FastGRNN: A Fast, Accurate, Stable and Tiny Kilobyte Sized Gated Recurrent Neural Network

Aditya Kusupati[†], **Manish Singh**[§], **Kush Bhatia**[‡],
Ashish Kumar[‡], **Prateek Jain**[†] **and Manik Varma**[†]
[†]Microsoft Research India
[§]Indian Institute of Technology Delhi
[‡]University of California Berkeley
{t-vekusu,prajain,manik}@microsoft.com, singhmanishiitd@gmail.com
kush@cs.berkeley.edu, ashish_kumar@berkeley.edu

Abstract

This paper develops the FastRNN and FastGRNN algorithms to address the twin RNN limitations of inaccurate training and inefficient prediction. Previous approaches have improved accuracy at the expense of increased prediction costs making them infeasible for resource-constrained and real-time applications. Unitary RNNs have increased accuracy somewhat by restricting the range of the state transition matrix's singular values but have also increased the model size as they required a larger number of hidden units to make up for the loss in expressive power. Gated RNNs have obtained state-of-the-art accuracies by adding extra parameters thereby resulting in even larger models. FastRNN addresses these limitations by developing a leaky integrator unit inspired peephole connection that does not constrain the range of the singular values explicitly and has only two extra scalar parameters. FastGRNN then extends the peephole to a gated architecture by reusing the RNN matrices in the gate to match state-of-the-art accuracies but with a 2-4x smaller model as compared to other gated architectures and with almost no overheads over a standard RNN. Further compression could be achieved by allowing FastGRNN's matrices to be low-rank, sparse and quantized without a significant loss in accuracy. Experiments on multiple benchmark datasets revealed that FastGRNN could make more accurate predictions with up to a 35x smaller model as compared to leading unitary and gated RNN techniques. FastGRNN's code can be publicly downloaded from [41].

1 Introduction

Objective: This paper develops the FastGRNN (an acronym for a Fast, Accurate, Stable and Tiny Gated Recurrent Neural Network) algorithm to address the twin RNN limitations of inaccurate training and inefficient prediction. FastGRNN almost matches the accuracies and training times of state-of-the-art unitary and gated RNNs but has significantly lower prediction costs with models ranging from 1 to 6 Kilobytes for real-world applications.

RNN training and prediction: It is well recognized that RNN training is inaccurate and unstable as non-unitary hidden state transition matrices could lead to exploding and vanishing gradients for long time series. An equally important concern for resource-constrained and real-time applications is the RNN's model size and prediction time. Squeezing the RNN model and code into a few Kilobytes could allow RNNs to be deployed on billions of Internet of Things (IoT) microcontrollers having just 2 KB RAM and 32 KB flash memory [18, 30]. Similarly, squeezing the RNN model and code into a few Kilobytes of the 32 KB L1 cache of a Raspberry Pi or smartphone, could significantly reduce the

Preprint. Work in progress.

prediction time and energy and make RNNs feasible for real-time applications such as wake word detection [28, 12, 13, 43, 47], predictive maintenance [46, 2], human activity recognition [4, 3], *etc*.

Unitary and gated RNNs: A number of techniques have been proposed to address some of these limitations based on improved training algorithms[40, 27], unitary RNNs [6, 23, 37, 48, 50, 52] and gated RNNs [20, 14, 15]. While such approaches have stabilized RNN training they have often done so at the cost of increased model size. Unitary RNNs have avoided gradients exploding and vanishing by limiting the range of the singular values of the hidden state transition matrix. This has led to only limited gains in prediction accuracy as the optimal transition matrix might often not be close to unitary. Unitary RNNs have compensated by learning higher dimensional representations but, unfortunately, this has led to larger model sizes. Gated RNNs [20, 14, 15] have stabilized training by adding extra parameters leading to state-of-the-art prediction accuracies but with models that might sometimes be even larger than unitary RNNs.

FastRNN and FastGRNN: This paper demonstrates that standard RNN training could be stabilized with the addition of a peephole connection [22] having just 2 additional scalar parameters. The RNN peephole architecture was first proposed by [22] as the Leaky Integrator Unit (LIU) and was further studied by [10]. By learning the extent of "leak", this paper demonstrates that an LIU inspired architecture, referred to as FastRNN, had lower training and prediction costs as compared to all unitary and gated RNNs. Furthermore, FastRNN's prediction accuracies could be: (a) up to 19% higher than a standard RNN; (b) could often surpass the accuracies of all existing unitary RNNs and (c) could be just shy of the accuracies of leading gated RNNs. FastRNN's empirical performance could be understood on the basis of theorems proving that for a time series with T steps: (a) FastRNN converges to a stationary point within $O(T^3)$ SGD iterations while the *same analysis* for a standard RNN reveals an upper bound of $O(2^T)$ iterations and (b) FastRNN's generalization error bound is linear in T whereas the *same proof technique* reveals an exponential bound for standard RNNs

Inspired by this analysis, the novel FastGRNN architecture is developed by converting the peephole connection to a gate while reusing the RNN matrices. FastGRNN, therefore, has 2-4x fewer parameters than other gated RNNs. FastGRNN could match the accuracies and training time of leading gated RNNs but at significantly lower prediction costs. Enforcing that FastGRNN's matrices be low-rank, sparse and quantized led to a minor increase in the training time and classification error but resulted in models that could be up to 35x smaller and fit in 1-6 Kilobytes for non-NLP applications.

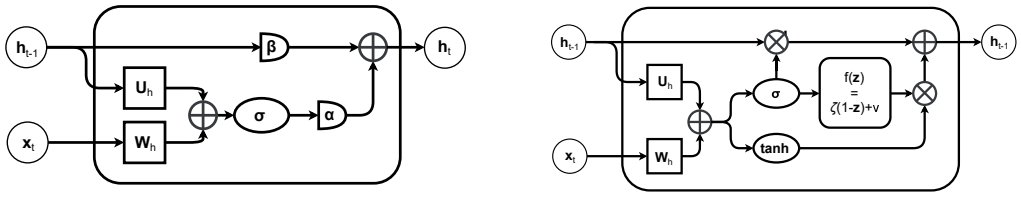
Contributions: This paper makes two contributions. First, it rigorously studies an LIU based architecture, FastRNN, which could be faster at training and prediction than all other RNNs while obtaining near state-of-the-art accuracies. Second, inspired by FastRNN, it develops a novel gated FastGRNN architecture which could almost match state-of-the-art accuracies and training times but with prediction costs that could be lower by an order of magnitude. FastRNN and FastGRNN's code can be publicly downloaded from [41].

Notation: Throughout the paper, $\{(\mathbf{X}_i, y_i)_{i=1}^N\}$ denote the training set, where each $\mathbf{X}_i \in \mathbb{R}^{T \times D}$ and $y^i \in \{1, 2, \ldots, L\}$. Each $\mathbf{X}_i = [\mathbf{x}_{i1}, \ldots, \mathbf{x}_{iT}]^\top$ is composed of T timesteps of data $\mathbf{x}_{it} \in \mathbb{R}^D$. $\mathbf{h}_t \in \mathbb{R}^{\hat{D}}$ denotes the hidden state of a recurrent architecture at time t. Parameters of a recurrent neural network are denoted by matrices $\mathbf{W}$, $\mathbf{U}$ and bias vectors by $\mathbf{b}$, often using subscripts if multiple matrices are required to specify the architecture. $\mathbf{a} \odot \mathbf{b}$ denotes the Hadamard product between $\mathbf{a}$ and $\mathbf{b}$, i.e., $(\mathbf{a} \odot \mathbf{b})_i = \mathbf{a}_i, \mathbf{b}_i$. $\|\cdot\|_0$ denotes the number of non-zeros entries in a matrix or vector.

2 Related Work

Unitary RNNs: Unitary RNNs [6, 50, 37, 23, 48, 26] stabilize RNN training by restricting the class of parameter matrices to a subset of well-conditioned matrices, which limits the expressive power of these architectures. Consequently, these methods have less classification accuracy than FastRNN and are also slower to train. The recently proposed SpectralRNN [52] allows the parameter matrix to be *any* matrix with singular values in $1 \pm \epsilon$. Unfortunately, the training algorithm converged only for small ϵ for most datasets, leading to inaccurate models. To improve accuracy, SpectralRNN required models to have a large number of hidden units, thereby increasing the training time, prediction time and the model size. For instance, on the Google-30 dataset, SpectralRNN was 3.3% less accurate despite using 3 times more hidden units and training for 8x longer as compared to FastGRNN.

Gated RNNs: Gated architectures achieve state-of-the-art classification accuracies by adding extra parameters and gates but also increases model size and prediction time when compared to RNN.



(a) FastRNN Peephole Connection

(b) FastGRNN Gate

Figure 1: Block Diagrams for FastRNN (left plot) and FastGRNN (right plot). FastGRNN uses shared matrices $\mathbf{W}_h$, $\mathbf{U}_h$ to compute both the hidden state $\mathbf{h}_t$ as well as gate $\mathbf{z}$ for peephole connection.

Therefore, there has been a trend to simplify gated architectures: UGRNN [15] simplify GRU [14] which in turn simplify LSTM [20]. GORU [24] addresses the issue of forgetting irrelevant information in Unitary RNNs with the use additional gated structure similar to GRUs. Unlike UGRNN and GORU, FastGRNN reuses parameter matrices in the gates and adds a degree of freedom by using scalars in the gating function to increase expressiveness of the architecture. The Leaky Integrator Unit (LIU) [22] has the same architecture as FastRNN, but it does not learn the RNN transition matrices, instead samples them from a prior hand-crafted distribution. In contrast, FastRNN learns RNN parameters as well as extent of "leak". As a result, FastRNN could be upto 34.41% better than LIU in classification accuracy. Note that this paper uses FastRNN as a benchmark to study gradient stabilization and convergence, and based on these ideas, develops the novel FastGRNN architecture.

Efficient Prediction: Low-rank and sparsity of parameters have long been exploited to obtain efficient prediction algorithms with better generalization performance. Most unitary methods effectively utilize a low rank representation of parameter matrices to control prediction and training complexity [23, 52]. Sparsity, low-rank, and quantization were shown to be effective in RNNs [51, 39], CNN architectures [19], trees [30] and nearest neighbor classifiers [18]. FastGRNN builds on these ideas to utilize low-rank, sparse and quantized representations for learning kilobyte sized classifiers without compromising on classification accuracy. Finally, [11] proposed SkipRNN which updated the state vector $\mathbf{h}_t$ intermittently to improving prediction time accuracy. Such techniques are complementary to the ones considered in this paper and can further improve FastGRNN's performance.

3 Method

Standard Recurrent Neural Network [42] architecture is known to be *unstable* and faces issues with exploding or vanishing gradient and hence is shunned for more expensive gated architectures.

This paper studies the FastRNN architecture that is inspired by leaky units [22], and shows that FastRNN can be significantly more stable and accurate than standard RNN without increasing the prediction complexity. Section 3.1.1 shows that appropriate parameter settings in FastRNN leads to stable gradients as well as significantly faster convergence rate and generalization error than standard RNN. Motivated by the analysis and success of FastRNN, this paper further strengthens it to develop the FastGRNN architecture that is more accurate than unitary methods [6, 52], while providing comparable accuracy to GRU/LSTM at 45x less computational cost (see Table 3).

3.1 FastRNN

The Recurrent Neural Network architecture proposed by [42] is a predictive model for sequential data. Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$ be the input data where $\mathbf{x}_t \in \mathbb{R}^D$ denotes the t-th step feature vector. Then, goal of multi-class RNNs is to learn a function $F : \mathbb{R}^{D \times T} \to [L]$ that predicts one of L classes for the given datapoint $\mathbf{X}$. Standard RNN has a provision to produce an output at every time step, but we focus on the setting where each data point is associated with a single label that is predicted at the end of the time horizon T. Standard RNN maintains a vector of hidden state $\mathbf{h}_t \in \mathbb{R}^{\hat{D}}$ which captures the long-term dependencies of the input sequence, i.e.,

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h). \tag{1}$$

As explained in next section, learning $\mathbf{U}, \mathbf{W}$ in the above architecture is difficult as the gradient can have exponentially large (in T) condition number. Unitary methods attempt to explicitly control the condition number but end up losing significantly on accuracy and/or training time.

