LETTER Communicated by Yoshua Bengio

Learning Simpler Language Models with the Differential State Framework

Alexander G. Ororbia II
ago109@ist.psu.edu
College of Information Sciences and Technology, Pennsylvania State University, State College, PA 16802, U.S.A.

Tomas Mikolov
tmikolov@fb.com
Facebook, New York, NY 10003, U.S.A.

David Reitter
reitter@psu.edu
College of Information Sciences and Technology, Pennsylvania State University, State College, PA 16802, U.S.A.

Learning useful information across long time lags is a critical and difficult problem for temporal neural models in tasks such as language modeling. Existing architectures that address the issue are often complex and costly to train. The differential state framework (DSF) is a simple and high-performing design that unifies previously introduced gated neural models. DSF models maintain longer-term memory by learning to interpolate between a fast-changing data-driven representation and a slowly changing, implicitly stable state. Within the DSF framework, a new architecture is presented, the delta-RNN. This model requires hardly any more parameters than a classical, simple recurrent network. In language modeling at the word and character levels, the delta-RNN outperforms popular complex architectures, such as the long short-term memory (LSTM) and the gated recurrent unit (GRU), and, when regularized, performs comparably to several state-of-the-art baselines. At the subword level, the delta-RNN's performance is comparable to that of complex gated architectures.

1 Introduction

Recurrent neural networks are becoming increasingly popular models for sequential data. The simple recurrent neural network (RNN) architecture (Elman, 1990), however, is not suitable for capturing longer-distance dependencies. Architectures that address this shortcoming include the long short-term memory (LSTM; Hochreiter & Schmidhuber, 1997a), the gated

Neural Computation **29, 3327–3352** (2017)
doi:10.1162/NECO_a_01017

recurrent unit (GRU; Chung, Gulcehre, Cho, & Bengio, 2014, 2015), and the structurally constrained recurrent network (SCRN; Mikolov, Joulin, Chopra, Mathieu, & Ranzato, 2014). While these can capture some longer-term patterns (20 to 50 words), their structural complexity makes it difficult to understand what is going on inside. One exception is the SCRN architecture, which is by design simple to understand. It shows that the memory acquired by complex LSTM models on language tasks does correlate strongly with simple weighted bags of words. This demystifies the abilities of the LSTM model to a degree; while some authors have suggested that the LSTM understands the language and even the thoughts being expressed in sentences (Choudhury, 2015), it is arguable whether this could be said about a model that performs equally well and is based on representations that are essentially equivalent to a bag of words.

One property of recurrent architectures that allows for the formation of longer-term memory is the self-connectedness of the basic units. This is most explicitly shown in the SCRN model, where one hidden layer contains neurons that do not have other recurrent connections except to themselves. Still, this architecture has several drawbacks; one has to choose the size of the fully connected and self-connected recurrent layers, and the model is not capable of modeling nonlinearities in the longer-term memory component.

In this work, we aim to increase representational efficiency—the ratio of performance to acquired parameters. We simplify general neural model architecture further and develop several variants under the differential state framework, where the hidden layer state of the next time step is a function of its current state and the delta change computed by the model. We do not present the differential state framework as a model of human memory for language. However, we point out its conceptual origins in surprisal theory (Boston, Hale, Kliegl, Patil, & Vasishth, 2008; Hale, 2001; Levy, 2008), which posits that the human language processor develops complex expectations of future words, phrases, and syntactic choices and that these expectations and deviations from them (surprisal) guide language processing (e.g., in reading comprehension). How complex the models are (in the human language processor) that form the expectation is an open question. The cognitive literature has approached this with existing parsing algorithms, probabilistic context-free grammars, or n-gram language models. We take a connectionist perspective. The differential state framework proposes to not just generatively develop expectations and compare them with actual state changes caused by observing new input; it explicitly maintains gates as a form of high-level error correction and interpolation. An instantiation of this framework, the delta-RNN, will be evaluated as a language model, and we will not attempt to simulate human performance such as in situations with garden-path sentences that need to be reanalyzed because of costly initial misanalysis.

2 The Differential State Framework and the Delta-RNN

In this section, we describe the proposed differential state framework (DSF), as well as several concrete implementations one can derive from it.

2.1 General Framework. The most general formulation of the architectures that fall under DSF distinguishes two forms of the hidden state. The first is a fast state, which is generally a function of the data at the current time step and a filtration (or summary function of past states). The second is a slow state, or data-independent state. This concept can be specifically viewed as a composition of two general functions, formally defined as

$$\begin{aligned}\mathbf{h}_t &= q_\Theta(\mathbf{x}_t, \mathbf{M}_{t-1}) \\ &= f_\psi[g_\theta(\mathbf{x}_t, \mathbf{M}_{t-1}), \mathbf{M}_{t-1}],\end{aligned} \tag{2.1}$$

where $\Theta = \{\theta, \psi\}$ are the parameters of the state machine and $\mathbf{M}_{t-1}$ is the previous latent information the model is conditioned on. In the case of most gated architectures, $\mathbf{M}_{t-1} = \mathbf{h}_{t-1}$, but in some others, as in the SCRN or the LSTM, $\mathbf{M}_{t-1} = \{\mathbf{h}_{t-1}, \mathbf{c}_{t-1}\}$[1] or could even include information such as decoupled memory, and in general will be updated as symbols are iteratively processed. We define $g_\theta(\cdot)$ to be any, possibly complicated, function that maps the previous hidden state and the currently encountered data point (e.g., a word, subword, or character token) to a real-valued vector of fixed dimensions using parameters θ. $f_\psi(\cdot)$, on the other hand, is defined to be the outer function that uses parameters ψ to integrate the fast state, as calculated by $g_\theta(\cdot)$, and the slowly moving, currently untransformed state $\mathbf{h}_{t-1}$. In the sections that follow, we describe simple formulations of these two core functions; in section 3, we show how currently popular architectures, like the LSTM and various simplifications, are instantiations of this framework.

The specific structure of equation 2.1 was chosen because we hypothesize the reason behind the success of gated neural architectures is largely that they have been treating next-step prediction tasks, like language modeling, as an interaction between two functions. One inner function focuses on integrating observed samples with a current filtration to create a new data-dependent hidden representation (or state "proposal"), while an outer function focuses on computing the difference, or "delta," between the impression of the sub-sequence observed so far (i.e., $\mathbf{h}_{t-1}$) with the newly formed impression. For example, as a sentence is iteratively processed, there might not be much new information (or "suprisal") in a token's mapped hidden representation (especially if it is a frequently encountered token), thus requiring less change to the iteratively inferred global

[1] $\mathbf{c}_t$ refers to the "cell state" as in Hochreiter and Schmidhuber (1997b).

representation of the sentence.[2] However, encountering a new or rare token (especially an unexpected one) might bias the outer function to allow the newly formed hidden impression to more strongly influence the overall impression of the sentence, which will be useful when predicting what token or symbol will come next. In section 5, we present a small demonstration using one of the trained word models to illustrate the intuition just described.

In the subsections that follow, we describe the ways we chose to formulate $g_\theta(\cdot)$ and $f_\psi(\cdot)$ in the experiments of our study. The process we followed for developing the concrete implementations of $g_\theta(\cdot)$ and $f_\psi(\cdot)$ involved starting from the simplest possible form using the fewest (if any) possible parameters to compose each function and testing it in preliminary experiments to verify its usefulness.

It is important to note that equation 2.1 is still general enough to allow for future design of more clever or efficient functions that might improve the performance and long-term memory capabilities of the framework. More importantly, one might view the parameters ψ that $f_\psi(\cdot)$ uses as possibly encapsulating structures that can be used to store explicit memory vectors, as is the case in stacked-based RNNs (Das, Giles, & Sun, 1992; Joulin & Mikolov, 2015) or linked-list-based RNNs (Joulin & Mikolov, 2015).

2.2 Forms of the Outer Function. Keeping $g_\theta(\cdot)$ as general as possible, here we describe several ways one could design $f_\psi(\cdot)$, the function meant to decide how new and old hidden representations will be combined at each time step. We will strive to introduce as few additional parameters as necessary, and experimental results will confirm the effectiveness of our simple designs.

