Rational Recurrences

Hao Peng[♠] **Roy Schwartz**[♠◇] **Sam Thomson**[♣] **Noah A. Smith**[♠◇]
[♠]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♣]School of Computer Science, Carnegie Mellon University
[◇]Allen Institute for Artificial Intelligence
{hapeng,roysch,nasmith}@cs.washington.edu, sthomson@cs.cmu.edu

Abstract

Despite the tremendous empirical success of neural models in natural language processing, many of them lack the strong intuitions that accompany classical machine learning approaches. Recently, connections have been shown between convolutional neural networks (CNNs) and weighted finite state automata (WFSAs), leading to new interpretations and insights. In this work, we show that some *recurrent* neural networks also share this connection to WFSAs. We characterize this connection formally, defining **rational recurrences** to be recurrent hidden state update functions that can be written as the Forward calculation of a finite set of WFSAs. We show that several recent neural models use rational recurrences. Our analysis provides a fresh view of these models and facilitates devising new neural architectures that draw inspiration from WFSAs. We present one such model, which performs better than two recent baselines on language modeling and text classification. Our results demonstrate that transferring intuitions from classical models like WFSAs can be an effective approach to designing and understanding neural models.

1 Introduction

Neural models, and in particular gated variants of recurrent neural networks (RNNs, e.g., Hochreiter and Schmidhuber, 1997; Cho et al., 2014), have become a core building block for state-of-the-art approaches in NLP (Goldberg, 2016). While these models empirically outperform classical NLP methods on many tasks (Zaremba et al., 2014; Bahdanau et al., 2015; Dyer et al., 2016; Peng et al., 2017, *inter alia*), they typically lack the intuition offered by classical models, making it hard to understand the roles played by each of their components. In this work we show that many neural models are more interpretable than previously thought, by drawing connections to weighted finite state automata (WFSAs). We study several recently proposed RNN architectures and show that one can use WFSAs to characterize their recurrent updates. We call such models **rational recurrences** (§3).[1] Analyzing recurrences in terms of WFSAs provides a new view of existing models and facilitates the development of new ones.

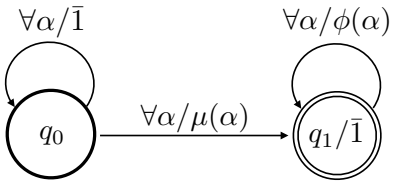

Figure 1: A two-state WFSA $\mathcal{B}$ described in §2. It is closely related to several models studied in this paper (§4.1). Bold circles indicate initial states, and double circles final states, which are associated with final weights. Arrows represent transitions, labeled by the symbols α they consume, and the weights as a function of α. Arcs not drawn are assumed to have weight $\bar{0}$. For brevity, $\forall\alpha$ means $\forall\alpha \in \Sigma$, with Σ being the alphabet.

In recent work, Schwartz et al. (2018) introduced SoPa, an RNN constructed from WFSAs, and thus rational by our definition. They also showed that a single-layer max-pooled CNN (LeCun, 1998) can be simulated by a set of simple WFSAs (one per output dimension), and accordingly are also rational. In this paper we broaden such efforts, and show that rational recurrences are in frequent use (Mikolov et al., 2014; Balduzzi and Ghifary, 2016; Lei et al., 2016, 2017a,b; Bradbury et al., 2017; Foerster et al., 2017). For instance, we will show in §4 that the WFSA diagrammed

[1] Where the term *regular* is used with unweighted FSAs (e.g., regular languages, regular expressions), *rational* is the weighted analog (e.g., rational series, Sakarovitch, 2009; rational kernels, Cortes et al., 2004).

in Figure 1 has strong connections to several of the models mentioned above.

Based on these observations, we then discuss potential approaches to deriving novel neural architectures from WFSAs (§5). As a case study, we present a new model motivated by the interpolation of a two-state WFSA and a three-state one, capturing (soft) unigram and bigram features, respectively. Our experiments show that in two tasks—language modeling and text classification—the proposed model outperforms recently proposed rational models (§6). Further extensions might lead to larger gains, and the rational recurrence view could facilitate easier exploration of such extensions. To promote such exploration, we publicly release our implementation at https://github.com/Noahs-ARK/rational-recurrences.

