., 2018b).*

This claim is empirically validated in our experiments.

4. Experiments

To demonstrate our approach, we refer to ERNN in our experiments as the special one solving Eq. 12 using Fig. 7(b). By default we set $K = 1$ without explicit mention.

Datasets: ERNN's performance was benchmarked on the mix of IoT and traditional RNN tasks. IoT tasks include: (a) Google-30 (Warden, 2018) and Google-12, *i.e.* detection of utterances of 30 and 10 commands plus background noise and silence and (b) HAR-2 (Anguita et al., 2012) and DSA-19 (Altun et al., 2010), *i.e.* Human Activity Recognition (HAR) from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone and Daily and Sports Activity (DSA) detection from a resource-constrained IoT wearable device with 5 Xsens MTx sensors having accelerometers, gyroscopes and magnetometers on the torso and four limbs. Traditional RNN tasks includes tasks such as language mod-

Table 1. Dataset Statistics

Dataset	#Train	#Fts	#Steps	#Test
Google-12	22,246	3,168	99	3,081
Google-30	51,088	3,168	99	6,835
Yelp-5	500,000	38,400	300	500,000
HAR-2	7,352	1,152	128	2,947
Pixel-MNIST-10	60,000	784	784	10,000
PTB-10000	929,589	-	300	82,430
DSA-19	4,560	5,625	125	4,560

eling on the Penn Treebank (PTB) dataset (McAuley & Leskovec, 2013), star rating prediction on a scale of 1 to 5 of Yelp reviews (Yelp, 2017) and classification of MNIST images on a pixel-by-pixel sequence (Lecun et al., 1998).

All the datasets are publicly available and their pre-processing and feature extraction details are provided in (Kusupati et al., 2018a). The publicly provided training set for each dataset was subdivided into 80% for training and 20% for validation. Once the hyperparameters had been fixed, the algorithms were trained on the full training set and the results were reported on the publicly available test set. Table 1 lists the statistics of all the datasets.

Baseline Algorithms and Implementation: We compared ERNN with standard RNN, SpectralRNN (Zhang et al., 2018), EURNN (Jing et al., 2017), LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), UGRNN (Collins et al., 2016), FastRNN and FastGRNN-LSQ (*i.e.* FastGRNN without model compression but achieving better accuracy and lower training time) (Kusupati et al., 2018a). Since reducing model size is not our concern, we did not pursue model compression experiments and thus did not compare ERNN with FastGRNN directly, though potentially all the compression techniques in FastGRNN could be applicable to ERNN as well.

We used the publicly available implementation (Microsoft, 2018) for FastRNN and FastGRNN-LSQ. Except for FastRNN and FastGRNN-LSQ that we reproduced the results mentioned by verifying the hyper-parameter settings, we simply cited the corresponding numbers for the other competitors from (Kusupati et al., 2018a). All the experiments

Table 2. PTB Layer Modeling - 1 Layer

Algorithm	Test Perplexity	Model Size (KB)	Train Time (min)
FastRNN	127.76	513	11.20
FastGRNN-LSQ	**115.92**	513	12.53
RNN	144.71	**129**	9.11
SpectralRNN	130.20	242	-
LSTM	117.41	2052	13.52
UGRNN	119.71	256	11.12
ERNN	119.71	529	**7.11**

Table 3. Comparison of different RNNs on benchmark datasets

Dataset	Algorithm	Accuracy (%)	Model Size (KB)	Train Time (hr)
HAR-2	FastRNN	94.50	29	0.063
	FastGRNN-LSQ	95.38	29	0.081
	RNN	91.31	29	0.114
	SpectralRNN	95.48	525	0.730
	EURNN	93.11	**12**	0.740
	LSTM	93.65	74	0.183
	GRU	93.62	71	0.130
	UGRNN	94.53	37	0.120
	ERNN	**95.59**	34	**0.061**
DSA-19	FastRNN	84.14	97	0.032
	FastGRNN-LSQ	85.00	208	0.036
	RNN	71.68	**20**	0.019
	SpectralRNN	80.37	50	0.038
	LSTM	84.84	526	0.043
	GRU	84.84	270	0.039
	UGRNN	84.74	399	0.039
	ERNN	**86.87**	36	**0.015**
Google-12	FastRNN	92.21	56	0.61
	FastGRNN-LSQ	93.18	57	0.63
	RNN	73.25	**56**	1.11
	SpectralRNN	91.59	228	19.0
	EURNN	76.79	210	120.00
	LSTM	92.30	212	1.36
	GRU	93.15	248	1.23
	UGRNN	92.63	75	0.78
	ERNN	**94.96**	66	**0.20**
Google-30	FastRNN	91.60	96	1.30
	FastGRNN-LSQ	92.03	**45**	1.41
	RNN	80.05	63	2.13
	SpectralRNN	88.73	128	11.0
	EURNN	56.35	135	19.00
	LSTM	90.31	219	2.63
	GRU	91.41	257	2.70
	UGRNN	90.54	260	2.11
	ERNN	**94.10**	70	**0.44**
Pixel-MNIST	FastRNN	96.44	166	15.10
	FastGRNN-LSQ	**98.72**	71	12.57
	EURNN	95.38	**64**	122.00
	RNN	94.10	71	45.56
	LSTM	97.81	265	26.57
	GRU	98.70	123	23.67
	UGRNN	97.29	84	15.17
	ERNN	98.13	80	**2.17**
Yelp-5	FastRNN	55.38	130	3.61
	FastGRNN-LSQ	**59.51**	130	3.91
	RNN	47.59	130	3.33
	SpectralRNN	56.56	**89**	4.92
	EURNN	59.01	122	72.00
	LSTM	59.49	516	8.61
	GRU	59.02	388	8.12
	UGRNN	58.67	258	4.34
	ERNN	57.21	138	**0.69**