One form that $f_\psi(\cdot)$ could take is a simple weighted summation, as follows:

$$\begin{aligned}\mathbf{h}_t &= f_\psi[g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}), \mathbf{h}_{t-1}] \\ &= \Phi(\gamma[g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})] + \beta\mathbf{h}_{t-1}),\end{aligned} \tag{2.2}$$

where $\Phi(\cdot)$ is an element-wise activation applied to the final summation and γ and β are bias vectors meant to weight the fast and slow states, respectively. In equation 2.2, if $\gamma = \beta = 1$, no additional parameters have been introduced, making the outer function simply a rigid summation operator followed by a nonlinearity. However, one will notice that $\mathbf{h}_{t-1}$ is transmitted

[2] One way to extract a sentence representation from a temporal neural language model would be simply to take the last hidden state calculated upon reaching a symbol such as punctuation (e.g., a period or exclamation point). This is sometimes referred to as encoding variable-length sentences or paragraphs to a real-valued vector of fixed dimensionality.

across a set of fixed identity connections in addition to being transformed by $g_\theta(\cdot)$.

While γ and β could be chosen to be hyperparameters and tuned externally (as sort of per-dimension scalar multipliers), it might prove to be more effective to allow the model to learn these coefficients. If we introduce a vector of parameters $\mathbf{r}$, we can choose the fast and slow weights to be $\gamma = (1 - \mathbf{r})$ and $\beta = (\mathbf{r})$, facilitating simple interpolation. Adding these negligibly few additional parameters to compose an interpolation mechanism yields the state-model

$$\mathbf{h}_t = \Phi((1 - \mathbf{r}) \otimes g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}) + \mathbf{r} \otimes \mathbf{h}_{t-1}). \tag{2.3}$$

Note that we define $\otimes$ to be the Hadamard product. Incorporating this interpolation mechanism can be interpreted as giving the DSF model a flexible mechanism for mixing various dimensions of its longer-term memory with its more localized memory. Interpolation, especially through a simple gating mechanism, can be an effective way to allow the model to learn how to turn on or off latent dimensions, potentially yielding improved generalization performance, as was empirically shown by Serban, Ororbia, Alexander, Pineau, and Courville (2016).

Beyond fixing $\mathbf{r}$ to some vector of pre-initialized values, there are two simple ways to parameterize $\mathbf{r}$:

$$\mathbf{r} = 1/(1 + \exp(-\mathbf{b}_r)) \text{ or} \tag{2.4}$$

$$\mathbf{r} = 1/(1 + \exp(-[W\mathbf{x}_t + \mathbf{b}_r])), \tag{2.5}$$

where both forms only introduce an additional set of learnable bias parameters; however, equation 2.5 allows the data at time step t to interact with the gate. Thus, the outer function takes into account additional information from the input distribution when mixing stable and local states together. Unlike Serban et al. (2016), we constrain the interpolation mechanism to lie in the range $[0, 1]$ by using the logistic link function, $\sigma(v) = 1/(1 + \exp(-v))$, which will transform the biases into rates much like the rates of the SCRN. We crucially choose to share W in this particular mechanism for two reasons: (1) we avoid adding yet another matrix of input to hidden parameters and, much to our advantage, reuse the computation of the linear preactivation term $W\mathbf{x}_t$, and (2) additionally coupling the data preactivation to the gating mechanism will serve as further regularization of the input-to-hidden parameters (by restricting the number of learnable parameters, much as in classical autoencoders). Two error signals, $\frac{\partial \mathbf{r}}{\partial W}$ and $\frac{\partial \mathbf{z}_t}{\partial W}$, now take part in the calculation of the partial derivative $\frac{\partial \mathcal{L}(\mathbf{y}_t, \mathbf{x}_{t+1})}{\partial W}$ ($\mathbf{y}_t$ is the output of the model at t and $\mathbf{z}_t$ is the currently calculated state proposal).

Figure 1 depicts the architecture using the simple late-integration mechanism.

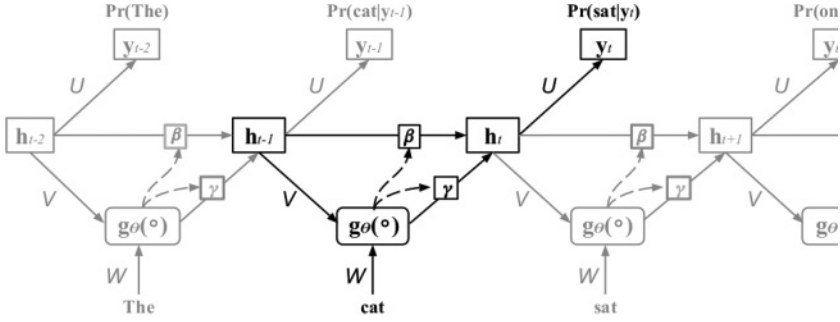

Figure 1: The Delta-RNN computation graph, unfolded over time. The learnable gates γ and β control how much influence the previous state and the current data-dependent state have on computing the model's next hidden state.

2.3 Forms of the Inner Function: Instantiating the Delta-RNN. When a concrete form of the inner function $g_\theta(\cdot)$ is chosen, we can fully specify an instantiation of the DSF. We will also show, in section 3, how many other commonly used RNN architectures can, in fact, be treated as special cases of this general framework defined under equation 2.1.

Starting from equation 2.2, if we fix $\gamma = 1$ and $\beta = 0$, we can recover the classical Elman RNN, where $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$ is a linear combination of the projection of the current data point and the projection of the previous hidden state, followed by a nonlinearity $\phi(\cdot)$. Specifically, the simple RNN's hidden state, where $\Phi(v) = v$ (the identity function), is

$$\begin{aligned}
\mathbf{h}_t &= \gamma \otimes g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}) + \beta \otimes \mathbf{h}_{t-1}, \\
\mathbf{h}_t &= 1 \otimes \phi(V\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b}) + 0 \otimes \mathbf{h}_{t-1}, \\
\mathbf{h}_t &= \phi(V\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b}),
\end{aligned} \tag{2.6}$$

where $\mathbf{h}_t$ is the hidden layer state at time t, $\mathbf{x}_t$ is the input vector, and $\theta = \{W, V\}$ contains the weight matrices. However, if we also set $\beta = 1$, we obtain a naive way to compute a delta change of states (the simple Delta-RNN) where instead $\phi(v) = v$, we have

$$\mathbf{h}_t = \Phi(V\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b} + \mathbf{h}_{t-1}). \tag{2.7}$$

Thus, the state can be implicitly stable, assuming W and V are initialized with small values and $\phi(\cdot)$ allows this by being partially linear. For example, we can choose $\phi(\cdot)$ to be the linear rectifier (or initialize the model so as to start out in the linear regime of the hyperbolic tangent). In this case, the

simple Delta-RNN does not need to learn anything to maintain a constant state over time.

Preliminary experimentation with this simple form, equation 2.7, often yielded unsatisfactory performance. This further motivated the development of the simple interpolation mechanism presented in equation 2.3. However, depending on how one chooses the nonlinearities, $\phi(\cdot)$ and $\Phi(\cdot)$, one can create different types of interpolation. Using an Elman RNN for $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$ as in equation 2.6, substituting into equation 2.3 can create what we propose as the late-integration[3] state model:

$$\begin{aligned}\mathbf{z}_t &= g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}) \\ &= \phi(V\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b}), \qquad (2.8)\\ \mathbf{h}_t &= \Phi((1-\mathbf{r}) \otimes \mathbf{z}_t + \mathbf{r} \otimes \mathbf{h}_{t-1}), \qquad (2.9)\end{aligned}$$

where $\Phi(\cdot)$ could be any choice of activation function, including the identity function. This form of interpolation allows for a more direct error propagation pathway since gradient information, once transmitted through the interpolation gate, has two pathways: through the nonlinearity of the local state (through $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$) and the pathway composed of implicit identity connections.