Instead, FastRNN uses a simple peephole connection to stabilize the gradients and the training significantly. In particular, FastRNN updates the hidden state $\mathbf{h}_t$ as follows:

$$\tilde{\mathbf{h}}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}_h), \tag{2}$$

$$\mathbf{h}_t = \alpha\tilde{\mathbf{h}}_t + \beta\mathbf{h}_{t-1}, \tag{3}$$

where $\alpha > 0, \beta > 0$ are trainable parameters. $\sigma : \mathbb{R} \to \mathbb{R}$ is a non-linear function such as $\tanh$, sigmoid, or ReLU, and can vary across datasets. Given $\mathbf{h}_T$, the label for a given point $\mathbf{X}$ is predicted by applying a standard classifier, e.g., logistic regression to $\mathbf{h}_T$.

Typically, $\beta \approx 1 - \alpha$ and $\alpha \ll 1$. FastRNN updates hidden state in a controlled manner where α, β control the extent to which the current feature vector $\mathbf{x}_t$ updates the hidden state. Also, FastRNN has only 2 more parameters than RNN and require only $\hat{D}$ more computations, which is a tiny fraction of per-step computation complexity of RNN. Unlike unitary methods [6, 52], FastRNN does not introduce expensive structural constraints on $\mathbf{U}$ and hence scales well to large datasets with standard optimization techniques [1, 29]. Finally, while FastRNN architecture is the same as LIU architecture [22], FastRNN learns both the parameter matrices as well as α, β unlike the models of [22, 10].

3.1.1 Analysis

This section shows how FastRNN addresses the issue of vanishing/exploding gradients, leading to stable training and lower generalization error. For simplicity, assume that the label decision function is one dimensional and is given by $f(\mathbf{X}) = \mathbf{v}^\top \mathbf{h}_T$ and let $L(\mathbf{X}, y; \boldsymbol{\theta}) = L(f(\mathbf{X}), y; \boldsymbol{\theta})$ be the loss function for given labeled data point $(\mathbf{X}, y)$ and parameters $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{U}, \mathbf{v})$. Then, the gradient of L w.r.t. $\mathbf{W}, \mathbf{U}, \mathbf{v}$ is given by:

$$\frac{\partial L}{\partial \mathbf{U}} = \alpha \sum_{t=0}^{T} \mathbf{D}_t \left(\prod_{k=t}^{T-1} (\alpha \mathbf{U}^\top \mathbf{D}_{k+1} + \beta \mathbf{I}) \right) (\nabla_{\mathbf{h}_T} L) \mathbf{h}_{t-1}^\top, \tag{4}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \alpha \sum_{t=0}^{T} \mathbf{D}_t \left(\prod_{k=t}^{T-1} (\alpha \mathbf{U}^\top \mathbf{D}_{k+1} + \beta \mathbf{I}) \right) (\nabla_{\mathbf{h}_T} L) \mathbf{x}_t^\top, \frac{\partial L}{\partial \mathbf{v}} = \frac{(-\mathbf{v}^\top \mathbf{h}_T) y \exp(-\mathbf{v}^\top \mathbf{h}_T)}{1 + \exp(-\mathbf{v}^\top \mathbf{h}_T)} \mathbf{h}_T, \tag{5}$$

where $\nabla_{\mathbf{h}_T} L = c(\boldsymbol{\theta})(-\mathbf{v}^\top \mathbf{h}_T)\mathbf{v}$, with $c(\boldsymbol{\theta}) = \frac{1}{1+\exp(\mathbf{v}^\top \mathbf{h}_T)}$. Note that the critical term in above gradient is: $M(\mathbf{U}) = \prod_{k=t}^{T-1} (\alpha \mathbf{U}^\top \mathbf{D}_{k+1} + \beta \mathbf{I})$, whose condition number can be as large as:

$$\kappa(M(\mathbf{U})) \leq \frac{(1 + \frac{\alpha}{\beta} \max_k \|\mathbf{U}^\top \mathbf{D}_{k+1}\|)^{T-t}}{(1 - \frac{\alpha}{\beta} \max_k \|\mathbf{U}^\top \mathbf{D}_{k+1}\|)^{T-t}}.$$

where $\mathbf{D}_k = \text{diag}(\sigma'(\mathbf{W}\mathbf{x}_k + \mathbf{U}\mathbf{h}_{k-1} + \mathbf{b}_h))$ is the Jacobian matrix of the pointwise nonlinearity. Also if $\alpha = 1$ and $\beta = 0$, which corresponds to standard RNN, the condition number of $M(\mathbf{U})$ can be as large as $(\max_k \frac{\|\mathbf{U}^\top \mathbf{D}_{k+1}\|}{\lambda_{min}(\mathbf{U}^\top \mathbf{D}_{k+1})})^{T-t}$ where $\lambda_{min}(\mathbf{A})$ denotes the minimum singular value of $\mathbf{A}$. That is the condition number for standard RNN can be exponential in T which implies that gradient can be nearly 0 in certain directions while in certain other directions it can be exponentially large.

In comparison, if $\beta \approx 1$ and $\alpha \approx 0$, then the condition number for FastRNN is bounded by a small term. For example, if $\beta = 1 - \alpha$ and $\alpha = \frac{1}{T \max_k \|\mathbf{U}^T \mathbf{D}_{k+1}\|}$, then $\kappa(M(\mathbf{U})) = O(1)$. Existing unitary methods are also motivated by similar observation. But they attempt to control the condition number of $M(\mathbf{U})$ by restricting the condition number of $\mathbf{U}$ which can still lead to vanishing gradient in certain directions as $\mathbf{U}^T \mathbf{D}_{k+1}$ might still be very small in certain directions. By using peephole connection, FastRNN is able to address this issue, and hence have faster training and more accurate model than state-of-the-art unitary methods.

Finally, by using the above observations and a careful perturbation analysis, we can provide the following convergence and generalization bounds for FastRNN:

Theorem 3.1 (Convergence Bound)**.** *Let $[(\mathbf{X}_1, y_1), \ldots, (\mathbf{X}_n, y_n)]$ be the given labeled sequential training data. Let $L(\boldsymbol{\theta}) = \frac{1}{n} \sum_i L(\mathbf{X}_i, y_i; \boldsymbol{\theta})$ be the loss function with $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{U}, \mathbf{v})$ be the parameters of FastRNN architecture (3). Then, randomized stochastic gradient descent [16], a minor variation of SGD, when applied to the data for a maximum of N iteration outputs a solution $\widehat{\boldsymbol{\theta}}$ such that:*

$$\mathbb{E}[\|\nabla_{\boldsymbol{\theta}} L(\widehat{\boldsymbol{\theta}})\|_2^2\|] \leq \mathcal{B}_N := \frac{O(T^2) L(\boldsymbol{\theta}_0)}{N} + \left(\bar{D} + \frac{4 R_{\mathbf{W}} R_{\mathbf{U}} R_{\mathbf{v}}}{\bar{D}} \right) \frac{O(T^3)}{\sqrt{N}},$$

where $R_{\mathbf{X}} = \max_{\mathbf{X}} \|\mathbf{X}\|_F$ *for* $\mathbf{X} = \{\mathbf{U}, \mathbf{W}, \mathbf{v}\}$, $L(\boldsymbol{\theta}_0)$ *is the loss of the initial classifier, and the step-size of the* k*-th SGD iteration is fixed as:* $\gamma_k = \min\left\{\frac{1}{T^2}, \frac{\bar{D}}{T\sqrt{N}}\right\}, k \in [N], \ \ \bar{D} \geq 0.$

Theorem 3.2 (Generalization Error Bound)**.** *[8] Let* $\mathcal{Y}, \hat{\mathcal{Y}} \subseteq [0, 1]$ *and let* $\mathcal{F}_T$ *denote the class of FastRNN with* $\|\mathbf{U}\|_F \leq R_{\mathbf{U}}, \|\mathbf{W}\|_F \leq R_{\mathbf{W}}$*. Let the final classifier be given by* $\sigma(\mathbf{v}^\top \mathbf{h}_T)$*,* $\|\mathbf{v}\|_2 \leq R_{\mathbf{v}}$ *. Let* $L : \mathcal{Y} \times \hat{\mathcal{Y}} \to [0, B]$ *be any* 1*-Lipschitz loss function. Let* D *be any distribution on* $\mathcal{X} \times \mathcal{Y}$ *such that* $\|\mathbf{x}_{it}\|_2 \leq R_x$ *a.s. Let* $0 \leq \delta \leq 1$*. Then with probability atleast* $(1 - \delta)$ *for* $\alpha \leq \frac{1}{2(2R_{\mathbf{U}}-1)T}$*, all functions* $f \in \mathbf{v} \circ \mathcal{F}_T$ *satisfy*

$$\mathbb{E}_D[L(f(\mathbf{X}), y)] \leq \frac{1}{n}\sum_{i=1}^{n} L(f(\mathbf{X}_i), y_i) + \mathcal{C}\frac{O(\alpha T)}{\sqrt{n}} + B\sqrt{\frac{\ln(\frac{1}{\delta})}{n}},$$

where $\mathcal{C} = R_{\mathbf{W}} R_{\mathbf{U}} R_{\mathbf{x}} R_{\mathbf{v}}$ *represents the boundedness of the parameter matrices and the data.*

The convergence bound states that if $\alpha = O(1/T)$ then the algorithm converges to a stationary point in polynomial time (in T and N), while generalization bound states that for $\alpha = O(1/T)$, the generalization error of FastRNN scales polynomially with T. In contrast, *similar proof technique* provide exponentially poor (in T) error bound and convergence rate for standard RNN. We would like to stress that we are comparing upper bounds and hence potentially one can obtain significantly better error bounds for RNN; matching lower bound results for standard RNN is an interesting research direction. Also, one can obtain a $O(T^2)$ generalization bound using VC-dimension based arguments [5]. But such bounds hold for specific settings like binary y, and are independent of problem hardness parameterized by the size of the weight matrices ($R_{\mathbf{W}}, R_{\mathbf{U}}$).

Finally, note that in the above analysis we fix $\alpha = O(1/T)$, $\beta = 1 - \alpha$ while in practice FastRNN learns α, β (which is similar to performing cross-validation on α, β). However, interestingly, across datasets the learned α values indeed display a similar scaling wrt T Figure 4, 6 & Table 7 in Appendix.

3.2 FastGRNN

While FastRNN controls the condition number of gradient reasonably well, it is still using only two scalars to control it, which might be restrictive for some datasets. This concern is addressed by a novel architecture (FastGRNN) that uses the scalar controlled peephole connection for every coordinate of the hidden state $\mathbf{h}_t$. That is,

$$\mathbf{z}_t = \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}_z), \tag{6}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}_h), \tag{7}$$

$$\mathbf{h}_t = (\zeta(\mathbf{1} - \mathbf{z}_t) + \nu) \odot \tilde{\mathbf{h}}_t + \mathbf{z}_t \odot \mathbf{h}_{t-1}, \tag{8}$$

where $0 \leq \zeta \leq 1$, $0 \leq \nu \leq 1$ are trainable parameters, and $\sigma : \mathbb{R} \to \mathbb{R}$ is a non-linear function such as $\tanh$, sigmoid and can vary across datasets. Note that each coordinate of $\mathbf{z}_t$ is similar to parameter β in (3) and $\zeta(\mathbf{1} - \mathbf{z}_t) + \nu$'s coordinates simulate α parameter; also if $\nu \approx 0, \zeta \approx 1$ then it satisfies the intuition that $\alpha + \beta = 1$. It was observed that across all datasets, this gating mechanism outperformed the simple vector extension of FastRNN where each coordinate of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ is learned.

Instead, FastGRNN computes each coordinate of gate $\mathbf{z}_t$ using a non-linear function of $\mathbf{x}_t$ and $\mathbf{h}_t$. To minimize the number of parameters, FastGRNN reuses the matrices $\mathbf{W}$, $\mathbf{U}$ for the vector-valued gating function as well. Hence, FastGRNN's computational complexity is almost same as that of RNN but its accuracy and training stability is on par with expensive gated architectures like LSTM/GRU.