2 Background: Weighted Finite State Automata (WFSAs)

This section reviews weighted finite-state automata and semirings, which underly our analyses in §3. WFSAs extend nondeterministic *unweighted* finite-state automata by assigning weights to transitions, start states, and final states. Instead of simply accepting or rejecting a string, a WFSA returns a score for the string, and this score summarizes the weights along all paths through the WFSA that consume the string. In order for this summary score to be efficiently computable, weights are taken from a **semiring**.

Definition 1 (Kuich and Salomaa, 1986)**.** A **semiring** is a set $\mathbb{K}$ along with two associative binary operations on $\mathbb{K}$, $\oplus$ (addition) and $\otimes$ (multiplication), and two identity elements: $\bar{0}$ for addition, and $\bar{1}$ for multiplication. Semirings also require that addition is commutative, multiplication distributes over addition, and that multiplication by $\bar{0}$ annihilates (i.e., $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$).

One common semiring is the **real** (or plus-times) semiring: $\langle \mathbb{R}, +, \cdot, 0, 1 \rangle$. The other one used in this work is the **max-plus** semiring $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$. We refer the reader to Kuich and Salomaa (1986) for others.

Definition 2. A **weighted finite-state automaton (WFSA)** over a semiring $\mathbb{K}$ is a 5-tuple, $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \tau, \lambda, \rho \rangle$,[2] with:

- a finite input alphabet Σ;
- a finite state set $\mathcal{Q}$;
- transition weights $\tau : \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\varepsilon\}) \to \mathbb{K}$;
- initial weights $\lambda : \mathcal{Q} \to \mathbb{K}$;
- and final weights $\rho : \mathcal{Q} \to \mathbb{K}$.

$\varepsilon \notin \Sigma$ marks special **ε-transitions** that may be taken without consuming any input. $\mathcal{A}$ assigns a score $\mathcal{A}[\![\mathbf{x}]\!]$ to a string $\mathbf{x} = x_1 \dots x_n \in \Sigma^*$ by summing over the scores of all possible paths deriving $\mathbf{x}$. The score of each individual path is the product of the weights of the transitions it consists of. Formally:

Definition 3 (path score)**.** Let $\boldsymbol{\pi} = \pi_1 \dots \pi_n$ be a sequence of adjacent **transitions** in $\mathcal{A}$, with each transition $\pi_i = (q_i, q_{i+1}, z_i) \in \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\varepsilon\})$. The path $\boldsymbol{\pi}$ **derives** string $\mathbf{x} \in \Sigma^*$, which is the substring of $\mathbf{z} = z_1 z_2 \dots z_n$ that excludes ε symbols (for example, if $\mathbf{z} = a\varepsilon bc\varepsilon\varepsilon\varepsilon d$, then $\mathbf{x} = abcd$). $\boldsymbol{\pi}$'s score in $\mathcal{A}$ is given by

$$\mathcal{A}[\boldsymbol{\pi}] = \lambda(q_1) \otimes \left(\bigotimes_{i=1}^{n} \tau(\pi_i) \right) \otimes \rho(q_{n+1}). \quad (1)$$

Definition 4 (string score)**.** Let $\Pi(\mathbf{x})$ denote the set of all paths in $\mathcal{A}$ that derive $\mathbf{x}$. Then the score assigned by $\mathcal{A}$ to $\mathbf{x}$ is defined to be

$$\mathcal{A}[\![\mathbf{x}]\!] = \bigoplus_{\boldsymbol{\pi} \in \Pi(\mathbf{x})} \mathcal{A}[\boldsymbol{\pi}]. \quad (2)$$

Because $\mathbb{K}$ is a semiring, $\mathcal{A}[\![\mathbf{x}]\!]$ can be computed in time linear in $|\mathbf{x}|$ by the Forward algorithm (Baum and Petrie, 1966). Here, for simplicity, we describe the Forward algorithm without ε-transitions.[3] Its dynamic program is given by:

$$\Omega_0(q) = \lambda(q), \quad (3a)$$

$$\Omega_{i+1}(q) = \bigoplus_{q' \in \mathcal{Q}} \Omega_i(q') \otimes \tau(q', q, x_i), \quad (3b)$$

$$\mathcal{A}[\![\mathbf{x}]\!] = \bigoplus_{q \in \mathcal{Q}} \Omega_n(q) \otimes \rho(q). \quad (3c)$$

$\Omega_i(q)$ gives the total score of all paths that derive $x_1 \dots x_i$ and end in state q.