When using a simple Elman RNN, we have essentially described a first-order Delta-RNN. However, historically, second-order recurrent neural architectures have been shown to be powerful models in tasks such as grammatical inference (Giles et al., 1991) and noisy time series prediction (Giles, Lawrence, & Tsoi, 2001), as well as incredibly useful in rule extraction when treated as finite-state automata (Giles et al., 1992; Goudreau, Giles, Chakradhar, & Chen, 1994). Very recently, Wu, Zhang, Zhang, Bengio, and Salakhutdinov (2016) showed that the gating effect between the state-driven component and data-driven components of a layer's preactivations facilitated better propagation of gradient signals as opposed to the usual linear combination. A second-order version of $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$ would be highly desirable, not only because it further mitigates the vanishing gradient problem that plagues backpropagation through time (used in calculating parameter gradients of neural architectures), but because the form introduces negligibly few additional parameters. We do note that the second-order form we use, as in Wu et al. (2016), is a rank-1 matrix approximation of the actual tensor used in Giles et al. (1992) and Goudreau et al. (1994).

We can take the late-integration model, equation 2.9, and replace, similar to Giles et al. (1991), $\mathbf{z}_t$ with

[3] Late-integration might remind readers of the phrase "late fusion," as in the context of Wang and Cho (2015). However, they focused on merging the information from an external bag-of-words context vector with the standard cell state of the LSTM.

$$\mathbf{z}_t = \phi(V\mathbf{h}_{t-1} \otimes W\mathbf{x}_t + \mathbf{b}) \tag{2.10}$$

or a more general form (Wu et al., 2016):

$$\begin{aligned} \mathbf{d}_t^1 &= \alpha \otimes V_d\mathbf{h}_{t-1} \otimes W\mathbf{x}_t, \\ \mathbf{d}_t^2 &= \beta_1 \otimes V_d\mathbf{h}_{t-1} + \beta_2 \otimes W\mathbf{x}_t, \\ \mathbf{z}_t &= \phi(\mathbf{d}_t^1 + \mathbf{d}_t^2 + \mathbf{b}), \end{aligned} \tag{2.11}$$

where we note that $\mathbf{z}_t$ can be a function of any arbitrary incoming set of information signals that are gated by the last known state. The Delta-RNN will ultimately combine this data-driven signal $\mathbf{z}_t$ with its slow-moving state. More important, observe that even in the most general form (see equation 2.11), only a few further bias vector parameters, α, β_1, and β_2, are required.

Assuming a single hidden layer language model, with H hidden units and V input units (where V corresponds to the cardinality of the symbol dictionary), a full late-integration Delta-RNN that employs a second-order $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$ (see equation 2.11), has only $((H * H) + 2(H * V) + 5H + V)$ parameters,[4] which is only slightly larger than a classical RNN with only $((H * H) + 2(H * V) + H + V)$ parameters. This stands in stark contrast to the sheer number of parameters required to train commonly used complex architectures such as the LSTM (with peephole connections), with $(4(H * H) + 8(H * V) + 4H + V)$ parameters, and the GRU, with $(3(H * H) + 4(H * V) + 3H + V)$ parameters.

2.4 Regularizing the Delta-RNN. Regularization is often important when training large, overparameterized models. To control for overfitting, approaches range from structural modifications to impositions of priors over parameters (Neal, 2012). Commonly employed modern approaches include dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) and variations (Gal & Ghahramani, 2016) or mechanisms to control for internal covariate drift, such as batch normalization (Ioffe & Szegedy, 2015) for large feedforward architectures. In this letter, we investigate the effect that dropout will have on the Delta-RNN's performance.[5]

To introduce simple (nonrecurrent) dropout to the framework, our preliminary experiments uncovered that dropout was most effective when applied to the inner function $g(\cdot)$ as opposed to the outer function's computed

[4] $5H$ counts the hidden bias, the full interpolation mechanism $\mathbf{r}_t$ (see equation 2.5), and the second-order biases, $\{\alpha, \beta_1, \beta_2\}$.

[5] In preliminary experiments, we also investigated incorporating layer normalization (Ba, Kiros, & Hinton, 2016) into the Delta-RNN architecture; the details are in the appendix. We did not observe noticeable gains using layer normalization over dropout and thus report only the results of dropout in this letter.

delta state. For the full Delta-RNN, under dropout probability p_{drop}, this would lead to the following modification:

$$\mathbf{h}_t = \Phi((1 - \mathbf{r}) \otimes DROP(g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}), p_{drop}) + \mathbf{r} \otimes \mathbf{h}_{t-1}). \tag{2.12}$$

$DROP(\mathbf{x}, p_{drop}) = \mathbf{x} \otimes (\sim \mathbf{B}(1, p_{drop}))$ is the dropout operator that masks its input argument with a binary vector sampled from H independent Bernoulli distributions.

2.5 Learning under the Delta-RNN. Let $w_1, \ldots, w_N$ be a variable-length sequence of N symbols (such as words that would compose a sentence). In general, the distribution over the variables follows the graphical model:

$$P_\theta(w_1, \ldots, w_T) = \prod_{t=1}^{T} P_\Theta(w_t | w_{<t}), \tag{2.13}$$

where $\Theta = \{\psi, \theta\} = \{V, W, R, \mathbf{b}, \mathbf{b}_r, \alpha, \beta_1, \beta_2\}$ are the model parameters (of a full Delta-RNN).

No matter how the hidden state $\mathbf{h}_t$ is calculated, in this letter, it will ultimately be fed into a maximum-entropy classifier[6] defined as

$$P(w, \mathbf{h}_t) = P_\Theta(w | \mathbf{h}_t) = \frac{\exp(w^{\mathrm{T}} R \mathbf{h}_t)}{\sum_{w'} \exp(w^{\mathrm{T}} R \mathbf{h}_t)}, \tag{2.14}$$

To learn parameters for any of our models, we optimize with respect to the sequence negative log likelihood:

$$\mathcal{L} = -\sum_{i=1}^{N} \sum_{t=1}^{T} \log P_\Theta(w_t | \mathbf{h}). \tag{2.15}$$

Model parameters, $\Theta = \{\theta, \psi\}$, of the Delta-RNN are learned under an empirical risk minimization framework. We employ backpropagation of errors (or, rather, reverse-mode automatic differentiation with respect to this negative log-likelihood objective function) to calculate gradients and update the parameters using the method of steepest gradient descent. For all experiments conducted in this letter, we found that the ADAM adaptive learning rate scheme (Kingma & Ba, 2014) (followed by a Polyak average; Polyak & Juditsky, 1992, for the subword experiments) yielded the most consistent and near-optimal performance. We therefore use this setup for optimization

[6] Note that the bias term has been omitted for clarity.

of parameters for all models (including baselines) unless otherwise mentioned. For all experiments, we unroll computation graphs T steps in time (where T varies across experiments and tasks), and, in order to approximate full backpropagation through time, we carry over the last hidden from the previous mini-batch (within a full sequence). More important, we found that by using the derivative of the loss with respect to the last hidden state, we can improve the approximation and thus perform one step of iterative inference to update the last hidden state carried over.[7] We ultimately used this proposed improved approximation for the subword models (since in those experiments, we could directly train all baseline and proposed models in a controlled, identical fashion to ensure fair comparison).

For all Delta-RNNs experimented with in this letter, the output activation of the inner function $g(\cdot)$ was chosen to be the hyperbolic tangent. The output activation of the outer function $f(\cdot)$ was set to be the identity for the word and character benchmark experiments and the hyperbolic tangent for the subword experiments (these decisions were made based on preliminary experimentation on subsets of the final training data). The exact configuration of the implementation we used in this letter involved using the late-integration form—either the unregularized (see equation 2.9) or the dropout regularized (see equation 2.12) variant, for the outer function and equation 2.11.

We compare our proposed models against a wide variety of unregularized baselines, as well as several state-of-the-art regularized baselines for the benchmark experiments. These baselines include the LSTM, GRU, and SCRN, as well as computationally more efficient formulations of each, such as the MGU. The goal is to see if our proposed Delta-RNN is a suitable replacement for complex gated architectures and can capture longer-term patterns in sequential text data.