Sparse Low-rank Representation: FastGRNN further compresses the model size by using a low-rank and a sparse representation of the parameter matrices $\mathbf{W}$, $\mathbf{U}$. That is,

$$\mathbf{W} = \mathbf{W}^1(\mathbf{W}^2)^T, \ \mathbf{U} = \mathbf{U}^1(\mathbf{U}^2)^T, \ \|\mathbf{W}^i\|_0 \leq s_w^i, \ \|\mathbf{U}^i\|_0 \leq s_u^i, \ i = \{1, 2\}, \tag{9}$$

where $\mathbf{W}^1 \in \mathbb{R}^{D \times r_w}, \mathbf{W}^2 \in \mathbb{R}^{\hat{D} \times r_w}$, and $\mathbf{U}^1, \mathbf{U}^2 \in \mathbb{R}^{\hat{D} \times r_u}$. Hyperparameters r_w, s_w, r_u, s_u provide an efficient way to control the *accuracy-memory* trade-off for FastGRNN and are typically set via cross-validation. In particular, such compression is critical for FastGRNN to fit in RAM of resource-constrained devices. Second, this low-rank representation brings down the prediction time by reducing the cost at each time step from $\mathcal{O}(\hat{D}(D + \hat{D}))$ to $\mathcal{O}(r_w(D + \hat{D}) + r_u\hat{D})$. This enables FastGRNN to provide on-device prediction in real-time on devices with limited battery resources.

3.2.1 Training FastGRNN

The weight parameters for FastGRNN: $\mathbf{\Theta}_{\text{FastGRNN}} = (\mathbf{W}^i, \mathbf{U}^i, \mathbf{b}^i)$ are trained jointly using projected batch stochastic gradient descent (b-SGD) (or other stochastic optimization methods) with typical batch sizes ranging from $64 - 128$. In particular, the optimization problem is given by:

$$\min_{\Theta_{\text{FastGRNN}}, \|\mathbf{W}^i\|_0 \leq s_w^i, \|\mathbf{U}^i\|_0 \leq s_u^i, i \in \{1,2\}} \mathcal{J}(\Theta_{\text{FastGRNN}}) = \frac{1}{n} \sum_j L(\mathbf{X}_j, y_j; \Theta_{\text{FastGRNN}}) \tag{10}$$

where L denotes the appropriate loss function (typically softmax cross-entropy). The training procedure for FastGRNN is divided into 3 stages:

(I) Learning low-rank representation (L): In the first stage of the training, FastGRNN is trained for e_1 epochs with the model as specified by (9) using b-SGD. This stage of optimization ignores the sparsity constraints on the parameters and learns a low-rank representation of the parameters.

(II) Learning sparsity structure (S): FastGRNN is next trained for e_2 epochs using b-SGD, projecting the parameters onto the space of sparse low-rank matrices after every few batches while maintaining support between two consecutive projection steps. This stage, using b-SGD with Iterative Hard Thresholding (IHT), helps FastGRNN identify the correct support for parameters $(\mathbf{W}^i, \mathbf{U}^i)$.

(III) Optimizing with fixed parameter support: In the last stage, FastGRNN is trained for e_3 epochs with b-SGD while freezing the support set of the parameters.

In practice, it is observed that $e_1 = e_2 = e_3 = 100$ generally leads to the convergence of FastGRNN to a good solution. Early stopping is often deployed in stages (II) and (III) to obtain the best models.

3.3 Byte Quantization (Q)

FastGRNN further compresses the model by quantizing each element of $\mathbf{W}^i, \mathbf{U}^i$, restricted to at most one byte along with byte indexing for sparse models. However, direct quantization of $\mathbf{W}^i, \mathbf{U}^i$ leads to a loss in accuracy due to gross approximation during integer arithmetic for integer valued parameters. Moreover, while such a quantization reduces model size, the prediction time can still be large due to floating point operations of the nonlinearities. FastGRNN overcomes these shortcomings by training $\mathbf{W}^i$ and $\mathbf{U}^i$ using *piecewise-linear* approximation of the non-linear functions, thereby ensuring that all the computations can be performed with integer arithmetic for integer valued $\mathbf{W}^i, \mathbf{U}^i, \mathbf{b}_h, \mathbf{b}_z$ and $\mathbf{X}_i$. During training, FastGRNN replaces the non-linear function in (8) with their respective approximations and uses the above mentioned training procedure to obtain $\mathbf{\Theta}_{\text{FastGRNN}}$. The floating point parameters are then quantized to ensure that all the relevant entities are integer-valued and the entire inference computation can be executed efficiently with integer arithmetic without significant drop in accuracy. For instance, Table 4, 5 shows that FastGRNN is able to reduce prediction time by 3-4x over FastGRNN-Q for several datasets with almost same accuracy. FastGRNN-LSQ, FastGRNN "minus" the Low-rank, Sparse and Quantized components, is the base model with no compression.

4 Experiments

Datasets: FastRNN and FastGRNN's performance was benchmarked on the following Internet of Things (IoT) tasks where having low model sizes and prediction times was critical to the success of the application: (a) Wakeword-2 [45] - detecting utterances of the words used to wake up a leading personal digital assistant (such as Alexa, Cortana, Google or Siri) widely available on smartphones, IoT devices, *etc.*; (b) Google30 [49] and Google12 - detection of utterances of 30 and 10 (plus background noise and silence) commands used to commonly interact with IoT devices and (c) HAR-2 [4] and DSA19 [3] - Human Activity Recognition (HAR) from an accelerometer and gyroscope on

Table 1: Dataset Statistics

Dataset	#Train	#Features	#Time Steps	#Test
Google-12	22,246	3,168	99	3,081
Google-30	51,088	3,168	99	6,835
Wakeword-2	195,800	5,184	162	83,915
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

Table 2: PTB Language Modeling - 1 Layer

Method	Test Perplexity	Train Perplexity	Model Size (KB)	Train Time (min)
RNN	144.71	68.11	129	**9.11**
FastRNN	127.76$^+$	109.07	513	11.20
FastGRNN-LSQ	**115.92**	89.58	513	12.53
FastGRNN	116.11	81.31	**39**	13.75
SpectralRNN	130.20	65.42	242	—
UGRNN	119.71	65.25	256	11.12
LSTM	117.41	69.44	2052	13.52

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 typically do not have prediction constraints and are therefore not the focus of this paper. Nevertheless, for the sake of completeness, experiments were also carried out on benchmark RNN tasks such as language modeling on the Penn Treebank (PTB) dataset [34], star rating prediction on a scale of 1 to 5 of Yelp reviews [21] and classification of MNIST images on a pixel-by-pixel sequence [32, 31].

All datasets, apart from Wakeword-2, are publicly available and their featurization and preprocessing details are provided in the Appendix. 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 results were reported on the publicly available test set. Whenever the train-val-test split was not available, the dataset was split into 70% for training (20% of which was used for validation) and 30% for testing. Table 1 lists the statistics of all datasets.

Baseline Algorithms & Implementation: FastRNN and FastGRNN were compared to standard RNNs [42], leading unitary RNN approaches such as SpectralRNN [52], Orthogonal RNN (oRNN) [37], Efficient Unitary Neural Network (EURNN) [23], FactoredRNN [48] and state-of-the-art gated RNNs including UGRNN [15], GRU [14] and LSTM [20]. Details of these methods are provided in Section 2 and the Appendix. Native Tensorflow implementations were used for the LSTM and GRU architectures. For all the other RNNs, publicly available implementations provided by the authors were used taking care to ensure that published results could be reproduced thereby verifying the code and hyper-parameter settings. All experiments were run on an Nvidia Tesla P40 GPU with Cuda 9.0 and Cudnn 7.1 on a machine with an Intel(R) Xeon(R) 2.60 GHz CPU with 12 cores.

Hyper-parameters: The hyper-parameters of each algorithm were set by a fine grained validation wherever possible or according to the settings recommended by the authors otherwise. Adam, Nesterov Momentum and SGD were used to optimize each algorithm on each dataset and the optimizer with the best validation performance was selected. The learning rate was initialized to 10^{-2} for all architectures except for RNNs where the learning rate was initialized to 10^{-3} to ensure stable training. Each algorithm was run for 200 epochs after which the learning rate was decayed by 10^{-1} and the algorithm run again for another 100 epochs. This procedure was carried out on all datasets except for Pixel MNIST where the learning rate was decayed by $\frac{1}{2}$ after each pass of 200 epochs. Batch sizes between 64 and 128 training points were tried for most architectures and a batch size of 100 was found to work well in general except for standard RNNs which required a batch size of 512. The nonlinearities, optimizers and exact hyperparameters for FastGRNN-Q are in Table 11. $^+$ represents use of ReLU as nonlinearity for FastRNN and $\tanh$ is the default.

Table 3: FastGRNN had up to 35x smaller models than leading RNNs with almost no loss in accuracy

	Dataset	Google-12			Google-30			Wakeword-2		
	Method	Accuracy (%)	Model Size (KB)	Train Time (hr)	Accuracy (%)	Model Size (KB)	Train Time (hr)	F1 Score	Model Size (KB)	Train Time(hr)
	RNN	73.25	56	1.11	80.05	63	2.13	89.17	8	**0.28**
Proposed	FastRNN	92.21$^+$	56	**0.61**	91.60$^+$	96	**1.30**	97.09	8	0.69
Proposed	FastGRNN-LSQ	**93.18**	57	0.63	**92.03**	45	1.41	**98.19**	8	0.83
Proposed	FastGRNN	92.10	**5.5**	0.75	90.78	**6.25**	1.77	97.83	**1**	1.08
Unitary	SpectralRNN	91.59	228	19.00	88.73	128	11.00	96.75	17	7.00
Unitary	EURNN	76.79	210	120.00	56.35	135	19.00	92.22	24	69.00
Unitary	oRNN	88.18	102	16.00	86.95	120	35.00	—	—	—
Unitary	FactoredRNN	53.33	1114	7.00	40.57	1150	8.52	—	—	—
Gated	UGRNN	92.63	75	0.78	90.54	260	2.11	98.17	16	1.00
Gated	GRU	93.15	248	1.23	91.41	257	2.70	97.63	24	1.38
Gated	LSTM	92.30	212	1.36	90.31	219	2.63	97.82	32	1.71

	Dataset	Yelp-5			HAR-2			DSA-19			Pixel-MNIST-10		
	Method	Accuracy (%)	RNN Model Size (KB)	Train Time (hr)	Accuracy (%)	Model Size (KB)	Train Time (hr)	Accuracy (%)	Model Size (KB)	Train Time (min)	Accuracy (%)	Model Size (KB)	Train Time (hr)
	RNN	47.59	130	**3.33**	91.31	29	0.11	71.68	20	**1.11**	94.10	71	45.56
Proposed	FastRNN	55.38	130	3.61	94.50$^+$	29	**0.06**	84.14	97	1.92	96.44	166	15.10
Proposed	FastGRNN-LSQ	**59.51**	130	3.91	95.38	29	0.08	**85.00**	208	2.15	**98.72**	71	**12.57**
Proposed	FastGRNN	59.43	**8**	4.62	**95.59**	3	0.10	83.73	**3.25**	2.10	98.20	**6**	16.97
Unitary	SpectralRNN	56.56	89	4.92	95.48	525	0.73	80.37	50	2.25	97.70	25	—
Unitary	EURNN	59.01	122	72.00	93.11	12	0.84	—	—	—	95.38	64	122.00
Unitary	oRNN	—	—	—	94.57	22	2.72	72.52	18	—	97.20	49	—
Unitary	FactoredRNN	—	—	—	78.65	**1**	0.11	73.20	1154	—	94.60	125	—
Gated	UGRNN	58.67	258	4.34	94.53	37	0.12	84.74	399	2.31	97.29	84	15.17
Gated	GRU	59.02	388	8.12	93.62	71	0.13	84.84	270	2.33	98.70	123	23.67
Gated	LSTM	59.49	516	8.61	93.65	74	0.18	84.84	526	2.58	97.80	265	26.57

Evaluation criteria: The emphasis in this paper is on designing RNN architectures which can run on low-memory IoT devices and which are efficient at prediction time. As such, the model size of each architecture is reported along with its training time and classification accuracy (F1 score on the Wakeword-2 dataset and perplexity on the PTB dataset). Prediction times on some of the popular IoT boards are also reported. Note that, for NLP applications such as PTB and Yelp, just the model size of the various RNN architectures has been reported. In a real application, the size of the learnt word-vector embeddings (10 MB for FastRNN and FastGRNN) would also have to be considered.

Results: Tables 2 and 3 compare the performance of FastRNN, FastGRNN and FastGRNN-LSQ to state-of-the-art RNNs. Three points are worth noting about FastRNN's performance. First, FastRNN's prediction accuracy gains over a standard RNN ranged from 2.34% on the Pixel-MNIST dataset to 19% on the Google-12 dataset. Second, FastRNN's prediction accuracy could surpass leading unitary RNNs on 6 out of the 8 datasets with gains up to 2.87% and 3.77% over SpectralRNN on the Google-12 and DSA-19 datasets respectively. Third, FastRNN's training speedups over all unitary and gated RNNs could range from 1.2x over UGRNN on the Yelp-5 and DSA-19 datasets to 196x over EURNN on the Google-12 dataset. This demonstrates that the vanishing and exploding gradient problem could be overcome by the addition of a simple peephole connection to the standard RNN thereby allowing FastRNN to train efficiently and stablely. This also demonstrates that the peephole connection offers a theoretically principled architecture that can often result in accuracy gains without limiting the expressive power of the hidden state transition matrix.