Example 5. Figure 1 diagrams a WFSA $\mathcal{B}$, consisting of two states. A path starts from the initial state q_0 (with $\lambda(q_0) = \bar{1}$); it then takes any num-

[2] Some authors define τ, λ, and ρ to be partial functions—applying only a subset of transitions, initial states, and final states respectively. Our definition is equivalent, giving the weight functions value $\bar{0}$ wherever they were undefined.

[3] ε-transitions can be handled with a slight modification (Schwartz et al., 2018). Note though that if $\mathcal{A}$ contains a cycle of ε-transitions, then either $\mathbb{K}$ must follow the **star semiring** laws (Kuich and Salomaa, 1986), or the number of consecutive ε-transitions allowed must be capped.

ber of "self-loop" transitions, each consuming an input without changing the path score (since it's weighted by $\bar{1}$); it then consumes an input symbol α and takes a transition weighted by $\mu(\alpha)$, and reaches the final state q_1 (with $\rho(q_1) = \bar{1}$); it may further consume more input by taking self-loops at q_1, updating the path score by multiplying it by $\phi(\alpha)$ for each symbol α. Then from Definition 4, we can calculate that $\mathcal{B}$ gives the empty string score $\bar{0}$, and gives any nonempty string $\mathbf{x} = x_1 \dots x_n \in \Sigma^+$ score $\mathcal{B}[\![\mathbf{x}]\!] =$

$$\bigoplus_{i=1}^{n-1} \left(\mu(x_i) \otimes \bigotimes_{j=i+1}^{n} \phi(x_j) \right) \oplus \mu(x_n). \tag{4}$$

$\mathcal{B}$ can be seen as capturing soft unigram patterns (Davidov et al., 2010), in the sense that it consumes one input symbol to reach the final state from the initial state. It is straightforward to design WFSAs capturing longer patterns by including more states (Schwartz et al., 2018), as we will discuss later in §4 and §5.

3 Rational Recurrences

Before formally defining rational recurrences in §3.2, we highlight the connection between WFSAs and RNNs using a motivating example (§3.1).

3.1 A Motivating Example

We describe a simplified RNN which strips away details of some recent RNNs, in order to highlight the behaviors of the forget gate and the input.

Example 6. For an input sequence $\mathbf{x} = x_1 \dots x_n$, let the word embedding vector for x_t be $\mathbf{v}_t$. As in many gated RNN variants (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), we use a forget gate $\mathbf{f}_t$, which is computed with an affine transformation followed by an elementwise sigmoid function $\boldsymbol{\sigma}$. The current input representation $\mathbf{u}_t$ is similarly computed, but with an optional nonlinearity (e.g., $\tanh$) $\boldsymbol{g}$. The hidden state $\mathbf{c}_t$ can be seen as a weighted sum of the previous state and the new input, controlled by the forget gate.

$$\mathbf{f}_t = \boldsymbol{\sigma}\big(\mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\big), \tag{5a}$$

$$\mathbf{u}_t = (\mathbf{1} - \mathbf{f}_t) \odot \boldsymbol{g}\big(\mathbf{W}_u \mathbf{v}_t + \mathbf{b}_u\big), \tag{5b}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t. \tag{5c}$$

The hidden state $\mathbf{c}_t$ can then be used in downstream computation, e.g., to calculate output state $\mathbf{h}_t = \tanh(\mathbf{c}_t)$, which is then fed to an MLP classifier. We focus only on the recurrent computation.

In Example 6, both $\mathbf{f}_t$ and $\mathbf{u}_t$ depend only on the current input token x_t (through $\mathbf{v}_t$), and not the previous state. Importantly, the interaction with the previous state $\mathbf{c}_{t-1}$ is *not* via affine transformations followed by nonlinearities, as in, e.g., an Elman network (Elman, 1990), where $\mathbf{c}_t = \tanh(\mathbf{W}_c \mathbf{c}_{t-1} + \mathbf{W}_v \mathbf{v}_t + \mathbf{b}_c)$. As we will discuss later, this is important in relating this recurrent update function to WFSAs.