were run on a Nvidia GTX 1080 GPU with CUDA 9 and cuDNN 7.0 on a machine with Intel Xeon 2.60 GHz GPU with 20 cores. We found that FastRNN and FastGRNN-LSQ can be trained to perform similar accuracy as reported in (Kusupati et al., 2018a) using slightly longer training time on our machine. This indicates that potentially all the other competitors can achieve similar accuracy using longer training time as well.

Hyper-parameters: The hyper-parameters of each algorithm were set by a fine-grained validation wherever possible or according to the settings published in (Kusupati et al., 2018a) otherwise. Both the learning rate and η's were initialized to 10^{-2}. Since ERNN converged much faster in comparison to FastRNN and FastGRNN-LSQ, the learning rate was halved periodically, where the period was learnt based on the validation set. Replicating this on FastRNN or FastGRNN-LSQ does not achieve the maximum accuracy reported in the paper. The batch size of 128 seems to work well across all the datasets. ERNN used ReLU as the non-linearity and Adam (Kingma & Ba, 2015) as the optimizer for all the experiments.

Evaluation Criteria: The primary focus in this paper is on achieving better results than state-of-the-art RNNs with much better convergence rate. For this purpose we reported model size, training time and accuracy (perplexity on the PTB dataset). Following the lines of FastRNN and FastGRNN-LSQ, for PTB and Yelp datasets, model size excludes the word-vector embedding storage.

Results: Table 2 and Table 3 compare the performance of ERNN to the state-of-the-art RNNs. Four points are worth noticing about ERNN's performance. First, ERNN's prediction gains over a standard RNN ranged from 3.16% on Pixel-MNIST dataset to 21.71% on Google-12 dataset. Similar observations are made for other previously wide-used RNNs as well, demonstrating the superiority of our approach. Second, ERNN's prediction accuracy always surpassed FastRNN's prediction accuracy. This indeed shows the advantage of learning a general **U** matrix rather than fixing it to **0**. Third, ERNN could surpass gating based FastGRNN-LSQ on 4 out of 6 dataset in terms of prediction accuracy with 2.87% on DSA-19 dataset and 2.7% on Google-30 dataset. Fourth, and most importantly, ERNN's

training speedups over FastRNN as well as FastGRNN-LSQ could range from 1.3x on HAR-2 dataset to 6x on Pixel-MNIST dataset. This emphasizes the fact that self-feedback can help ERNN achieve better results than gating backed methods with significantly better training efficiency. Note that the model size of our ERNN is always comparable to,

Table 4. Comparison on different ERNNs on HAR-2 dataset

Algorithm	Accuracy (%)	Model Size (KB)	Train Time (hr)
FastRNN	94.50	**29**	0.063
FastGRNN-LSQ	95.38	**29**	0.081
RNN	91.31	**29**	0.114
LSTM	93.65	74	0.183
ERNN(K=1)	95.59	34	**0.061**
ERNN(K=2)	**96.33**	35	0.083
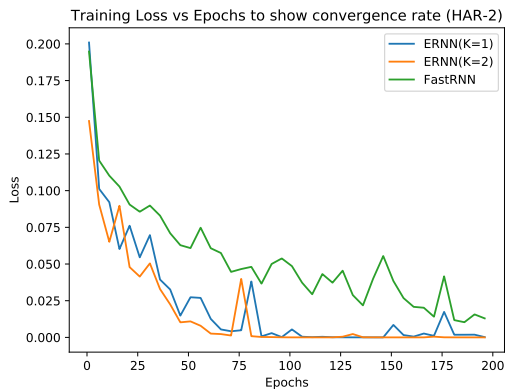
Figure 8. Comparison on convergence of different approaches.