Tables 2 and 3 also demonstrate that FastGRNN-LSQ could be more accurate and faster to train than all unitary RNNs. Furthermore, FastGRNN-LSQ could match the accuracies and training times of state-of-the-art gated RNNs while having models that could be 1.18-4.87x smaller. This demonstrates that extending the peephole connection to a gate which reuses the RNN matrices increased accuracy with virtually no increase in model size over FastRNN in most cases. In fact, on Google-30 and Pixel-MNIST FastGRNN-LSQ's model size was 51 KB and 95 KB, lower than FastRNN's, as it had a lower hidden dimension indicating that the gate efficiently increased expressive power.

Finally, Tables 2 and 3 show that FastGRNN's accuracy was at most 1.13% worse than the best RNN but its model could be up to 35x smaller even as compared to low-rank unitary methods such as SpectralRNN. Figure 2 and 3 in the Appendix also shows that FastGRNN-LSQ and FastGRNN's classification accuracies could be higher than those obtained by the best unitary and gated RNNs for any given model size in the 0-128 KB range. This demonstrates the effectiveness of FastGRNN's low-rank, sparsification, and quantization components and allows FastGRNN to fit on the Arduino Uno having just 2 KB RAM and 32 KB flash memory. The individual impact of each of these three components can also be seen in the ablation experiments in Table 8, 9 and 10 of the Appendix.

Table 4: Prediction time in ms on the Arduino MKR1000

Method	Google-12	HAR-2	Wakeword-2
FastGRNN	537	162	175
FastGRNN-Q	2282	553	755
RNN	12028	2249	2232
UGRNN	22875	4207	6724
SpectralRNN	70902	—	10144

Table 5: Prediction time in ms on the Arduino Due

Method	Google-12	HAR-2	Wakeword-2
FastGRNN	242	62	77
FastGRNN-Q	779	172	238
RNN	3472	590	653
UGRNN	6693	1142	1823
SpectralRNN	17766	55558	2691

Prediction time on IoT boards: Unfortunately, most RNNs were too large to fit on an Arduino Uno apart from FastGRNN, Nevertheless, Table 4 shows that, for the same accuracy, FastGRNN could be 25-45x faster at prediction than UGRNN and 57-132x faster than SpectralRNN on an Arduino MKR1000 having an ARM Cortex M0+ microcontroller operating at 48 MHz with 32 KB RAM and 256 KB flash memory. Results on a more powerful board (Arduino Due) are presented in Table 5 and results on Raspberry Pi are presented in Table 12 of the Appendix.

5 Conclusion

This paper studies the FastRNN algorithm for addressing the issues of inaccurate training and inefficient prediction in RNNs. FastRNN develops a peephole connection architecture with the addition of two extra scalar parameters to address this problem. This paper then builds on FastRNN to develop a novel gated architecture, FastGRNN, which reuses RNN matrices in the gating unit. Further compression in the model size of FastGRNN is achieved by allowing the parameter matrices to be

low-rank, sparse and quantized. The performance of FastGRNN and FastRNN is benchmarked on several datasets and are shown to achieve state-of-the-art accuracies whilst having upto 35x smaller model as compared to leading gated RNN techniques.

6 Acknowledgements

We are grateful to Ankit Anand, Kunal Dahiya, Don Dennis, Dinesh Khandelwal, Shishir Patil, Adithya Pratapa, Harsha Vardhan Simhadri and Raghav Somani for many helpful discussions and feedback.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association. ISBN 978-1-931971-33-1. URL http://dl.acm.org/citation.cfm?id=3026877.3026899.

[2] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[3] K. Altun, B. Barshan, and O. Tunçel. Comparative study on classifying human activities with miniature inertial and magnetic sensors. *Pattern Recognition*, 43(10):3605–3620, 2010. URL https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities.

[4] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International workshop on ambient assisted living*, pages 216–223. Springer, 2012. URL https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones.

[5] M. Anthony and P. L. Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

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

[7] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[8] P. L. Bartlett and S. Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[9] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[10] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE, 2013.

[11] V. Campos, B. Jou, X. G. i Nieto, J. Torres, and S.-F. Chang. Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkwVAXyCW.

[12] G. Chen, C. Parada, and G. Heigold. Small-footprint keyword spotting using deep neural networks. In *Acoustics, speech and signal processing (icassp), 2014 ieee international conference on*, pages 4087–4091. IEEE, 2014.

[13] G. Chen, C. Parada, and T. N. Sainath. Query-by-example keyword spotting using long short-term memory networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 5236–5240. IEEE, 2015.

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

[15] J. Collins, J. Sohl-Dickstein, and D. Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

[16] S. Ghadimi and G. Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[17] N. Golowich, A. Rakhlin, and O. Shamir. Size-independent sample complexity of neural networks. *arXiv preprint arXiv:1712.06541*, 2017.

[18] C. Gupta, A. S. Suggala, A. Gupta, H. V. Simhadri, B. Paranjape, A. Kumar, S. Goyal, M. Udupa, R. amd Varma, and P. Jain. Protonn: Compressed and accurate knn for resource-scarce devices. In *Proceedings of the International Conference on Machine Learning*, August 2017.

[19] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

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

[21] Y. Inc. Yelp dataset challenge, 2017. URL https://www.yelp.com/dataset/challenge.

[22] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural networks*, 20(3):335–352, 2007.

[23] L. Jing, Y. Shen, T. Dubcek, J. Peurifoy, S. Skirlo, M. Tegmark, and M. Soljacić. Tunable efficient unitary neural networks (eunn) and their application to RNN. *arXiv preprint arXiv:1612.05231*, 2016.

[24] L. Jing, C. Gulcehre, J. Peurifoy, Y. Shen, M. Tegmark, M. Soljacić, and Y. Bengio. Gated orthogonal recurrent units: On learning to forget. *arXiv preprint arXiv:1706.02761*, 2017.

[25] C. Jose, P. Goyal, P. Aggrwal, and M. Varma. Local deep kernel learning for efficient non-linear svm prediction. In *International Conference on Machine Learning*, pages 486–494, 2013.

[26] C. Jose, M. Cisse, and F. Fleuret. Kronecker recurrent units. *arXiv preprint arXiv:1705.10142*, 2017.

[27] S. Kanai, Y. Fujiwara, and S. Iwamura. Preventing gradient explosions in gated recurrent units. In *Advances in Neural Information Processing Systems*, pages 435–444, 2017.

[28] V. Këpuska and T. Klein. A novel wake-up-word speech recognition system, wake-up-word recognition task, technology and evaluation. *Nonlinear Analysis: Theory, Methods & Applications*, 71(12):e2772–e2789, 2009.

[29] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] A. Kumar, S. Goyal, and M. Varma. Resource-efficient machine learning in 2 kb ram for the internet of things. In *Proceedings of the International Conference on Machine Learning*, August 2017.

[31] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[33] Z. Lu, V. Sindhwani, and T. N. Sainath. Learning compact recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5960–5964. IEEE, 2016.

[34] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

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

[36] J. McAuley and J. Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[37] Z. Mhammedi, A. Hellicar, A. Rahman, and J. Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. *arXiv preprint arXiv:1612.00188*, 2016.

[38] T. Mikolov, A. Deoras, S. Kombrink, L. Burget, and J. Cernocky. Empirical evaluation and combination of advanced language modeling techniques. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[39] S. Narang, E. Elsen, G. Diamos, and S. Sengupta. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017.

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

[41] M. Research. Edgeml. URL https://github.com/Microsoft/EdgeML.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[43] T. N. Sainath and C. Parada. Convolutional neural networks for small-footprint keyword spotting. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[44] S. Shankar, D. Robertson, Y. Ioannou, A. Criminisi, and R. Cipolla. Refining architectures of deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2212–2220, 2016.

[45] M. STCI. Wakeword dataset.

[46] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820, 2015.

[47] S. Team. Hey siri: An on-device dnn-powered voice trigger for apple's personal assistant, 2017. URL https://machinelearning.apple.com/2017/10/01/hey-siri.html.

[48] E. Vorontsov, C. Trabelsi, S. Kadoury, and C. Pal. On orthogonality and learning recurrent networks with long term dependencies. *arXiv preprint arXiv:1702.00071*, 2017.

[49] P. Warden. Speech commands: A public dataset for single-word speech recognition. *Dataset available from http://download. tensorflow. org/data/speech_ commands_ v0*, 1, 2017.

[50] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

[51] J. Ye, L. Wang, G. Li, D. Chen, S. Zhe, X. Chu, and Z. Xu. Learning compact recurrent neural networks with block-term tensor decomposition. *arXiv preprint arXiv:1712.05134*, 2017.

[52] J. Zhang, Q. Lei, and I. S. Dhillon. Stabilizing gradients for deep neural networks via efficient SVD parameterization. *arXiv preprint arXiv:1803.09327*, 2018.

A Convergence Analysis for FastRNN

Algorithm 1: Randomized Stochastic Gradient

Input: Initial point $\boldsymbol{\theta}_1$, iteration limit N, step sizes $\gamma_{k\geq 1}$, Probability mass function $P_R(\cdot)$ supported on $\{1, 2, \ldots, N\}$

Initialize: R be a random variable with probability mass function P_R

for $t = 1, \ldots, R$ **do**

  Obtain sample of stochastic gradient $\nabla f_t(\boldsymbol{\theta}_t)$

  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \gamma_t \nabla f_t(\boldsymbol{\theta}_t)$

Output: $\boldsymbol{\theta}_R$

Let $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{U}, \mathbf{v})$ represent the set of parameters of the scalar gated recurrent neural network. In order to prove the convergence properties of Randomized Stochastic Gradient (see Algorithm 1) as in [16], we first obtain a bound on the Lipschitz constant of the loss function $L(\mathbf{X}, y; \theta) := y \log(1 + \exp(-\mathbf{v}^\top \mathbf{h}_T))$ where $\mathbf{h}_T$ is the output of FastRNN after T time steps given input $\mathbf{X}$.

The gradient $\nabla_\theta L$ of the loss function is given by $(\frac{\partial L}{\partial \mathbf{W}}, \frac{\partial L}{\partial \mathbf{U}}, \frac{\partial L}{\partial \mathbf{v}})$ wherein

$$\frac{\partial L}{\partial \mathbf{U}} = \alpha \sum_{t=0}^{T} \mathbf{D}_t \left(\prod_{k=t}^{T-1} (\alpha \mathbf{U}^\top \mathbf{D}_{k+1} + \beta \mathbf{I}) \right) \nabla_{\mathbf{h}_T} f(\mathbf{h}_T) \mathbf{h}_{t-1}^T \tag{11}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \alpha \sum_{t=0}^{T} \mathbf{D}_t \left(\prod_{k=t}^{T-1} (\alpha U^\top \mathbf{D}_{k+1} + \beta \mathbf{I}) \right) \nabla_{\mathbf{h}_T} f(\mathbf{h}_T) \mathbf{x}_t^T \tag{12}$$

$$\frac{\partial L}{\partial \mathbf{v}} = \frac{y \exp(-\mathbf{v}^\top \mathbf{h}_T)}{1 + \exp(-\mathbf{v}^\top \mathbf{h}_T)} (-\mathbf{v}^\top \mathbf{h}_T) \mathbf{h}_T, \tag{13}$$

where $\nabla_{\mathbf{h}_T} f(\mathbf{h}_T) = c(\theta)(-\mathbf{v}^\top \mathbf{h}_T)v$, with $c(\theta) = \frac{1}{1+\exp(\mathbf{v}^\top \mathbf{h}_T)}$. We do a perturbation analysis and obtain a bound on $\|\nabla_\theta L(\theta) - \nabla_\theta L(\theta + \delta)\|_2$ where $\delta = (\delta_{\mathbf{W}}, \delta_{\mathbf{U}}, \delta_{\mathbf{v}})$.