Since the recurrent update in Equation 5c is elementwise, for simplicity we focus on just the ith dimension. Unrolling it in time steps, we get

$$\begin{aligned} [\mathbf{c}_t]_i &= [\mathbf{f}_t]_i \, [\mathbf{c}_{t-1}]_i + [\mathbf{u}_t]_i \\ &= \sum_{j=1}^{t-1} \left([\mathbf{u}_j]_i \prod_{k=j+1}^{t} [\mathbf{f}_k]_i \right) + [\mathbf{u}_t]_i, \end{aligned} \tag{6}$$

where $[\cdot]_i$ denotes the ith dimension of a vector. As noted by Lee et al. (2017), the hidden state at time step t can be seen as a sum of previous input representations, weighted by the forget gate; longer histories typically get a smaller weight, since the forget gate values are between 0 and 1 due to the sigmoid function.

Let's recall the WFSA $\mathcal{B}$ (Figure 1 and Example 5) using the real semiring $\langle \mathbb{R}, +, \cdot, 0, 1 \rangle$. Equation 6 is recovered by parameterizing $\mathcal{B}$'s weight functions μ and ϕ with

$$\mu(x_t) = [\mathbf{u}_t]_i, \quad \phi(x_t) = [\mathbf{f}_t]_i. \tag{7}$$

Denote the resulting WFSA by $\mathcal{B}_i$, and we have:

Proposition 7. *Running a single layer RNN in Example 6 over any nonempty input string $\mathbf{x} \in \Sigma^+$, the ith dimension of its hidden state at time step t equals the score assigned by $\mathcal{B}_i$ to $\mathbf{x}_{:t}$:*

$$[\mathbf{c}_t]_i = \mathcal{B}_i[\![\mathbf{x}_{:t}]\!]. \tag{8}$$

In other words, the ith dimension of the RNN in Example 6 can be seen as a WFSA structurally equivalent to $\mathcal{B}$. Its weight functions are implemented as the ith dimension of Equations 5, and the learned parameters are the ith row of $\mathbf{W}$ and $\mathbf{b}$. Then it is straightforward to recover the full d-dimensional RNN, by collecting d such WFSAs, each of which is parametrized by a row in the $\mathbf{W}$s and $\mathbf{b}$s. Based on this observation, we are now ready to formally define rational recurrences.

3.2 Recurrences and Rationality

For a function $\mathbf{c}\colon \Sigma^* \to \mathbb{K}^d$, its *recurrence* is said to be the dependence of $\mathbf{c}(\mathbf{x}_{:t})$ on $\mathbf{c}(\mathbf{x}_{:t-1})$, for input

sequence $\forall \mathbf{x} \in \Sigma^+$. We discuss a class of recurrences that can be characterized by WFSAs. The mathematical counterpart of WFSAs are *rational power series* (Berstel and Reutenauer, 1988), justifying naming such recurrences *rational*:

Definition 8 (rational recurrence)**.** The recurrence of $\mathbf{c}\colon \Sigma^* \to \mathbb{K}^d$ is said to be *rational*, if there exists a set of weighted finite state automata $\{\mathcal{A}_i\}_{i=1}^d$ over alphabet Σ and semiring $\langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$ with both $\oplus$ and $\otimes$ taking constant time and space, such that $\forall \mathbf{x} \in \Sigma^*$,

$$\big[\mathbf{c}(\mathbf{x})\big]_i = \mathcal{A}_i[\![\mathbf{x}]\!],\ \forall i \in \{1, 2, \ldots, d\}.^4 \qquad (9)$$

It directly follows from Proposition 7 that

Corollary 9. *The recurrence in Example 6 is rational.*

4 Relationship to Existing Neural Models

This section studies several recently proposed neural architectures, and relates them to rational recurrences. §4.1 begins by relating some of them to the RNN defined in Example 6, and then to the WFSA $\mathcal{B}$ (Example 5). We then describe a WFSA similar to $\mathcal{B}$, but with one additional state, and discuss how it provides a new view of RNN models motivated by n-gram features (§4.2). In §4.3 we study rational recurrences that are not elementwise, using an existing model.

In the following discussion, we shall assume the real seimiring, unless otherwise noted.