(a) $K = 1$ on HAR-2   (b) $K = 1$ on DSA-19

(c) $k = 1, K = 2$ on HAR-2   (d) $k = 2, K = 2$ on HAR-2

Figure 9. Illustration of learned η's over timesteps.

or even better than, the model size of either FastRNN or FastGRNN-LSQ.

Table 4 compares the performance of ERNN with K=1 to that of K=2 (see Fig. 6 for the definition of K) on HAR-2 dataset. It can be seen that ERNN(K=2) achieves nearly 1% higher prediction accuracy with almost no change in model size but slightly higher training time. In comparison to FastGRNN-LSQ, ERRN(K=2) achieves 1% higher prediction accuracy with similar training time.

To further verify the advantage of our approach on convergence, we show the training behavior of different approaches in Fig. 8. As we see, our ERNNs converge significantly faster than FastRNN while achieving lower losses. This observation demonstrates the importance of locating equilibrium points for dynamical systems. Meanwhile, the curve for ERNN(K=2) tends to stay below the curve for ERNN(K=1), indicating that finer approximate solutions for equilibrium points lead to faster convergence as well as better generalization (see Table 2). The training time reported in Table 2, Table 3, and Table 4 is the one for achieving a convergent model with best accuracy.

To understand the effect of learnable weights η's, we show these learned values in Fig. 9, where we fit red lines to the scatter plots of these (η, T) pairs based on least squares. Overall all the numbers here are very small, which is very useful to make the condition in Thm. 1 feasible in practice. In general, we observe similar decreasing behavior to that reported in (Kusupati et al., 2018a). In contrast, our learning

procedure does not constrain η to be positive, and thus we can learn some negatives to better fit the supervision signals, as illustrated in Fig. 9(a). Across different datasets, η's form different patterns. For the case of $K = 2$ on the same dataset, however, we observe that the patterns of learned η's are almost identical with slightly change in values. This indicates that we may just need to learn a single η for different k's. We will investigate more on the impact of η's on accuracy in our future work.

5. Conclusion

Motivated by autapse in neuroscience, we propose a novel *Equilibrated Recurrent Neural Network (ERNN)* method. We introduce neuronal time-delayed self-feedback into conventional recurrent models, which leads to better generalization as well as efficient training. We demonstrate empirically that such neuronal self-feedback helps stabilize the hidden state transition matrices rapidly by learning discriminative latent features, leading to fast convergence in training and good accuracy in testing. To locate fixed points efficiently, we propose a novel inexact Newton method that can be proven to converge locally with linear rate (under mild conditions). As a result we can recast ERNN training into networks with residual connections in a principled way and train them efficiently based on backpropagation. We demonstrate the superiority of ERNNs over the state-of-the-art on several benchmark datasets in terms of both accuracy and training efficiency. For instance, on the Google-30 dataset our ERNN outperforms FastRNN by 2.50% in accuracy with $\sim 3\times$ faster training speed.

References

Altun, K., Barshan, B., and Tunçel, O. Comparative study on classifying human activities with miniature inertial and magnetic sensors. *Pattern Recogn.*, 43(10):3605–3620, October 2010. ISSN 0031-3203. doi: 10.1016/j.patcog.2010.04.019. URL http://dx.doi.org/10.1016/j.patcog.2010.04.019.

Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proceedings of the 4th International Conference on Ambient Assisted Living and Home Care*, IWAAL'12, pp. 216–223, Berlin, Heidelberg, 2012. Springer-Verlag. ISBN 978-3-642-35394-9. doi: 10.1007/978-3-642-35395-6_30. URL http://dx.doi.org/10.1007/978-3-642-35395-6_30.

Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

Barabanov, N. E. and Prokhorov, D. V. Stability analysis of discrete-time recurrent neural networks. *Trans. Neur. Netw.*, 13(2):292–303, March 2002. ISSN 1045-9227. doi: 10.1109/72.991416. URL https://doi.org/10.1109/72.991416.

Barzilai, J. and Borwein, J. M. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.

Bradbury, J., Merity, S., Xiong, C., and Socher, R. Quasi-recurrent neural networks. *arXiv preprint arXiv:1611.01576*, 2016.