Deviation bound for $\mathbf{h}_T$: In this subsection, we consider bounding the term $\|\mathbf{h}_T(\theta + \delta) - \mathbf{h}_T(\theta)\|_2$ evaluated on the same input $\mathbf{X}$. Note that for FastRNN, $\mathbf{h}_T = \alpha \tilde{h}_T + \beta \mathbf{h}_{T-1}$. For notational convenience, we use $\mathbf{h}'_T = \mathbf{h}_T(\theta + \delta)$ and $\mathbf{h}_T = \mathbf{h}_T(\theta)$.

$$\begin{aligned}
\|\mathbf{h}'_T - \mathbf{h}_T\|_2 &\leq \beta \|\mathbf{h}'_{T-1} - \mathbf{h}_{T-1}\|_2 + \alpha |\sigma(\mathbf{W}\mathbf{x}_{t-1} + \mathbf{U}\mathbf{h}_{t-1} - \sigma((\mathbf{W} + \delta_{\mathbf{W}})\mathbf{x}_{T-1} + (\mathbf{U} + \delta_{\mathbf{U}})\mathbf{h}'_{T-1})| \\
&\overset{\zeta_1}{\leq} \beta \|\mathbf{h}'_{T-1} - \mathbf{h}_{T-1}\|_2 + \alpha D \|\mathbf{U}\mathbf{h}_{T-1} - \delta_{\mathbf{W}}\mathbf{x}_{T-1} - \mathbf{U}\mathbf{h}'_{t-1} - \delta_{\mathbf{U}}\mathbf{h}'_{T-1}\|_2 \\
&\leq (\alpha D \|\mathbf{U}\|_2 + \beta) \|\mathbf{h}'_{T-1} - \mathbf{h}_{T-1}\|_2 + \alpha D (\|\delta_{\mathbf{U}}\|_2 + \|\delta_{\mathbf{W}}\|_2 R) \\
&\vdots \\
&\leq \alpha D (\|\delta_{\mathbf{U}}\|_2 + \|\delta_{\mathbf{W}}\|_2 R) \left(1 + (\alpha D \|\mathbf{U}\|_2 + \beta) + \ldots + (\alpha D \|\mathbf{U}\|_2 + \beta)^{T-1}\right) \\
&\overset{\zeta_2}{\leq} \alpha D (\|\delta_{\mathbf{U}}\|_2 + \|\delta_{\mathbf{W}}\|_2 R) \frac{(\alpha(D\|\mathbf{U}\|_2 - 1) + 1)^T}{\alpha(D\|\mathbf{U}\|_2 - 1)} \leq D (\|\delta_{\mathbf{U}}\|_2 + \|\delta_{\mathbf{W}}\|_2 R) \frac{(T\alpha(D\|\mathbf{U}\|_2 - 1) + 1)}{(D\|\mathbf{U}\|_2 - 1)} \\
&\leq O(TD^2 R(\alpha + 1))(\|\delta_{\mathbf{U}}\|_2 + \|\delta_{\mathbf{W}}\|_2)
\end{aligned}$$

where ζ_1 follows by using 1-lipschitz property of the sigmoid function and ζ_2 follows by setting $\alpha = O(1/TD)$ and $\beta \approx 1 - \alpha$.

Deviation bound for $c(\theta)$: In this subsection, we consider bounding the deviation $c(\theta) - c(\theta + \delta)$.

$$\begin{aligned}
|c(\theta) - c(\theta + \delta)| &\leq |\mathbf{v}^\top x_T - (v + \delta_{\mathbf{v}}^\top \mathbf{h}'_T)| \\
&\leq |\mathbf{v}^\top (\mathbf{h}_T - \mathbf{h}'_T)| + \|\delta_{\mathbf{v}}\|_2 \\
&\leq \|\mathbf{v}\|_2 \|\mathbf{h}_T - \mathbf{h}'_T\|_2 + \|\delta_{\mathbf{v}}\|_2 \\
&\leq \|\delta_{\mathbf{v}}\|_2 + R_{\mathbf{W}} O(TD^2 R(\alpha + 1))(\|\delta_{\mathbf{U}}\|_2 + \|\delta_{\mathbf{W}}\|_2)
\end{aligned}$$

Deviation bound for $\frac{\partial L}{\partial \mathbf{v}}$: In this subsection we consider the bounds on $\|\frac{\partial L}{\partial \mathbf{v}}(\theta) - \frac{\partial L}{\partial \mathbf{v}}(\theta+\delta)\|_2$.

$$\begin{aligned}
\left\|\frac{\partial L}{\partial \mathbf{v}}(\theta) - \frac{\partial L}{\partial \mathbf{v}}(\theta+\delta)\right\|_2 &= \left\|\frac{v^\top \mathbf{h}_T \mathbf{h}_T}{1+\exp\left(\mathbf{v}^\top \mathbf{h}_T\right)} - \frac{(\mathbf{v}+\delta_\mathbf{v})^\top \mathbf{h}'_T \mathbf{h}'_T}{1+\exp\left((\mathbf{v}+\delta_\mathbf{v})^\top \mathbf{h}'_T\right)}\right\|_2 \\
&= \left\|c(\theta)\mathbf{v}^\top \mathbf{h}_T \mathbf{h}_T - c(\theta+\delta)(\mathbf{v}+\delta_\mathbf{v})^\top \mathbf{h}'_T \mathbf{h}'_T)\right\|_2 \\
&\leq c(\theta+\delta)\|(\mathbf{v}^\top \mathbf{h}_T \mathbf{h}_T - v^\top \mathbf{h}'_T \mathbf{h}'_T))\|_2 \\
&\quad + R_\mathbf{W}(\|\delta_\mathbf{v}\|_2 + R_\mathbf{v} O(TD^2R(\alpha+1))(\|\delta_\mathbf{U}\|_2 + \|\delta_\mathbf{W}\|_2)) + \|\delta_\mathbf{v}\|_2 \\
&\leq \|(\mathbf{v}^\top \mathbf{h}'_T(\mathbf{h}_T - \mathbf{h}'_T)))\|_2 + R_\mathbf{v}\|\mathbf{h}_T - \mathbf{h}'_T\|_2 \\
&\quad + R_\mathbf{W}(\|\delta_\mathbf{v}\|_2 + R_\mathbf{v} O(TD^2R(\alpha+1))(\|\delta_\mathbf{U}\|_2 + \|\delta_\mathbf{W}\|_2)) + \|\delta_\mathbf{v}\|_2 \\
&\leq O(TD^2 R R_\mathbf{W} R_\mathbf{v})(\|\delta_\mathbf{v}\|_2 + \|\delta_\mathbf{U}\|_2 + \|\delta_\mathbf{v}\|_2)
\end{aligned}$$

Remark: Going forward, we use the O notation to contain *polynomial* dependence of the Lipschitz smoothness constant of L on $R_\mathbf{W}, R_\mathbf{U}, R_\mathbf{v}$ and the ambient dimensions $D, \hat{D}$.

Deviation bound for $\frac{\partial L}{\partial \mathbf{W}}$: In this subsection, we analyze $\|\frac{\partial L}{\partial \mathbf{W}}(\theta) - \frac{\partial L}{\partial \mathbf{W}}(\theta+\delta)\|_2$. Let $\|D\| = \max_k \|\mathbf{D}_k\|$. For this section, we let $c = c(\theta)$ and $c' = c(\theta+\delta)$.

$$\begin{aligned}
&\|\frac{\partial L}{\partial \mathbf{W}}(\theta) - \frac{\partial L}{\partial \mathbf{W}}(\theta+\delta)\|_2 \\
&= \alpha\|D\|R\left\|\sum_{t=0}^{T}\left(\prod_{k=t}^{T-1}(\alpha U^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)c(\mathbf{v}^\top \mathbf{h}_T)v - \left(\prod_{k=t}^{T-1}(\alpha(\mathbf{U}+\delta_\mathbf{U})^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)c'((v+\delta_\mathbf{v}))^\top \mathbf{h}'_T)(\mathbf{v}+\delta_\mathbf{v})\right\|_2 .
\end{aligned} \tag{14}$$

We first analyze the term $\sum_{t=0}^{T}\left(\prod_{k=t}^{T-1}(\alpha\mathbf{U}^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)c(v^\top \mathbf{h}_T)\mathbf{v}$ and obtain a decomposition of this which will be useful in further calculations.

$$\begin{aligned}
c(\mathbf{v}^\top \mathbf{h}_T)\mathbf{v} &= c'\mathbf{v}^\top \mathbf{h}_T)v + (c-c')\mathbf{v}^\top \mathbf{h}_T)\mathbf{v} \\
&= c'((\mathbf{v}+\delta_\mathbf{v})^\top \mathbf{h}_T)\mathbf{v} - \delta_\mathbf{v}^\top \mathbf{h}_T\mathbf{v} + (c-c')\mathbf{v}^\top \mathbf{h}_T)\mathbf{v} \\
&= c'((\mathbf{v}+\delta_\mathbf{v})^\top \mathbf{h}'_T)\mathbf{v} + c'((\mathbf{v}+\delta_\mathbf{v})^\top(\mathbf{h}_T - \mathbf{h}'_T))\mathbf{v} - \delta_\mathbf{v}^\top \mathbf{h}_T\mathbf{v} + (c-c')\mathbf{v}^\top \mathbf{h}_T)\mathbf{v}.
\end{aligned}$$

By using bounds computed in the previous subsections, we can write

$$\begin{aligned}
&\sum_{t=0}^{T}\left(\prod_{k=t}^{T-1}(\alpha\mathbf{U}^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)c(\mathbf{v}^\top \mathbf{h}_T)\mathbf{v} \\
&= \sum_{t=0}^{T}\left(\prod_{k=t}^{T-1}(\alpha\mathbf{U}^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)c'((\mathbf{v}+\delta_\mathbf{v})^\top \mathbf{h}'_T)\mathbf{v} + O(T^2)(\|\delta_\mathbf{U}\|_2 + \|\delta_\mathbf{W}\|_2 + \|\delta_\mathbf{v}\|_2)
\end{aligned}$$

For any fixed value of t, consider the term

$$\left(\prod_{k=t}^{T-1}(\alpha U^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)v - \left(\prod_{k=t}^{T-1}(\alpha(\mathbf{U}+\delta_\mathbf{U})^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)(\mathbf{v}+\delta_\mathbf{v}) \tag{15}$$

$$= \underbrace{\left(\prod_{k=t}^{T-1}(\alpha\mathbf{U}^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)\mathbf{v}}_{(I)} - \underbrace{\left(\prod_{k=t}^{T-1}(\alpha(\mathbf{U}+\delta_\mathbf{U})^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)\mathbf{v}}_{(II)} + \underbrace{\left(\prod_{k=t}^{T-1}(\alpha(\mathbf{U}+\delta_\mathbf{U})^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)\delta_\mathbf{v}}_{(III)} \tag{16}$$

We consider the second terms in the expansion above,

$$\begin{aligned}
\left(\prod_{k=t}^{T-1}(\alpha(\mathbf{U}+\delta_\mathbf{U})^\top \mathbf{D}_{k+1} + \beta\mathbf{I})\right)\mathbf{v} &= \left(\prod_{k=t}^{T-1}(\underbrace{\alpha\mathbf{U}^\top \mathbf{D}_{k+1} + \beta\mathbf{I}}_{c_{k+1}} + \delta + \mathbf{U}^\top \mathbf{D}_{k+1})\right)\mathbf{v} \\
&= \left(\prod_{k=t}^{T-1}\alpha\mathbf{U}^\top \mathbf{D}_{k+1} + \beta\mathbf{I}\right)\mathbf{v} + \delta_\mathbf{U}^\top\left(\sum_{k=t}^{T}\prod_{j=t}^{k} c_{j+1}\mathbf{D}_{k+1}\prod_{j=k+1}^{T-1} c_{j+1}\right)\mathbf{v} + O(\|\delta_\mathbf{U}\|_2^2)
\end{aligned}$$

Now, we consider the equation (16),

$$
\begin{aligned}
&\|(I)-(II)-(III)\|_2 \\
&\leq R_v\left(\left\|\delta_U^\top\left(\sum_{k=t}^{T}\prod_{j=t}^{k}c_{j+1}\mathbf{D}_{k+1}\prod_{j=k+1}^{T-1}c_{j+1}\right)v\right\|_2+O(\|\delta_U\|_2^2)+\left\|\left(\prod_{k=t}^{T-1}(\alpha(U+\delta_U)^\top\mathbf{D}_{k+1}+\beta\mathbf{I})\right)\delta_v\right\|_2\right) \\
&\leq R_v\|\delta_U\|_2\|v\|_2\|D\|\sum_{k=t}^{T-1}\left(\alpha\|U\|_2\|D\|+\beta\right)^{T-t}+R_v\|\delta_v\|_2\left(\alpha\|U+\delta_U\|_2\|D\|+\beta\right)^{T-t}
\end{aligned}
$$