3 Related Work: Recovering Previous Models

A contribution of this work is that our general framework, presented in section 2.1, offers a way to unify previous proposals for gated neural architectures (especially for use in next-step prediction tasks like language modeling) and explore directions of improvement. Since we will ultimately compare our proposed Delta-RNN of section 2.3 to these architectures, we next present how to derive several key architectures from our general form, such as the gated recurrent unit and the long short-term memory. More important, we introduce them in the same notation and design as the Delta-RNN and highlight the differences between previous work and our own through the lens of $f_\psi(\cdot)$ and $g_\theta(\mathbf{x}_t, \mathbf{M}_{t-1})$.

[7] We searched the step-size λ over the values $\{0.05, 0.1, 0.15\}$ for all experiments in this letter.

Simple models, largely based on the original Elman RNN (Elman, 1990), have often been shown to perform quite well in language modeling tasks (Mikolov, Karafiát, Burget, Černocký, & Khudanpur, 2010; Mikolov, Kombrink, Burget, Černocký, & Khudanpur, 2011). The structurally constrained recurrent network (SCRN: Mikolov et al., 2014), an important predecessor and inspiration for this work, showed that one fruitful path to learning longer-term dependencies was to impose a hard constraint on how quickly the values of hidden units could change, yielding more "stable" long-term memory. The SCRN itself is very similar to a combination of the RNN architectures of Jordan (1990) and Mozer (1993). The key element of its design is the constraint that part of the recurrent weight matrix must stay close to the identity, a constraint that is also satisfied by the Delta-RNN. These identity connections (and corresponding context units that use them) allow for improved information travel over many time steps and can even be viewed as an exponential trace memory (Mozer, 1993). Residual networks, though feedforward in nature, also share a similar motivation (He, Zhang, Ren, & Sun, 2016). Unlike the SCRN, the proposed Delta-RNN does not require a separation of the slow- and fast-moving units, but instead models this slower timescale through implicitly stable states.

The long short-term memory (LSTM; Hochreiter & Schmidhuber, 1997a) is arguably the currently most popular and often-used gated neural architecture, especially in the domain of natural language processing. Starting from our general form, equation 2.1, we can see how the LSTM can be deconstructed, where setting $\mathbf{c}_t = g_\theta(\mathbf{x}_t, \mathbf{M}_{t-1})$ yields

$$\begin{aligned}
\mathbf{h}_t &= f_\psi[g_\theta(\mathbf{x}_t, \mathbf{M}_{t-1}), \mathbf{M}_{t-1}], \\
\mathbf{h}_t &= \mathbf{r}_t \otimes \Phi(\mathbf{c}_t), \text{ where,} \qquad (3.1) \\
\mathbf{r}_t &= \sigma(W_r \mathbf{x}_t + V_r \mathbf{h}_{t-1} + U_r \mathbf{c}_t + \mathbf{b}_r), \qquad (3.2)
\end{aligned}$$

where $\mathbf{M}_{t-1} = \{\mathbf{h}_{t-1}, \mathbf{c}_{t-1}\}$, noting that $\mathbf{c}_{t-1}$ is the cell state designed to act as the constant error carousel in mitigating the problem of vanishing gradients when using backpropagation through time. A great deal of recent work has attempted to improve the training of the LSTM, often by increasing its complexity, such as through the introduction of so-called peephole connections (Gers & Schmidhuber, 2000). To compute $\mathbf{c}_t = g_\theta(\mathbf{x}_t, \mathbf{M}_{t-1})$ using peephole connections, we use the following set of equations:

$$\begin{aligned}
\mathbf{c}_t &= \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \mathbf{z}_t, \text{ where,} \\
\mathbf{z}_t &= \Phi(W_z \mathbf{x}_t + V_z \mathbf{h}_{t-1} + \mathbf{b}_z), \\
\mathbf{i}_t &= \sigma(W_i \mathbf{x}_t + V_i \mathbf{h}_{t-1} + U_i \mathbf{c}_{t-1} + \mathbf{b}_i), \\
\mathbf{f}_t &= \sigma(W_f \mathbf{x}_t + V_f \mathbf{h}_{t-1} + U_f \mathbf{c}_{t-1} + \mathbf{b}_f).
\end{aligned}$$

The gated recurrent unit (GRU; Chung et al., 2014, 2015) can be viewed as one of the more successful attempts to simplify the LSTM. We see that $f_\psi(\cdot)$ and $g_\theta(\cdot)$ are still quite complex, requiring many intermediate computations to reach an output. In the case of the outer mixing function, $f_\psi(\cdot)$, we see that

$$\mathbf{h}_t = \Phi(\gamma[g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}) + \beta\mathbf{h}_{t-1}),$$
$$\gamma = \mathbf{r}_t \text{ and } \beta = (1 - \mathbf{r}_t), \text{ where,} \tag{3.3}$$
$$\mathbf{r}_t = \sigma(V_r\mathbf{h}_{t-1} + W_r\mathbf{x}_t + \mathbf{b}_r), \tag{3.4}$$

noting that the state gate $\mathbf{r}_t$ is also a function of the RNN's previous hidden state and introduces parameters specialized for $\mathbf{r}$. In contrast, the Delta-RNN does not use an extra set of input-to-hidden weights, and, more directly, the preactivation of the input projection can be reused for the interpolation gate. The inner function of the GRU, $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$, is defined as

$$g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}) = \phi(V_h(\mathbf{q}_t \otimes \mathbf{h}_{t-1}) + W_h\mathbf{x}_t + \mathbf{b}_h),$$
$$\mathbf{q}_t = \sigma(V_q\mathbf{h}_{t-1} + W_q\mathbf{x}_t + \mathbf{b}_q),$$

where $\phi()$ is generally set to be the hyperbolic tangent activation function. A mutated architecture (MUT; Jozefowicz, Zaremba, & Sutskever, 2015) was an attempt to simplify the GRU somewhat, as, much like the Delta-RNN, its interpolation mechanism is not a function of the previous hidden state but is still largely as parameter heavy as the GRU, shedding only a single extra parameter matrix, especially since its interpolation mechanism retains a specialized parameter matrix to transform the data. The Delta-RNN shares this with its primary calculation of the data's preactivation values. The minimally gated unit (MGU; Zhou, Wu, Zhang, & Zhou, 2016) is yet a further attempt to reduce the complexity of the GRU by merging its reset and update gates into a single forget gate, essentially using the same outer function under the GRU defined in equation 3.4, but simplifying the inner function $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$ to be quite close to the Elman-RNN but conditioned on the forget gate as follows:

$$g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}) = \phi(V_h(\mathbf{r}_t \otimes \mathbf{h}_{t-1}) + W_h\mathbf{x}_t + \mathbf{b}_h).$$

While the MGU certainly does reduce the number of parameters, viewing it from the perspective of our general Delta-RNN framework, one can see that it still largely uses a $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$ that is rather limited (only the capabilities of the Elman-RNN). The most effective version of our Delta-RNN emerged from the insight that a more powerful $g_\theta(\mathbf{x}_t, \mathbf{h}_{t-1})$ could be obtained by (approximately) increasing its order, which requires a few more bias parameters, and nesting it within a nonlinear interpolation mechanism that will

compute the delta states. Our framework is general enough to also allow designers to incorporate functions that augment the general state engine with an external memory to create architectures that can exploit the strengths of models with decoupled memory architectures (Weston, Chopra, & Bordes, 2014; Sukhbaatar, Szlam, Weston, & Fergus, 2015; Graves et al., 2016) or data structures that serve as memory (Sun, Giles, & Chen, 1998; Joulin & Mikolov, 2015).

A final related, but important, strand of work uses depth (i.e., number of processing layers) to directly model various timescales, as emulated in models such as the hierarchical/multiresolutional recurrent neural network (HM-RNN) (Chung, Ahn, & Bengio, 2016). Since the Delta-RNN is designed to allow its interpolation gate $\mathbf{r}$ to be driven by the data, it is possible that the model might already be learning how to make use of boundary information (word boundaries at the character or subword level; sentence boundaries as marked by punctuation at the word level). The HM-RNN, however, more directly attacks this problem by modifying an LSTM to learn how to manipulate its states when certain types of symbols are encountered. (This is different from models like the clockwork RNN that require explicit boundary information; Koutnik, Greff, Gomez, & Schmidhuber, 2014.) One way to take advantage of the ideas behind the HM-RNN would be to manipulate the DSF to incorporate the explicit modeling of timescales through layer depth (each layer is responsible for modeling a different timescale). Furthermore, it would be worth investigating how the HM-RNN's performance would change when built from modifying a Delta-RNN instead of an LSTM.