Campos, V., Jou, B., Giró-i Nieto, X., Torres, J., and Chang, S.-F. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834*, 2017.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. A., and Huang, T. S. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 77–87, 2017.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Cho, M., Dhir, C., and Lee, J. Hessian-free optimization for learning deep multidimensional recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 883–891, 2015.

Collins, J., Sohl-Dickstein, J., and Sussillo, D. Capacity and Trainability in Recurrent Neural Networks. *arXiv e-prints*, art. arXiv:1611.09913, November 2016.

Dauphin, Y., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. In *ICML*, 2017.

Dembo, R. S., Eisenstat, S. C., and Steihaug, T. Inexact newton methods. *SIAM Journal on Numerical analysis*, 19(2):400–408, 1982.

Fan, H., Wang, Y., Wang, H., Lai, Y.-C., and Wang, X. Autapses promote synchronization in neuronal networks. *Scientific reports*, 8(1):580, 2018.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. Convolutional sequence to sequence learning. In *ICML*, 2017.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., and Graves, A. Memory-efficient backpropagation through time. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4125–4133. 2016.

Herrmann, C. S. and Klaus, A. Autapse turns neuron into oscillator. *International Journal of Bifurcation and Chaos*, 14(02):623–633, 2004.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hori, C., Hori, T., Lee, T.-Y., Zhang, Z., Harsham, B., Hershey, J. R., Marks, T. K., and Sumi, K. Attention-based multimodal fusion for video description. In *ICCV*, pp. 4203–4212, 2017.

Jaeger, H. *Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the" echo state network" approach*, volume 5. 2002.

Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., Tegmark, M., and Soljačić, M. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *International Conference on Machine Learning*, pp. 1733–1741, 2017.

Jing, L., Shen, Y., Dubček, T., Peurifoy, J., Skirlo, S., LeCun, Y., Tegmark, M., and Soljačić, M. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *ICML*, 2017.

Jordan, M. I. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pp. 471–495. Elsevier, 1997.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICML*, 2015.

Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., and Varma, M. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems 31*, pp. 9031–9042. 2018a.

Kusupati, A., Singh, M., Bhatia, K., Kumar, A., Jain, P., and Varma, M. Fastgrnn: A fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In *Advances in Neural Information Processing Systems*, 2018b.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324, 1998.

Lei, T., Zhang, Y., and Artzi, Y. Training rnns as fast as cnns. *arXiv preprint arXiv:1709.02755*, 2017.

Liao, R., Xiong, Y., Fetaya, E., Zhang, L., Yoon, K., Pitkow, X., Urtasun, R., and Zemel, R. Reviving and improving recurrent back-propagation. *arXiv preprint arXiv:1803.06396*, 2018.

MacKay, M., Vicol, P., Ba, J., and Grosse, R. B. Reversible recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 9043–9054, 2018.

Martens, J. and Sutskever, I. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033–1040, 2011.

Martin, E. and Cundy, C. Parallelizing linear recurrent neural nets over sequence length. In *International Conference on Learning Representations*, 2018.

McAuley, J. and Leskovec, J. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pp. 165–172, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2409-0. doi: 10.1145/2507157.2507163. URL http://doi.acm.org/10.1145/2507157.2507163.

Microsoft. Edge machine learning. 2018. URL https://github.com/Microsoft/EdgeML.

Miller, J. and Hardt, M. When recurrent models don't need to be recurrent. *arXiv preprint arXiv:1805.10369*, 2018.

Mujika, A., Meier, F., and Steger, A. Approximating real-time recurrent learning with random kronecker factors. In *Advances in Neural Information Processing Systems 31*, pp. 6594–6603. 2018.

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pp. 652–660, 2017.

Qin, H., Wu, Y., Wang, C., and Ma, J. Emitting waves from defects in network with autapses. *Communications in Nonlinear Science and Numerical Simulation*, 23(1-3):164–174, 2015.

Seung, H. S., Lee, D. D., Reis, B. Y., and Tank, D. W. The autapse: a simple illustration of short-term analog memory storage by tuned synaptic feedback. *Journal of computational neuroscience*, 9(2):171–185, 2000.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Tallec, C. and Ollivier, Y. Unbiased online recurrent optimization. *arXiv preprint arXiv:1702.05043*, 2017.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. In *SSW*, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *NIPS*, 2017.

Warden, P. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *arXiv e-prints*, art. arXiv:1804.03209, April 2018.

Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1 (2):270–280, 1989.

Yelp, I. Yelp dataset challenge. 2017. URL https://www.yelp.com/dataset/challenge.

Zhang, J., Lei, Q., and Dhillon, I. S. Stabilizing gradients for deep neural networks via efficient svd parameterization. In *ICML*, 2018.