Using the bound above, we bound (14) as follows, where $R=\sup_x\|x\|_2$.

$$
\begin{aligned}
&\alpha R_{\mathbf{v}}\|D\|R\left\|\left(\|\delta_{\mathbf{U}}\|_2\|\mathbf{v}\|_2\|D\|\sum_{t=0}^{T}(T-t)\left(\alpha\|\mathbf{U}\|_2\|D\|+\beta\right)^{T-t}+\|\delta_{\mathbf{v}}\|_2\sum_{t=0}^{T}\left(\alpha\|\mathbf{U}+\delta_{\mathbf{U}}\|_2\|D\|+\beta\right)^{T-t}\right)\right\|_2+O(T\|\delta_{\mathbf{U}}\|_2^2) \\
&\overset{\zeta_1}{=}\alpha R_{\mathbf{v}}\|D\|R\left\|\left(\|\delta_{\mathbf{U}}\|_2\|\mathbf{v}\|_2\|D\|\sum_{t=0}^{T}(T-t)\left(\alpha(\|\mathbf{U}\|_2\|D\|-1)+1\right)^{T-t}+\|\delta_{\mathbf{v}}\|_2\sum_{t=0}^{T}\left(\alpha(\|\mathbf{U}+\delta_{\mathbf{U}}\|_2\|D\|-1)+1\right)^{T-t}\right)\right\|_2 \\
&\quad+O(T\|\delta_{\mathbf{U}}\|_2^2) \\
&\overset{\zeta_2}{\leq}\alpha R_{\mathbf{v}}\|D\|R\left(\|\delta_{\mathbf{U}}\|_2\|\mathbf{v}\|_2\|D\|T\frac{(1+\alpha(\|\mathbf{U}\|_2\|D\|-1))^{T+1}}{\alpha(\|\mathbf{U}\|_2\|D\|-1)}+\|\delta_{\mathbf{v}}\|_2\frac{(1+\alpha(\|\mathbf{U}\|_2\|D\|-1))^{T+1}}{\alpha(\|\mathbf{U}\|_2\|D\|-1)}\right)+O(T\|\delta_{\mathbf{U}}\|_2^2) \\
&\overset{\zeta_3}{\leq}O(T^2)\|\delta_{\mathbf{U}}\|_2+O(T\|\delta_{\mathbf{U}}\|_2^2)+O(T^2)\|\delta_{\mathbf{v}}\|_2
\end{aligned}
$$

where ζ_1 follows by setting $\beta=1-\alpha$, ζ_2 follows from summing up the arithmetic-geometric progression and the geometric progression respectively, and, ζ_3 follows from the inequality $(1+x)^r\leq 1+rx/(1-(r-1)x)$ for $x(r-1)<1$ and using the assumption that $\alpha\approx O(1/T)$. This gives us a final bound on $\|\frac{\partial L}{\partial\mathbf{W}}(\theta)-\frac{\partial L}{\partial\mathbf{W}}(\theta+\delta)\|_F=O(T^2)\|\delta\|_F$, where we note again that the O notation consists of terms polynomial in $(R,R_{\mathbf{W}},R_{\mathbf{U}},R_{\mathbf{v}},D,\hat{D})$.

Deviation bound for $\frac{\partial L}{\partial\mathbf{U}}$: Following similar arguments as we did above for $\frac{\partial L}{\partial\mathbf{W}}$, we can derive the perturbation bound for the term $\frac{\partial L}{\partial\mathbf{U}}$ as

$$
\|\frac{\partial L}{\partial\mathbf{W}}(\theta)-\frac{\partial L}{\partial\mathbf{W}}(\theta+\delta)\|_F=O(T^2)\|\delta\|_F \tag{17}
$$

where the O notation consists of terms polynomial in $(R,R_{\mathbf{W}},R_{\mathbf{U}},R_{\mathbf{v}},D,\hat{D})$.

Using our bounds in corollary 2.2 of [16], we obtain the following convergence theorem.

Theorem 3.1 (Convergence Bound)**.** *Let $[(\mathbf{X}_1,y_1),\ldots,(\mathbf{X}_n,y_n)]$ be the given labeled sequential training data. Let $L(\boldsymbol{\theta})=\frac{1}{n}\sum_i L(\mathbf{X}_i,y_i;\boldsymbol{\theta})$ be the loss function with $\boldsymbol{\theta}=(\mathbf{W},\mathbf{U},\mathbf{v})$ be the parameters of FastRNN architecture (3). Then, randomized stochastic gradient descent [16], a minor variation of SGD, when applied to the data for a maximum of N iteration outputs a solution $\widehat{\boldsymbol{\theta}}$ such that:*

$$
\mathbb{E}[\|\nabla_{\boldsymbol{\theta}}L(\widehat{\boldsymbol{\theta}})\|_2^2\|]\leq\mathcal{B}_N:=\frac{O(T^2)L(\boldsymbol{\theta}_0)}{N}+\left(\bar{D}+\frac{4R_{\mathbf{W}}R_{\mathbf{U}}R_{\mathbf{v}}}{\bar{D}}\right)\frac{O(T^3)}{\sqrt{N}},
$$

where $R_{\mathbf{X}}=\max_{\mathbf{X}}\|\mathbf{X}\|_F$ for $\mathbf{X}=\{\mathbf{U},\mathbf{W},\mathbf{v}\}$, $L(\boldsymbol{\theta}_0)$ is the loss of the initial classifier, and the step-size of the k-th SGD iteration is fixed as: $\gamma_k=\min\left\{\frac{1}{T^2},\frac{\bar{D}}{T\sqrt{N}}\right\}, k\in[N],\ \ \bar{D}\geq 0$.

A.1 Generalization Bound for FastRNN

In this subsection, we compute the Rademacher complexity of the class of real valued scalar gated recurrent neural networks such that $\|\mathbf{U}\|_F\leq R_{\mathbf{U}},\|\mathbf{W}\|_F\leq R_{\mathbf{W}}$. Also the input $\mathbf{x}_t$ at time step t is assumed to be point-wise bounded $\|\mathbf{x}_t\|_2\leq R_{\mathbf{x}}$ The update equation of FastRNN is given by

$$
\mathbf{h}_{t+1}=\alpha\sigma(\mathbf{W}\mathbf{x}_t+\mathbf{U}\mathbf{h}_t)+\beta\mathbf{h}_t.
$$

For the purpose of this section, we use the shorthand $\mathbf{h}_t^i$ to denote the hidden vector at time t corresponding to the i^{th} data point $\mathbf{X}^i$. We denote the Rademacher complexity of a T layer FastRNN by $\mathcal{R}_n(\mathcal{F}_T)$ evaluated

using n data points.

$$
\begin{aligned}
n\mathcal{R}_n(\mathcal{F}_T) &= \mathbb{E}_\epsilon \left[\sup_{\mathbf{W},\mathbf{U}} \left\| \sum_{i=1}^{n} \epsilon_i \mathbf{h}_T^i \right\| \right] \\
&= \mathbb{E}_\epsilon \left[\sup_{\mathbf{W},\mathbf{U}} \left\| \sum_{i=1}^{n} \epsilon_i \left(\alpha\sigma(\mathbf{W}\mathbf{x}_{T-1}^i + \mathbf{U}\mathbf{h}_{T-1}^i) + \beta \mathbf{h}_{T-1}^i \right) \right\| \right] \\
&\overset{\zeta_1}{\leq} \mathbb{E}_\epsilon \left[\sup_{\mathbf{W},\mathbf{U}} \beta \left\| \sum_{i=1}^{n} \epsilon_i \mathbf{h}_{T-1}^i \right\| \right] + \mathbb{E}_\epsilon \left[\sup_{\mathbf{W},\mathbf{U}} \alpha \left\| \sum_{i=1}^{n} \epsilon_i (\sigma(\mathbf{W}\mathbf{x}_{T-1}^i + \mathbf{U}\mathbf{h}_{T-1}^i)) \right\| \right] \\
&\overset{\zeta_2}{\leq} \beta\mathcal{R}_n(\mathcal{F}_{T-1}) + 2\mathbb{E}_\epsilon \left[\sup_{\mathbf{W},\mathbf{U}} \alpha \left\| \sum_{i=1}^{n} \epsilon_i (\mathbf{W}\mathbf{x}_{T-1}^i + \mathbf{U}\mathbf{h}_{T-1}^i) \right\| \right] \\
&\overset{\zeta_3}{\leq} \beta\mathcal{R}_n(\mathcal{F}_{T-1}) + 2\alpha\mathbb{E}_\epsilon \left[\sup_{W} \left\| \sum_{i=1}^{n} \epsilon_i \mathbf{W}\mathbf{x}_{T-1}^i \right\| \right] + 2\alpha\mathbb{E}_\epsilon \left[\sup_{\mathbf{W},\mathbf{U}} \left\| \sum_{i=1}^{n} \epsilon_i \mathbf{U}\mathbf{h}_{T-1}^i) \right\| \right] \\
&\overset{\zeta_4}{\leq} \beta\mathcal{R}_n(\mathcal{F}_{T-1}) + 2\alpha R_{\mathbf{W}}\mathbb{E}_\epsilon \left[\left\| \sum_{i=1}^{n} \epsilon_i \mathbf{x}_{T-1}^i \right\| \right] + 2\alpha R_{\mathbf{U}}\mathbb{E}_\epsilon \left[\sup_{\mathbf{W},\mathbf{U}} \left\| \sum_{i=1}^{n} \epsilon_i \mathbf{h}_{T-1}^i) \right\| \right] \\
&\leq (\beta + 2\alpha R_{\mathbf{U}})\mathcal{R}_n(\mathcal{F}_{T-1}) + 2\alpha R_{\mathbf{W}} R_{\mathbf{X}} \sqrt{n} \\
&\leq (\beta + 2\alpha R_{\mathbf{U}})^2 \mathcal{R}_n(\mathcal{F}_{T-2}) + 2\alpha R_{\mathbf{W}} R_{\mathbf{X}} \sqrt{n}(1 + (\beta + 2\alpha R_{\mathbf{U}})) \\
&\vdots \\
&\leq 2\alpha R_{\mathbf{W}} R_{\mathbf{X}} \sum_{t=0}^{T-1} (\beta + 2\alpha R_{\mathbf{U}})^{T-t} \sqrt{n} \\
&\leq 2\alpha R_{\mathbf{W}} R_{\mathbf{X}} \left(\frac{(\beta + 2\alpha R_{\mathbf{U}})^{T+1} - 1}{(\beta + 2\alpha R_{\mathbf{U}}) - 1} \right) \sqrt{n} \\
&\leq 2\alpha R_{\mathbf{W}} R_{\mathbf{X}} \left(\frac{(1 + \alpha(2R_U - 1))^{T+1} - 1}{\alpha(2R_{\mathbf{U}} - 1)} \right) \sqrt{n} \\
&\overset{\zeta_5}{\leq} 2R_{\mathbf{W}} R_{\mathbf{X}} \left(\frac{1 + 2\alpha(2R_{\mathbf{U}} - 1)(T+1)}{(2R_{\mathbf{U}} - 1)} \right) \sqrt{n},
\end{aligned}
$$

where ζ_1, ζ_3 follows by triangle inequality and noting that the terms in the sum of expectation are pointwise bigger than the previous term, ζ_2 follows from the Ledoux-Talagrand contraction, ζ_4 follows using an argument similar from Lemma 1 in [17] and ζ_5 holds for $\alpha \leq \frac{1}{2(2R_{\mathbf{U}}-1)T}$.

Theorem 3.2 (Generalization Error Bound)**.** *[8] Let $\mathcal{Y}, \hat{\mathcal{Y}} \subseteq [0,1]$ and let $\mathcal{F}_T$ denote the class of FastRNN with $\|\mathbf{U}\|_F \leq R_{\mathbf{U}}, \|\mathbf{W}\|_F \leq R_{\mathbf{W}}$. Let the final classifier be given by $\sigma(\mathbf{v}^\top \mathbf{h}_T)$, $\|\mathbf{v}\|_2 \leq R_{\mathbf{v}}$. Let $L : \mathcal{Y} \times \hat{\mathcal{Y}} \to [0, B]$ be any 1-Lipschitz loss function. Let D be any distribution on $\mathcal{X} \times \mathcal{Y}$ such that $\|\mathbf{x}_{it}\|_2 \leq R_x$ a.s. Let $0 \leq \delta \leq 1$. Then with probability atleast $(1-\delta)$ for $\alpha \leq \frac{1}{2(2R_{\mathbf{U}}-1)T}$, all functions $f \in \mathbf{v} \circ \mathcal{F}_T$ satisfy*

$$
\mathbb{E}_D[L(f(\mathbf{X}), y)] \leq \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{X}_i), y_i) + \mathcal{C} \frac{O(\alpha T)}{\sqrt{n}} + B\sqrt{\frac{\ln(\frac{1}{\delta})}{n}},
$$

where $\mathcal{C} = R_{\mathbf{W}} R_{\mathbf{U}} R_{\mathbf{x}} R_{\mathbf{v}}$ represents the boundedness of the parameter matrices and the data.