4 Experimental Results

Language modeling is an incredibly important next-step prediction task, with applications in downstream applications in speech recognition, parsing, and information retrieval. As such, we will focus this letter on experiments on this task domain to gauge the efficacy of our Delta-RNN framework, noting that this framework might prove useful in, for instance, machine translation (Bahdanau, Cho, & Bengio, 2014) or light chunking (Turian, Bergstra, & Bengio, 2009). Beyond improving language modeling performance, the sentence (and document) representations iteratively inferred by our architectures might also prove useful in composing higher-level representations of text corpora, a subject we will investigate in future work.

4.1 Data Sets

4.1.1 Penn Treebank Corpus. The Penn Treebank corpus (Marcus, Marcinkiewicz, & Santorini, 1993) is often used to benchmark both word- and character-level models via perplexity or bits per character, and thus we

start here.[8] The corpus contains 42,068 sentences (971,657 tokens; average token length of about 4.727 characters) of varying length (the range is from 3 to 84 tokens at the word level).

4.1.2 IMDB Corpus. The large sentiment analysis corpus (Maas et al., 2011) is often used to benchmark algorithms for predicting the positive or negative tonality of documents. However, we opt to use this large corpus (training consists of 149,714 documents, 1,875,523 sentences, 40,765,697 tokens with average token length about 3.4291415 characters) to evaluate our proposed Delta-RNN as a (subword) language model. The IMDB data set serves as a case when the context extends beyond the sentence level in the form of actual documents.

4.2 Word and Character-Level Benchmark. The first set of experiments allows us to examine our proposed Delta-RNN models against reported state-of-the-art models. These reported measures have been on traditional word- and character-level language modeling tasks: we measure the per symbol perplexity of models. For the word-level models, we calculate the per word perplexity (PPL) using the measure $PPL = \exp[-(1/N)\sum_{i=1}^{N}\sum_{t=1}^{T}\log P_\Theta(w_t|\mathbf{h})]$. For the character-level models, we report the standard bits per character (BPC), which can be calculated from the log likelihood using the formula $BPC = -1/(N\log(2))$ $\sum_{i=1}^{N}\sum_{t=1}^{T}\log P_\Theta(w_t|\mathbf{h})$.

Over 100 epochs, word-level models were traned with mini-batches of 64 (padded) sequences. (Early stopping with a look-ahead of 10 was used.) Gradients were clipped using a simple magnitude-based scheme (Pascanu, Mikolov, & Bengio, 2013), with the magnitude threshold set to 5. A simple grid search was performed to tune the learning rate, $\lambda = \{0.002, 0.001, 0.0005, 0.0002\}$, as well as the size of the hidden layer $H = \{500, 1000, 1500\}$. Parameters (non biases) were initialized from zero-mean gaussian distributions with variance tuned, $\sigma = \{0.1, 0.01, 0.005, 0.001\}$.[9] The character-level models were updated using mini-batches of 64 samples over 100 epochs. (Early stopping with a look-ahead of 10 was used.) The parameter initializations and grid search for the learning rate and hidden

[8] To be directly comparable with previously reported results, we make use of the specific preprocessed train/valid/test splits found at http://www.fit.vutbr.cz/imikolov/rnnlm/.

[9] We also experimented with other initializations, most notably the identity matrix for the recurrent weight parameters as in Le, Jaitly, and Hinton (2015). We found that this initialization often worsened performance. For the activation functions of the first-order models, we experimented with the linear rectifier, the parameterized linear rectifier, and even our own proposed parameterized smoothened linear rectifier, but we found that such activations lead to less-than-satisfactory results. The results of this inquiry are documented in the code that will accompany this letter.

layer size were the same as for the word models, with the exception of the hidden layer size, which was searched over $H = \{500, 1000, 1500, 2000\}$.[10]

A simple learning rate decay schedule was employed: if the validation loss did not decrease after a single epoch, the learning rate was halved (unless a lower bound on the value had been reached). When dropout was applied to the Delta-RNN (*Delta-RNN-drop*), we set the probability of dropping a unit to $p_{drop} = 0.15$ for the character-level models and $p_{drop} = 0.5$ for the word-level models. We present the results for the unregularized and regularized versions of the models. For all of the Delta-RNNs, we furthermore experiment with two variations of dynamic evaluation, which facilitates fair comparison to compression algorithms, inspired by the improvements observed in Mikolov (2012). *Delta-RNN-drop, dynamic #1* refers to simply updating the model sample-by-sample after each evaluation, where in this case, we update parameters using simple stochastic gradient descent (Mikolov, 2012), with a step-size $\lambda = 0.005$. We develop a second variation of dynamic evaluation, *Delta-RNN-drop, dynamic #2*, where we allow the model to first iterate (and update) once over the validation set and then finally the test set, completely allowing the model to compress the Penn Treebank corpus. These two schemes are used for both the word- and character-level benchmarks. It is important to stress that the BPC and PPL measures reported for the dynamic models follow a strict "test-then-train" online paradigm, meaning that each next-step prediction is made before updating model parameters.

The standard vocabulary for the word-level models contains 10,000 unique words (including an unknown token for out-of-vocabulary symbols and an end-of-sequence token),[11] and the standard vocabulary for the character-level models includes 49 unique characters (including a symbol for spaces). Results for the word-level and character-level models are reported in Table 1.

4.3 Subword Language Modeling. We chose to measure the negative log likelihood of the various architectures in the task of subword modeling. Subwords are particularly appealing not only in that the input distribution is of lower dimensionality but, as evidenced by the positive results of Mikolov et al. (2012), subword/character hybrid language models improve over the performance of pure character-level models. Subword models also enjoy the advantage held by character-level models when it comes to handling out-of-vocabulary words, avoiding the need for an "unknown" token. Research in psycholinguistics has long suggested that even human infants are sensitive to word boundaries at an early stage (e.g., Aslin, Saffran, &

[10] Note that $H = 2000$ would yield nearly 4 million parameters, our upper bound on total number of parameters allowed for experiments in order to be commensurable with the work of Wu et al. (2016), who actually used $H = 2048$ for all Penn Treebank models.

[11] We use a special null token (or zero vector) to mark the start of a sequence.

Table 1: Test Set Results on the Penn Treebank Word-Level and Character-Level Language Modeling Tasks.

Penn Treebank: Word Models	PPL
N-Gram (Mikolov et al., 2014)	141
NNLM (Mikolov, 2012)	140.2
N-Gram+cache (Mikolov et al., 2014)	125
RNN (Gulcehre et al., 2016)	129
RNN (Mikolov, 2012)	124.7
LSTM (Mikolov et al., 2014)	115
SCRN (Mikolov et al., 2014)	115
LSTM (Sundermeyer, 2016)	107
MI-RNN (Wu et al., 2016, our implementation)	109.2
Delta-RNN (present work)	100.324
Delta-RNN, dynamic #1 (present work)	93.296
Delta-RNN, dynamic #2 (present work)	90.301
LSTM-recurrent drop (Krueger et al., 2016)	87.0
NR-dropout (Zaremba et al., 2014)	78.4
V-dropout (Gal & Ghahramani, 2016)	73.4
Delta-RNN-drop, static (present work)	84.088
Delta-RNN-drop, dynamic #1 (present work)	79.527
Delta-RNN-drop, dynamic #2 (present work)	78.029
Penn Treebank: Character Models	**BPC**
N-discount N-gram (Mikolov et al., 2012)	1.48
RNN+stabilization (Krueger et al., 2016)	1.48
linear MI-RNN (Wu et al., 2016)	1.48
Clockwork RNN (Koutnik et al., 2014)	1.46
RNN (Mikolov et al., 2012)	1.42
GRU (Jernite, Grave, Joulin, & Mikolov, 2016)	1.42
HF-MRNN (Mikolov et al., 2012)	1.41
MI-RNN (Wu et al., 2016)	1.39
Max-Ent N-gram (Mikolov et al., 2012)	1.37
LSTM (Krueger et al., 2016)	1.356
Delta-RNN (present work)	1.347
Delta-RNN, dynamic #1 (present work)	1.331
Delta-RNN, dynamic #2 (present work)	1.326
LSTM-norm stabilizer (Krueger et al., 2016)	1.352
LSTM-weight noise (Krueger et al., 2016)	1.344
LSTM-stochastic depth (Krueger et al., 2016)	1.343
LSTM-recurrent drop (Krueger et al., 2016)	1.286
RBN (Cooijmans, Ballas, Laurent, Gülçehre, & Courville, 2016)	1.32
LSTM-zone out (Krueger et al., 2016)	1.252
H-LSTM + LN (Ha, Dai, & Le, 2016)	1.25
TARDIS (Gulcehre et al., 2017)	1.25
3-HM-LSTM + LN (Chung et al., 2016)	1.24
Delta-RNN-drop, static (present work)	1.251
Delta-RNN-drop, dynamic #1 (present work)	1.247
Delta-RNN-drop, dynamic #2 (present work)	1.245