The Rademacher complexity bounds for the function class $\mathcal{F}_T$ have been instantiated from the calculations above.

B Related Work Details

Recurrent Neural Networks (RNN): RNN were first proposed by [42] as models for prediction with time series data. The key idea was to allow connections between hidden units that can help the model retain information about the past inputs. It was shown in [9] that such recurrent models can be hard to train because of the vanishing and exploding gradient problem. The vanishing (exploding) gradient problem refers to the exponential decrease (increase) in the magnitude of the gradient of the long term components as compared to the short term components.

Unitary/Orthogonal Formulations: Recent attempts to rectify this problem have focused on controlling the magnitude of the singular values of the $\mathbf{U}_h$ matrix. [6] proposed the Unitary RNN wherein the transition matrix

$\mathbf{U}_h$ is parameterized to be a unitary matrix by construction, thereby constraining the magnitude of the singular values to be 1. The parameterization enforced by the authors, which is a product of 7 individual unitary matrices, restricts the matrix $\mathbf{U}_h$ to lie in a small subspace of the unitary matrices. However, such a parameterization has computational benefits, with each backpropagation update requiring $\mathcal{O}(\hat{D}\log(\hat{D}))$ time. [50] overcome the limited expressibility of this method by updating the $\mathbf{U}_h$ matrix using a projected gradient method (using Cayley transformations) to ensure that $\mathbf{U}_h$ remains unitary at the end of each update. Although $\mathbf{U}_h$ can span the complete space of unitary matrices, each backpropagation update requires $\mathcal{O}(\hat{D}^3)$ time. [23] propose another unitary parameterization for $\mathbf{U}_h$ using FFT matrices and show that their updates can be performed in $\mathcal{O}(\hat{D}^2\log(\hat{D}))$ time. [37] proposed the Orthogonal RNN (oRNN) architecture wherein $\mathbf{U}_h$ was constrained to be an orthogonal matrix. Specifically, $\mathbf{U_h}$ is written as a product of Householder reflection matrices. Using such a parameterization, the backpropagation updates could be efficiently performed in $\mathcal{O}(\hat{D}^2)$ time. All the above parameterizations of the RNN network limit the expressibility of the network by limiting the space in which $\mathbf{U}_h$ can lie.

Factorized Formulations: [48] propose a factorized RNN architecture where $\mathbf{U_h}$ is parameterized by its SVD $\mathbf{U}_h = \mathbf{U\Sigma V}^\top$. The entries of the diagonal matrix $\mathbf{\Sigma}$ are restricted to lie in a range $1 \pm \epsilon$, for a small value of ϵ while the orthogonality of $\mathbf{U}$ and $\mathbf{V}$ are maintained by updating them with a projected gradient method as in [50]. Recently, [52] proposed the Spectral RNN parameterization which combines the advantages of Factorized RNN and Orthogonal RNN. They parameterize the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ as a product of householder reflection matrices which allows them to perform the backpropogation updates in $\mathcal{O}(\hat{D}^2)$ time.

Although the Factorized RNN and Spectral RNN, in theory, allow the $\mathbf{U}_h$ to span the entire space of matrices by setting ϵ to be a very large value, in practice it is often the case that to stabilize training, the authors suggest using a small value of ϵ, which limits the expressibility of the network.

Gated Architectures: [20] proposed to change the architecture of RNN with the introduction of gated units in order to stabilize training. The authors proposed the Long Short-Term Memory (LSTM) architecture which had gated units inside every cell. Along with remembering past information, these gated units also allowed the model to forget parts of the information which are no longer required. More recently, [14] introduced the Gated Recurrent Unit (GRU) which differs from the LSTM network by not having an explicit memory unit. FastGRNN can be upto 2x-3x faster to train than GRU or LSTM networks while producing models which are 3x-4x smaller.

[15] proposed the UGRNN architecture which is a simplification of the GRU architecture. This architecture removes a gate (and hence two weight matrices $\mathbf{W}_r$ and $\mathbf{U}_r$) from the GRU architecture, thereby decreasing training and prediction time, without compromising on accuracy of the network. When compared with FastGRNN, UGRNN achieves a similar performance in terms of accuracy but has twice the model size, prediction time and energy requirements – which can be critical for on-device prediction for IoT applications.

The Leaky Integrator Unit (LIU) [22] has the same architecture as FastRNN, but it does not learn the RNN transition matrices, instead samples them from a prior hand-crafted distribution. In contrast, FastRNN learns RNN parameters as well as extent of "leak". As a result, FastRNN could be upto 34.41% better than LIU in classification accuracy.

Resource-efficient machine learning: Resource-efficient machine learning has seen a plethora of work in recent years [19, 18, 30, 7, 25, 44]. For a more detailed background, we refer the reader to [30] and references therein. Closely related to this work is the work by [18] (protoNN) and [30] (Bonsai) which focus on building classifiers to perform on IoT devices where both memory and battery life come at a premium. However, for the case when the input data is in the form of a time series, none of the above mentioned methods achieve the desired levels of accuracy.

Low rank and Sparsity Constraints: In order to compress the model to within the limits of the available flash memory on IoT devices, the weight matrices in FastGRNN are further restrained to be sparse and low rank during training without any loss in prediction accuracy. This additional compression, although, comes at a small cost of training time which increases by a factor of 1.2x. A similar technique was explored by [33] which used low rank constraints on the weight matrices of the RNN architecture. Also, [19] used a three stage pipeline of pruning the network, quantization of the trained weights and finally using Huffman coding to get a further compression. Their method, Deep Compression, was benchmarked on convolutional neural networks for image classification tasks and their techniques are orthogonal to the ones considered in this paper. [11] proposed SkipRNN which updated the state vector $\mathbf{h}_t$ intermittently to improving prediction time accuracy. Such techniques are complementary to the ones considered in this paper and can further improve FastGRNN's performance.

Optimization based methods: Another direction of research has been to develop better learning algorithms for training stabilization. [38] and [40] develop a gradient clipping approach wherein the authors clip the norm of the gradient vector used to update the parameters. [35] propose to use second order Hessian Free (HF) optimization method for stabilizing training. However, these methods are not the focus of this paper since they do not achieve state-of-the-art accuracies on benchmark datasets.

C Dataset Information

Google-12 & Google-30: Google Speech Commands dataset contains 1 second long utterances of 30 short words (30 classes) sampled at 16KHz. Standard log Mel-filter-bank featurization with 32 filters over a window size of 25ms and stride of 10ms gave 99 timesteps of 32 filter responses for a 1-second audio clip. For the 12 class version, 10 classes used in Kaggle's Tensorflow Speech Recognition challenge[1] were used and remaining two classes were noise and background sounds (taken randomly from remaining 20 short word utterances). Both the datasets were zero mean - unit variance normalized during training and prediction.

Wakeword-2: Wakeword-2 consists of 1.63 second long utterances sampled at 16KHz. This dataset was featurized in the same way as the Google Speech Commands dataset and led to 162 timesteps of 32 filter responses. The dataset was zero mean - unit variance normalized during training and prediction.

HAR-2[2]: Human Activity Recognition (HAR) dataset was collected from an accelerometer and gyroscope on a Samsung Galaxy S3 smartphone. The features available on the repository were directly used for experiments. The 6 activities were merged to get the binarized version. The classes {Sitting, Laying, Walking_Upstairs} and {Standing, Walking, Walking_Downstairs} were merged to obtain the two classes. The dataset was zero mean - unit variance normalized during training and prediction.

DSA-19[3]: This dataset is based on 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. The features available on the repository were used for experiments. The dataset was zero mean - unit variance normalized during training and prediction.

Yelp-5: Sentiment Classification dataset based on the text reviews[4]. The data consists of 500,000 train points and 500,000 test points from the first 1 million reviews. Each review was clipped or padded to be 300 words long. The vocabulary consisted of 20000 words and 128 dimensional word embeddings were jointly trained with the network.

Penn Treebank: 300 length word sequences were used for word level language modeling task using Penn Treebank (PTB) corpus. The vocabulary consisted of 10,000 words and the size of trainable word embeddings was kept the same as the number of hidden units of architecture.

Pixel-MNIST-10: Pixel-by-pixel version of the standard MNIST-10 dataset [5].The dataset was zero mean - unit variance normalized during training and prediction.

AmazonCat-13K [36]:[6] AmazonCat-13K is an extreme multi-label classification dataset with 13,330 labels. The raw text from title and content for Amazon products was provided as an input with each product being assigned to multiple categories. The input text was clipped or padded to ensure that it was 500 words long with a vocabulary of size 267,134. The 50 dimensional trainable word embeddings were initialized with GloVe vectors trained on Wikipedia.

The models were trained using Adam optimizer with a learning rate of 0.009 and batch size of 128. Binary Cross Entropy loss was used where the output of each neuron corresponds to the probability of a label being positive. 128 hidden units were chosen across architectures and were trained using PyTorch framework.

Table 6: Extreme Multi Label Classification

Dataset	AmazonCat - 13K					
	P@1	P@2	P@3	P@4	P@5	Model Size - RNN (KB)
GRU	92.82	85.18	77.09	69.42	61.85	268
RNN	40.24	28.13	22.83	20.29	18.25	89.5
FastGRNN-LSQ	92.66	84.67	76.19	66.67	60.63	90.0
FastRNN	91.03	81.75	72.37	64.13	56.81	89.5
UGRNN	92.84	84.93	76.33	68.27	60.63	179

[1] https://www.kaggle.com/c/tensorflow-speech-recognition-challenge

[2] https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

[3] https://archive.ics.uci.edu/ml/datasets/Daily+and+Sports+Activities

[4] https://www.yelp.com/dataset/challenge

[5] http://yann.lecun.com/exdb/mnist/

[6] http://manikvarma.org/downloads/XC/XMLRepository.html

The results in Table 6 show that FastGRNN-LSQ achieves classification performance similar to state-of-the-art gated architectures (GRU, LSTM) while still having 2-3x lower memory footprint. Note that the model size reported doesn't include the embeddings and the final linear classifier which are memory intensive when compared to the model itself. FastRNN, as shown in the earlier experiments, stabilizes standard RNN and achieves an improvement of over 50% in classification accuracy (P@1).

D Supplementary Experiments

Accuracy vs Model Size: This paper evaluates the trade-off between model size (in the range 0-128Kb) and accuracy across various architectures.

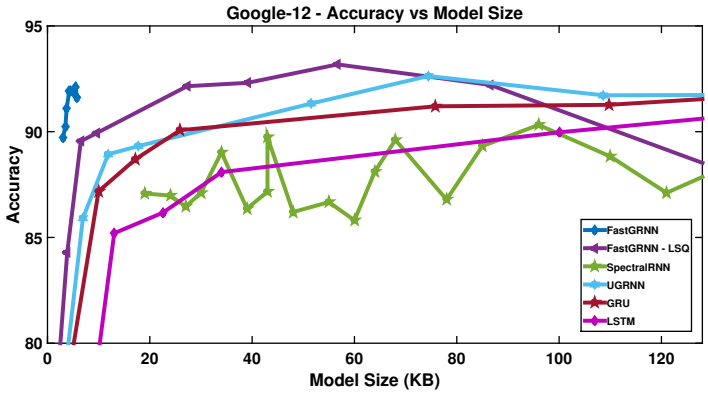


Figure 2: Accuracy vs Model Size

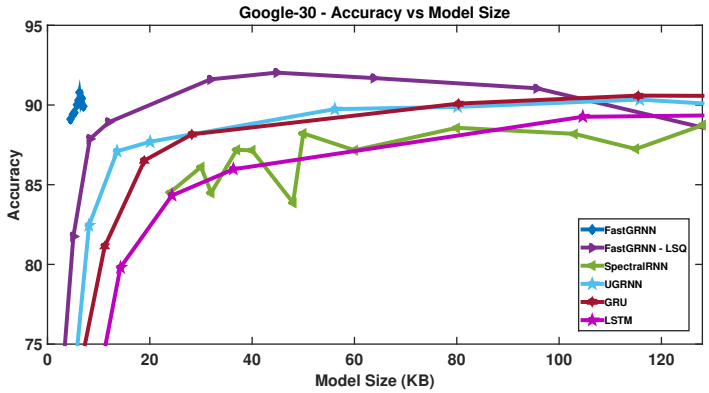


Figure 3: Accuracy vs Model Size

Figures 2, 3 show the plots for analyzing the model-size vs accuracy trade-off for FastGRNN, FastGRNN-LSQ along with leading unitary method SpectralRNN and the gated methods like UGRNN, GRU, and LSTM. FastGRNN is able to achieve state-of-the-art accuracies on Google-12 and Goole-30 datasets at significantly lower model sizes as compared to other baseline methods.

α, β and α/β vs T in FastRNN: Theorem 3.1 suggests that for the convergence analysis to hold, the scalar constant α should scale as $O(1/T)$. Indeed this was observed during the experiments when α was made a trainable parameter. Figure 4 and Table 7 summarize these results.

Table 7 shows the values of α and β learnt on three datasets. Table 7 shows that the relative error between β and $1-\alpha$ for Google-12 with 99 timesteps is 2.15%, HAR-2 with 128 timesteps is 3.21% and MNSIT-10 with 112 timesteps is 0.68%. Also, α/β value for Google-12 with 99 timesteps is 0.069, HAR-2 with 128 timesteps is 0.067 and MNIST-10 with 112 timesteps is 0.065, which reflects the trend suggested by our analysis. Figure 5 & 6 show the variation of β and α/β with number of timesteps T respectively. For all three datasets, we observe that $\alpha/\beta << 1$ and for large enough timesteps, $\beta \rightarrow 1-\alpha$. For short sequences, where there is a lower likelihood of gradients exploding or vanishing, it is helpful to let β be a free parameter, different from $1-\alpha$ (see Table 11, Appendix B). Enforcing the constraint $\beta = 1-\alpha$ on short sequences can often lead to a drop in accuracy. For example, on the Google-12 dataset with 33 timesteps, one observes a drop of upto 1.5%.

Table 7: α vs Timesteps for FastRNN with $\tanh$ non-linearity

Google-12			HAR-2			MNIST-10		
Timesteps	α	β	Timesteps	α	β	Timesteps	α	β
99	0.0654	0.9531	128	0.0643	0.9652	112	0.0617	0.9447
33	0.2042	0.8898	64	0.117	0.9505	56	0.1193	0.9266
11	0.5319	0.7885	32	0.1641	0.9606	28	0.2338	0.8746
9	0.5996	0.7926	16	0.2505	0.9718	14	0.385	0.8251
3	0.6878	0.8246	8	0.3618	0.9678	7	0.5587	0.8935

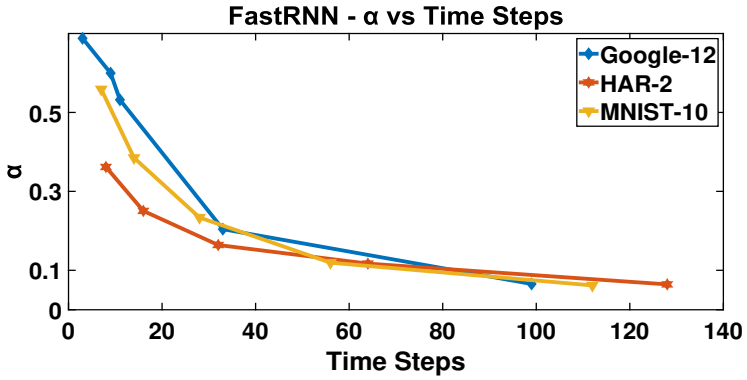


Figure 4: α vs T in FastRNN

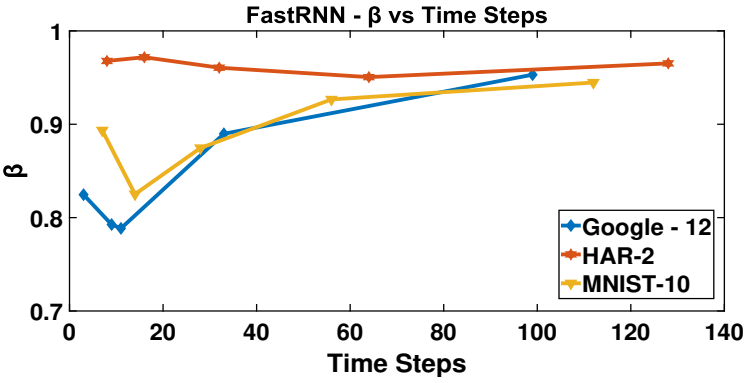


Figure 5: β vs T in FastRNN

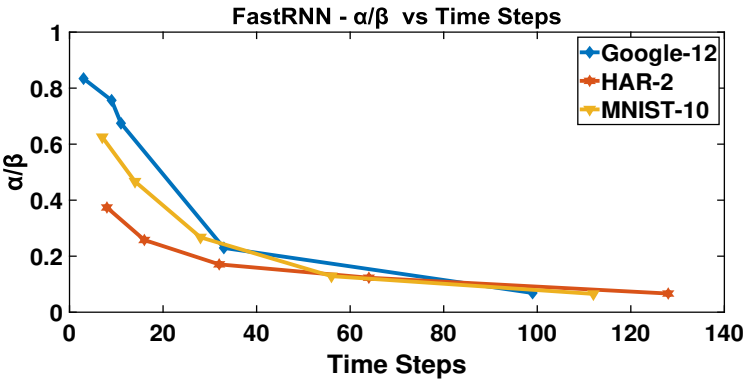


Figure 6: α/β vs T in FastRNN

Bias due to the initial hidden states: In order to understand the bias induced at the output by the initial hidden state h_0, we evaluated a trained FastRNN classifier on the Google-12 dataset with 3 different initializations sampled from a standard normal distribution. The resulting accuracies had a mean value of 92.08 with a standard deviation of 0.09, indicating that the initial state does not induce a bias in FastRNN prediction in the learning setting.In the non-learning setting, the initial state can bias the final solution for very small values of α. Indeed, setting $\alpha = 0$ and $\beta = 1$ will bias the final output to the initial state. However, as Figure 7indicates, such an effect is observed only for extremely small values of $\alpha \in (0, 0.005)$. In addition, there is a large enough range for $\alpha \in (0.005, 0.08)$ where the final output of FastRNN is not biased and is easily learnt by FastRNN.

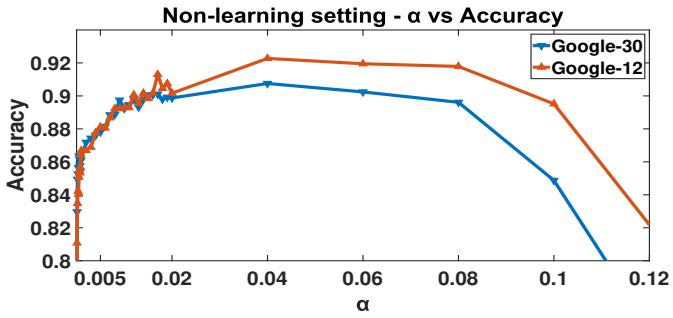


Figure 7: Accuracy vs α in non-learning setting where the parameters of the classifier was learnt and evaluated for a range of fixed α values (using 99 timesteps).

E Compression Components of FastGRNN

The Compression aspect of FastGRNN has 3 major components: 1) Low-Rank Parameterization (L) 2) Sparsity (S) and 3) Byte Quantization (Q). The general trend observed across dataset is that Low-Rank parameterization increase classification accuracies while the sparsity and quantization help reduced the model sizes by 2x and 4x respectively across datasets.

Tables 8, 9 and 10 show the trend when each of the component is gradually removed from FastGRNN to get to FastGRNN-LSQ. Note that the hyperparameters have been re-tuned along with the relevant constraints to obtain each model in the table.

Table 8: Components of Compression

Dataset	FastGRNN		FastGRNN-Q		FastGRNN-SQ		FastGRNN-LSQ	
	Accuracy (%)	Model Size (KB)	Accuracy (%)	Model Size (KB)	Accuracy (%)	Model Size (KB)	Accuracy (%)	Model Size (KB)
Google-12	92.10	5.50	92.60	22	93.76	41	93.18	57
Google-30	90.78	6.25	91.18	25	91.99	38	92.03	45
HAR-2	95.59	3.00	96.37	17	96.81	28	95.38	29
DSA-19	83.73	3.25	83.93	13	85.67	22	85.00	208
Yelp-5	59.43	8.00	59.61	30	60.52	130	59.51	130
Pixel-MNIST-10	98.20	6.00	98.58	25	98.72	37	98.72	71

Table 9: Components of Compression for Wakeword-2

Dataset	FastGRNN		FastGRNN-Q		FastGRNN-SQ		FastGRNN-LSQ	
	F1 Score	Model Size (KB)	F1 Score	Model Size (KB)	F1 Score	Model Size (KB)	F1 Score	Model Size (KB)
Wakeword-2	97.83	1	98.07	4	98.27	8	98.19	8

Table 10: Components of Compression for PTB

Dataset	FastGRNN		FastGRNN-Q		FastGRNN-SQ		FastGRNN-LSQ	
	Test Perplexity	Model Size (KB)	Test Perplexity	Model Size (KB)	Test Perplexity	Model Size (KB)	Test Perplexity	Model Size (KB)
PTB-300	116.11	38.5	115.71	154	115.23	384	115.92	513

F Hyperparameters of FastGRNN-Q for reproducibility:

Table 11 lists the hyperparameters which were used to run the experiments with a random-seed of 42 on a P40 GPU card with CUDA 9.0 and CuDNN 7.1. One can use the Piece-wise linear approximations of $\tanh$ or sigmoid if they wishe to quantize the weights.

Table 11: Hyperparameters for reproducibility - FastGRNN-Q

Dataset	Hidden Units	r_w	r_u	s_w	s_u	Nonlinearity	Optimizer
Google-12	100	16	25	0.30	0.30	sigmoid	Momentum
Google-30	100	16	35	0.20	0.20	$\tanh$	Momentum
Wakeword-2	32	10	15	0.20	0.30	$\tanh$	Momentum
Yelp-5	128	16	32	0.30	0.30	sigmoid	Adam
HAR-2	80	5	40	0.20	0.30	$\tanh$	Momentum
DSA-19	64	16	20	0.15	0.05	sigmoid	Adam
Pixel-MNIST-10	128	1	30	1.00	0.30	sigmoid	Adam
PTB-10000	256	64	64	0.30	0.30	sigmoid	Adam

G Timing Experiments on IoT boards

Table 12 summarizes the timing results on the Raspberry Pi which has a more powerful processor as compared with Arduino Due. Note that the Raspberry Pi has special instructions for floating point arithmetic and hence quantization doesn't provide any benefit with respect to compute in this case, apart from bringing down the model size considerably.

Table 12: Prediction Time on Raspberry Pi 3 (ms)

Method	Google-12	HAR-2	Wakeword-2
FastGRNN	7.7	1.8	2.5
RNN	15.7	2.9	3.6
UGRNN	29.7	5.6	9.5
SpectralRNN	123.2	391.0	17.2

H Vectorized FastRNN

As a natural extension of FastRNN, this paper also benchmarked FastRNN-vector wherein the scalar α in FastRNN was extended to a vector and β was substituted with $\zeta(1-\alpha)+\nu$ with ζ and ν are trainable scalars in $[0, 1]$. Tables 13, 14 and 15 summarize the results for FastRNN-vector and a direct comparison shows that the gating enable FastGRNN is more accurate than FastRNN-vector.

Table 13: FastRNN Vector - 1

Dataset	Accuracy (%)	Model Size (KB)	Train Time (hr)
Google-12	92.98^{+}	57	0.71
Google-30	91.68^{+}	64	1.63
HAR-2	95.24^{+}	19	0.06
DSA-19	83.24	322	0.04
Yelp-5	57.19	130	3.73
Pixel-MNIST-10	97.27	44	13.75

Table 14: FastRNN Vector - 2

Dataset	F1 Score	Model Size (KB)	Train Time (hr)
Wakeword-2	97.82	8	0.86

Table 15: FastRNN Vector - 3

Dataset	Test Perplexity	Train Perplexity	Model Size (KB)	Train Time (min)
PTB-300	126.84	98.29	513	11.7