Table 2: Test Set Negative Log Likelihoods While Holding Number of Parameters Approximately Constant.

	Performance			Performance	
PTB-SW	Number of Parameters	NLL	IMDB-SW	Number of Parameters	NLL
RNN	1,272,464	1.8939	RNN	499,176	2.1691
SCRN	1,268,604	1.8420	SCRN	496,196	2.2370
MGU	1,278,692	1.8694	MGU	495,444	2.1312
MI-RNN	1,267,904	1.8441	MI-RNN	495,446	2.1741
GRU	1,272,404	1.8251	GRU	499,374	2.1551
LSTM	1,274,804	1.8412	LSTM	503,664	2.2080
Delta-RNN	1,268,154	1.8260	Delta-RNN	495,570	2.1333

Note: Subword modeling tasks on Penn Treebank and IMDB.

Newport, 1998), and that morphologically complex words enjoy dedicated processing mechanisms (Baayen & Schreuder, 2006). Subword-level language models may approximate such an architecture. Consistency in subword formation is critical in order to obtain meaningful results (Mikolov et al., 2012). Thus, we design our subword algorithm to partition a word according to the following scheme:

1. Split on vowels (using a predefined list).
2. Link or merge each vowel with a consonant to the immediate right if applicable.
3. Merge straggling single characters to subwords on the immediate right unless a subword of shorter character length is to the left.

This simple partitioning scheme was designed to ensure that no subword was shorter than two characters in length. Future work will entail designing a more realistic subword partitioning algorithm. Subwords below a certain frequency were discarded and combined with 26 single characters to create the final dictionary. For Penn Treebank, this yields a vocabulary of 2405 symbols (2378 subwords + 26 characters + 1 end token). For the IMDB corpus, after replacing all emoticons and special nonword symbols with special tokens, we obtain a dictionary of 1926 symbols (1899 subwords + 26 single characters + 1 end token). Results for all subword models are reported in Table 2.

Specifically, we test our implementations of the LSTM[12] (with peephole connections as described in Graves, 2013), the GRU, the MGU, the SCRN, and a classical Elman network, of both first and second order (Giles et al.,

[12] We experimented with initializing the forget gate biases of all LSTMs with values searched over $\{1, 2, 3\}$ since previous work has shown this can improve model performance.

1991; Wu et al., 2016).[13] Subword models were trained in a similar fashion as the character-level models, updated (every 50 steps) using mini-batches of 20 samples but over 30 epochs. Learning rates were tuned in the same fashion as the word-level models, and the same parameter initialization schemes were explored. The notable difference between this experiment and the previous ones is that we fix the number of parameters for each model to be equivalent to that of an LSTM with 100 hidden units for PTB and 50 hidden units for IMDB. This ensures a controlled, fair comparison across models and allows us to evaluate if the Delta-RNN can learn similar to models with more complicated processing elements (an LSTM cell versus a GRU cell versus a Delta-RNN unit). Furthermore, this allows us to measure parameter efficiency, where we can focus on the value of actual specific cell types (e.g., allowing us to compare the value of a much more complex LSTM memory unit versus a simple Delta-RNN cell) when the number of parameters is held roughly constant. We are currently running larger versions of the models depicted Table 2 to determine if the results hold at scale.

5 Discussion

With respect to the word- and character-level benchmarks, we see that the Delta-RNN outperforms all previous, unregularized models and performs comparably to regularized state-of-the-art. As documented in Table 2, we further trained a second-order, word-level RNN (MI-RNN) to complete the comparison and note that the second-order connections appear to be quite useful in general, outperforming the SCRN and coming close to that of the LSTM. This extends the results of Wu et al. (2016) to the word level. However, the Delta-RNN, which also makes use of second-order units within its inner function, ultimately offers the best performance and performs better than the LSTM in all experiments. In both Penn Treebank and IMDB subword language modeling experiments, the Delta-RNN is competitive with complex architectures such as the GRU and the MGU. In both cases, the Delta-RNN nearly reaches the same performance as the best-performing baseline model in either data set (i.e., it nearly reaches the same performance as the GRU on Penn Treebank and the MGU on IMDB). Surprisingly, on IMDB, a simple Elman network performs quite well, even outperforming the MI-RNN. We argue that this might be the result of constraining all neural architectures to only a small number of parameters for such a large data set, a constraint we intend to relax in future work.

[13] Code to build and train the architectures in this study can be found online at http://github.com/ago109/Delta-RNN-Theano.git.

The Delta-RNN is far more efficient than a complex LSTM and certainly a memory-augmented network like TARDIS (Gulcehre, Chander, & Bengio, 2017). Moreover, it appears to learn how to make appropriate use of its interpolation mechanism to decide how and when to update its hidden state in the presence of new data.[14] Given our derivations in section 3, one could argue that nearly all previously proposed gated neural architectures are essentially trying do the same thing under the DSF. The key advantage offered by the Delta-RNN is that this functionality is offered directly and cheaply (in terms of required parameters).

It is important to contrast these (unregularized) results with those that use some form of regularization. Zaremba, Sutskever, and Vinyals (2014) reported that a single LSTM (for word-level Penn Treebank) can reach a PPL of about 80, but this was achieved via dropout regularization (Srivastava et al., 2014). There is a strong relationship between using dropout and training an ensemble of models. Thus, one can argue that a single model trained with dropout actually is not a single model, but an implicit ensemble (see also Srivastava et al., 2014). An ensemble of 20 simple RNNs and cache models previously did reach PPL as low as 72, while a single RNN model gives only 124 (Mikolov, 2012). Zaremba et al. (2014) trained an ensemble of 38 LSTMs regularized with dropout, each with 100 times more parameters than the RNNs used by Mikolov (2012), achieving PPL 68. This is arguably a small improvement over 72 and seems to strengthen our claim that dropout is an implicit model ensemble and thus should not be used when one wants to report the performance of a single model. However, the Delta-RNN is amenable to regularization, including dropout. As our results show, when simple dropout is applied, the Delta-RNN can reach much lower perplexities, even similar to the state of the art with much larger models, especially when dynamic evaluation is permitted. This even extends to very complex architectures, such as the recently proposed TARDIS, which is a memory-augmented network (and when dynamic evaluation is used, the simple Delta-RNN can outperform this complex model). Though we investigate the utility of simple dropout in this letter, our comparative results suggest that more sophisticated variants such as variational dropout (Gal & Ghahramani, 2016) could yield yet further improvement in performance.

What is the lesson to be learned from the DSF? First and foremost, we can obtain strong performance in language modeling with a simpler, more efficient (in terms of number of parameters), and thus faster architecture. Second, the Delta-RNN is designed from the interpretation that the computation of the next hidden state is the result of a composition of two functions. One inner function decides how to "propose" a new hidden state

[14] At greater computational cost, a somewhat lower perplexity for an LSTM may be attainable, such as the perplexity of 107 reported by Sundermeyer (2016) (see Table 1). However, this requires many more training epochs and precludes batch training.

while the outer function decides how to use this new proposal in updating the previously calculated state. The data-driven interpolation mechanism is used by the model to decide how much impact the newly proposed state has in updating what is likely to be a slowly changing representation. The SCRN, which could be viewed as the predecessor to the Delta-RNN framework, was designed with the idea that some constrained units could serve as a sort of cache meant to capture longer-term dependencies. Like the SCRN, the Delta-RNN is designed to help mitigate the problem of vanishing gradients and, through the interpolation mechanism, has multiple pathways through which the gradient might be carried, boosting the error signal's longevity down the propagation path through time. However, the SCRN combines the slow-moving and fast-changing hidden states through a simple summation and thus cannot model nonlinear interactions between its shorter- and longer-term memories, furthermore requiring tuning of the sizes of these separated layers. On the other hand, the Delta-RNN, which does not require special tuning of an additional hidden layer, can nonlinearly combine the two types of states in a data-dependent fashion, possibly allowing the model to exploit boundary information from text, which is quite powerful in the case of documents. The key intuition is that the gating mechanism allows the state proposal to affect the maintained memory state only if the currently observed data point carries any useful information. This warrants a comparison, albeit indirect, to surprisal theory. This "surprisal" proves useful in iteratively forming a sentence impression that will help to better predict the words that come later.

With respect to the last point made, we briefly examine the evolution of a trained Delta-RNN's hidden state across several sample sentences. The first two sentences are hand-created (constrained to use only the vocabulary of Penn Treebank), and the last one is sampled from the Penn Treebank training split. Since the Delta-RNN iteratively processes symbols of an ordered sequence, we measure the L1 norm across consecutive pairs of hidden states. We report the (min-max) normalized L1 scores[15] in Figure 2 and observe that in accordance with our intuition, we can see that the L1 norm is lower for high-frequency words (indicating a smaller delta) such as *the* or *of* or *is*, which are words generally less informative about the general subject of a sentence or document. As this qualitative demonstration illustrates, the Delta-RNN appears to learn what to do with its internal state in the presence of symbols of variable information content.

[15] If we calculate the L1 norm, or Manhattan distance, for every contiguous pair of state vectors across a sequence of length T and $\mathbf{h}_0$ is the state calculated for the start or null token, we obtain the sequence of L1 measures $L1_{seq} = \{L1_0(\mathbf{h}_0, \mathbf{h}_1), \ldots, L1_T(\mathbf{h}_{T-1}, \mathbf{h}_T)\}$ (the L1 for the start token is simply excluded). Calculating the score for any $\mathbf{h}_t$ ($t \in T$) is then as simple as performing min-max normalization, or $L1_{score} = (L1(\mathbf{h}_{t-1}, \mathbf{h}_t) - \min(L1_{seq}))/(\max(L1_{seq}) - \min(L1_{seq}))$.

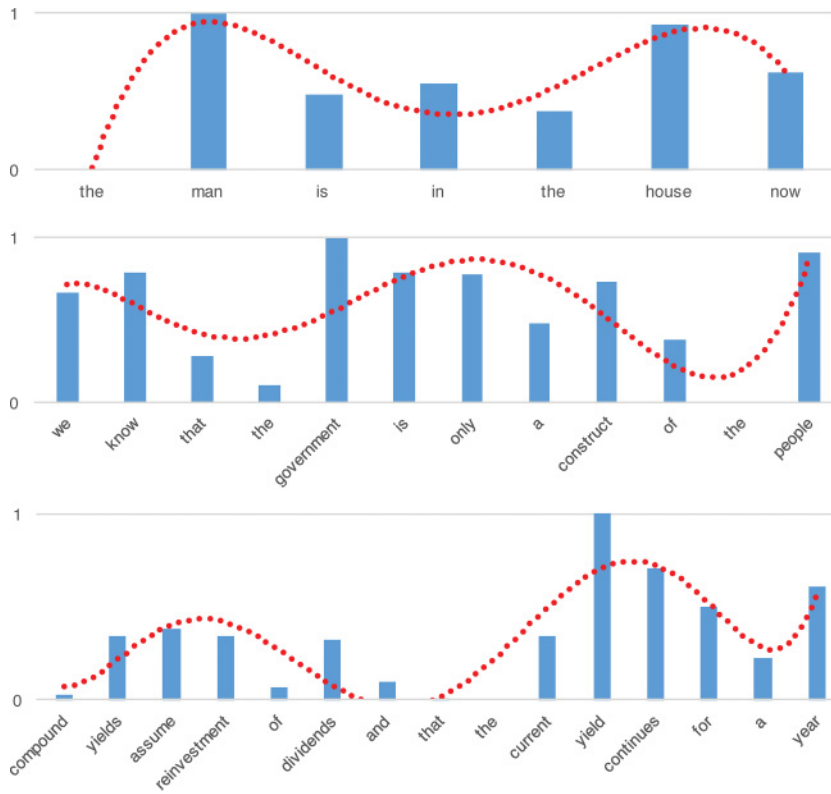

Figure 2: L1 norm of deltas between consecutive states of model trained on Penn Treebank plotted over words of example sentences. A simple polynomial trend line (dashed red) was fit to the bar heights in order to illustrate the informative bumps of each sample sentence. The main observation is that the norm is in general lower for low-information content words, such as the article *the*, and higher for informative words, such as *government*.

6 Conclusion

We present the differential state framework, which affords us a useful perspective on viewing computation in recurrent neural networks. Instead of recomputing the whole state from scratch at every time step, the Delta-RNN learns only how to update the current state. This seems to be better suited for many types of problems, especially those that involve longer-term patterns where part of the recurrent network's state should be constant most of the time. Comparison to the currently widely popular LSTM and GRU architectures shows that the Delta-RNN can achieve similar or better

performance on language modeling tasks while being conceptually much simpler and with far fewer parameters. Comparison to the structurally constrained recurrent network (SCRN), which shares many of the main ideas and motivation, shows better accuracy and a simpler model architecture (in the SCRN, tuning the sizes of two separate hidden layers is required, and this model cannot learn nonlinear interactions within its longer memory).

Future work includes larger-scale language modeling experiments to test the efficacy of the Delta-RNN, as well as architectural variants that employ decoupled memory. Since the Delta-RNN can also be stacked just like any other neural architecture, we intend to investigate if depth (in terms of hidden layers) might prove useful on larger-scale data sets. In addition, we intend to explore how useful the Delta-RNN might be in other tasks that the architectures such as the LSTM currently hold state-of-the-art performance in. Finally, it would be useful to explore if the Delta-RNN's simpler, faster design can speed up the performance of grander architectures, such as the differentiable neural computer (Graves et al., 2016), which is largely made up of multiple LSTM modules.

Appendix: Layer Normalized Delta-RNNs

In this appendix, we describe how layer normalization would be applied to a Delta-RNN. Though our preliminary experiments did not uncover that layer normalization gave much improvement over dropout, this was observed only on the Penn Treebank benchmark. Future work will investigate the benefits of layer normalization over dropout (as well as model ensembling) on larger-scale benchmarks.

A simple RNN requires the layer normalization to be applied after calculating the full linear preactivation (a sum of the filtration and the projected data point). A Delta-RNN requires further care (like the GRU) to ensure the correct components are normalized without damaging the favorable properties inherent in the model's multiplicative gating. If layer normalization is applied to the preactivations of the late-integration Delta-RNN proposed in this letter, the update equations become

$$\mathbf{d}_t^{rec} = LN(V_d \mathbf{h}_{t-1}),\ \mathbf{d}_t^{dat} = LN(W \mathbf{x}_t), \tag{A.1}$$

$$\mathbf{z}_t = \phi_{hid}(\mathbf{d}_t^{rec} \otimes \mathbf{d}_t^{dat} + \mathbf{d}_t^{rec} + \mathbf{d}_t^{dat}), \tag{A.2}$$

$$\mathbf{h}_t = \Phi((1 - \mathbf{r}) \otimes \mathbf{z}_t + \mathbf{r} \otimes \mathbf{h}_{t-1}), \tag{A.3}$$

$$\mathbf{r} = 1/(1 + \exp(-[\mathbf{d}_t^{dat} + \mathbf{b}_r])). \tag{A.4}$$

Note that the additional bias parameters introduced in the original update equations are now omitted. This can be done since the layer normalization operation applied will now perform the work of shifting and scaling. Since the Delta-RNN takes advantage of parameter sharing, it notably requires

substantially fewer layer normalizations than a more complex model (such as the GRU) would. A standard GRU would require nine layer normalizations while the Delta-RNN simply requires two.

Acknowledgments

We thank C. Lee Giles and Prasenjit Mitra for their advice. We thank NVIDIA for providing GPU hardware that supported this letter. A.O. was funded by a NACME-Sloan scholarship; D.R. acknowledges funding from NSF IIS-1459300.

References

Aslin, R. N., Saffran, J. R., & Newport, E. L. (1998). Computation of conditional probability statistics by 8-month-old infants. *Psychological Science*, *9*(4), 321–324.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). *Layer normalization*. arXiv:1607.06450.

Baayen, R. H., & Schreuder, R. (2006). *Morphological processing*. Hoboken, NJ: Wiley.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). *Neural machine translation by jointly learning to align and translate*. arXiv:1409.0473.

Boston, M. F., Hale, J., Kliegl, R., Patil, U., & Vasishth, S. (2008). Parsing costs as predictors of reading difficulty: An evaluation using the Potsdam Sentence Corpus. *Journal of Eye Movement Research*, *2*(1).

Choudhury, V. (2015). *Thought vectors: Bringing common sense to artificial intelligence*. www.iamwire.com.

Chung, J., Ahn, S., & Bengio, Y. (2016). *Hierarchical multiscale recurrent neural networks*. arXiv:1609.01704.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical evaluation of gated recurrent neural networks on sequence modeling*. arXiv:1412.3555.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2015). Gated feedback recurrent neural networks. In *Proceedings of the International Conference on Machine Learning* (pp. 2067–2075).

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., & Courville, A. (2016). *Recurrent batch normalization*. arXiv:1603.09025.

Das, S., Giles, C. L., & Sun, G.-Z. (1992). Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society* (p. 14). San Mateo, CA: Morgan Kaufmann.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*(2), 179–211.

Gal, Y., & Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems, 29* (pp. 1019–1027). Red Hook, NY: Curran.

Gers, F. A., & Schmidhuber, J. (2000). Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks* (vol. *3*, pp. 189–194). Piscataway, NJ: IEEE.

Giles, C. L., Chen, D., Miller, C., Chen, H., Sun, G., & Lee, Y. (1991). Second-order recurrent neural networks for grammatical inference. In *Proceedings of the International Joint Conference on Neural Networks* (vol. *2*, pp. 273–281). Piscataway, NJ: IEEE.

Giles, C. L., Lawrence, S., & Tsoi, A. C. (2001). Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine Learning*, *44*(1–2), 161–183.

Giles, C. L., Miller, C. B., Chen, D., Chen, H.-H., Sun, G.-Z., & Lee, Y.-C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, *4*(3), 393–405.

Goudreau, M. W., Giles, C. L., Chakradhar, S. T., & Chen, D. (1994). First-order versus second-order single-layer recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(3), 511–513.

Graves, A. (2013). *Generating sequences with recurrent neural networks*. arXiv:1308.0850.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., . . . Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, *538*(7626), 471–476.

Gulcehre, C., Chander, S., & Bengio, Y. (2017). *Memory augmented neural networks with wormhole connections*. arXiv:1701.08718.

Gulcehre, C., Moczulski, M., Denil, M., & Bengio, Y. (2016). *Noisy activation functions*. arXiv:1603.00391.

Ha, D., Dai, A., & Le, Q. V. (2016). *Hypernetworks*. arXiv:1609.09106.

Hale, J. (2001). A probabilistic Earley parser as a psycholinguistic model. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics* (pp. 1–8). Stroudsburg, PA: ACL.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. In *Proceedings of the European Conference on Computer Vision* (pp. 630–645). New York: Springer.

Hochreiter, S., & Schmidhuber, J. (1997a). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hochreiter, S., & Schmidhuber, J. (1997b). LTSM can solve hard time lag problems. In M. C. Mozer, M. I. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems*, *9* (pp. 473–479). Cambridge, MA: MIT Press.

Ioffe, S., & Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. arXiv:1502.03167.

Jernite, Y., Grave, E., Joulin, A., & Mikolov, T. (2016). *Variable computation in recurrent neural networks*. arXiv:1611.06188.

Jordan, M. I. (1990). Artificial neural networks. In J. Diederich Joachim (Ed.), *Attractor dynamics and parallelism in a connectionist sequential machine* (pp. 112–127). Piscataway, NJ: IEEE Press.

Joulin, A., & Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. In *Proceedings of the 28th International Conference on Neural Information Processing Systems* (pp. 190–198). Cambridge, MA: MIT Press.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning* (pp. 2342–2350).

Kingma, D., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv:1412.6980.

Koutnik, J., Greff, K., Gomez, F., & Schmidhuber, J. (2014). *A clockwork RNN*. arXiv:1402.3511.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., . . . Pal, C. (2016). *Zoneout: Regularizing rnns by randomly preserving hidden activations*. arXiv:1606.01305.

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). *A simple way to initialize recurrent networks of rectified linear units*. arXiv:1504.00941.

Levy, R. (2008). Expectation-based syntactic comprehension. *Cognition*, *106*(3), 1126–1177.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011). Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Stroudsburg, PA: Association for Computational Linguistics.

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, *19*(2), 313–330.

Mikolov, T. (2012). *Statistical language models based on neural networks*. Ph.D. diss., University of Brno, Brno, CZ.

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M., & Ranzato, M. (2014). *Learning longer memory in recurrent neural networks*. arXiv:1412.7753.

Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association* (vol. 2, pp. 1045–1048). International Speech Communication Association.

Mikolov, T., Kombrink, S., Burget, L., Černocký, J., & Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 5528–5531). Piscataway, NJ: IEEE.

Mikolov, T., Sutskever, I., Deoras, A., Le, H.-S., Kombrink, S., & Černocký, J. (2012). Subword language modeling with neural networks. http://www.fit.vutbr.cz/∼imikolov/rnnlm/char.pdf.

Mozer, M. C. (1993). *Neural net architectures for temporal sequence processing*. Reading, MA: Addison-Wesley.

Neal, R. M. (2012). *Bayesian learning for neural networks*. New York: Springer Science & Business Media.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference of Machine Learning* (pp. 1310–1318).

Polyak, B. T., & Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, *30*(4), 838–855.

Serban, I. V., Ororbia, I., Alexander, G., Pineau, J., & Courville, A. (2016). *Piecewise latent variables for neural variational text processing*. arXiv:1612.00377.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

Sukhbaatar, S., Szlam, A., Weston, J., & Fergus, R. (2015). *End-to-end memory networks*. arXiv:1503.08895.

Sun, G.-Z., Giles, C. L., & Chen, H.-H. (1998). The neural network pushdown automaton: Architecture, dynamics and training. In C. L. Giles & M. Gori (Eds.), *Adaptive processing of sequences and data structures* (pp. 296–345). New York: Springer.

Sundermeyer, M. (2016). *Improvements in language and translation modeling*. Ph.D. diss., RWTH Aachen University.

Turian, J., Bergstra, J., & Bengio, Y. (2009). Quadratic features and deep architectures for chunking. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers* (pp. 245–248). Stroudsburg, PA: Association for Computational Linguistics.

Wang, T., & Cho, K. (2015). *Larger-context language modelling*. arXiv:1511.03729.

Weston, J., Chopra, S., & Bordes, A. (2014). *Memory networks*. arXiv:1410.3916.

Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., & Salakhutdinov, R. R. (2016). On multiplicative integration with recurrent neural networks. arXiv:1606.06630.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). *Recurrent neural network regularization*. arXiv:1409.2329.

Zhou, G.-B., Wu, J., Zhang, C.-L., & Zhou, Z.-H. (2016). Minimal gated unit for recurrent neural networks. *International Journal of Automation and Computing*, *13*(3), 226–234.

Received March 24, 2017; accepted July 2, 2017.