4.1 Neural Architectures Related to $\mathcal{B}$

Despite its simplicity, Example 6 corresponds to several existing neural architectures. For instance, quasi-RNN (QRNN; Bradbury et al., 2017) and simple recurrent unit (SRU; Lei et al., 2017b) aim to speed up the recurrent computation. To do so, they drop the matrix multiplication dependence on the previous hidden state, resulting in similar recurrences to that in Example 6.[5] Other works start from different motivations, but land on similar recurrences, e.g., strongly-typed RNNs (T-RNN; Balduzzi and Ghifary, 2016) and its gated variants (T-LSTM and T-GRU), and structurally constrained RNNs (SCRN; Mikolov et al., 2014).

The analysis in §3.1 directly applies to SRU, T-RNN, and SCRN. In fact, Example 6 presents a slightly more complicated version of them. In these models, input representations are computed without the bias term or any nonlinearity: $\mathbf{u}_t = \mathbf{W}_u \mathbf{v}_t$. By Proposition 7 and Corollary 9:

Corollary 10. *The recurrences of single-layer SRU, T-RNN, and SCRN architectures are rational.*

It is slightly more complicated to analyze the recurrences of the QRNN, T-LSTM, and T-GRU. Although their hidden states $\mathbf{c}_t$ are updated in the same way as Equation 5c, the input representations and gates may depend on *previous inputs*. For example, in T-LSTM and T-GRU, the forget gate is a function of two consecutive inputs:

$$\mathbf{f}_t = \boldsymbol{\sigma}\left(\mathbf{V}_f \mathbf{v}_{t-1} + \mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\right). \qquad (10)$$

QRNNs are similar, but may depend on up to K tokens, due to the K-window convolutions. Eisner (2002) discuss finite state machines for second (or higher) order probabilistic sequence models. Following the same intuition, we sketch the construction of WFSAs corresponding to QRNNs with 2-window convolutions in Appendix A, and summarize the key results here:

Proposition 11. *The recurrences of single-layer T-GRU, T-LSTM, and QRNN are rational. In particular, a single-layer d-dimensional QRNN using K-window convolutions can be recovered by a set of d WFSAs, each with $O(2\,|\Sigma|^{K-1})$ states.*

The size of WFSAs needed to recover QRNN grows exponentially in the window size. Therefore, at least for QRNNs, Proposition 11 has more conceptual value than practical.

4.2 More than Two States

So far our discussion has centered on $\mathcal{B}$, a two-state WFSA capturing unigram patterns (Example 5). In the same spirit as going from unigram to n-gram features, one can use WFSAs with more states to capture longer patterns (Schwartz et al., 2018). In this section we augment $\mathcal{B}$ by introducing more states, and explore its relationship to some neural architectures motivated by n-gram features. We start with a three-state WFSA as an example, and then discuss more general cases.

Figure 2 diagrams a WFSA $\mathcal{C}$, augmenting $\mathcal{B}$ with another state. To reach the final state q_2, at

[4] We restrict that both operations take constant time and space, to exclude the use of arbitrarily complex semirings (§4.3).

[5] The SRU architecture discussed through this work is based on Lei et al. (2017b). In a later updated version, Lei et al. (2018) introduce diagonal matrix multiplication interaction in the hidden state updates, inspired by (Li et al., 2018), which yields a recurrence not obviously rational.

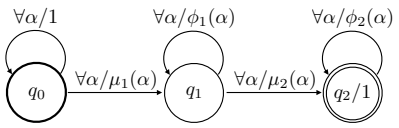

Figure 2: A three-state WFSA $\mathcal{C}$ discussed in §4.2.

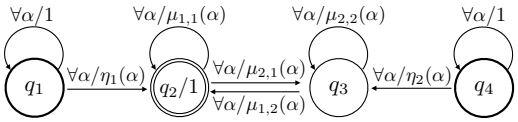

Figure 3: WFSA $\mathcal{D}_1$ discussed in §4.3. Two initial states q_1 and q_4 are used here.

least two transitions must be taken, in contrast to one in $\mathcal{B}$. History information is decayed by the self-loop at the final state q_2, assuming ϕ_2 is between 0 and 1. $\mathcal{C}$ has another self-loop over q_1, weighted by $\phi_1 \in (0, 1)$. The motivation is to allow (but down-weight) nonconsecutive bigrams, as we will soon show.

The scores assigned by $\mathcal{C}$ can be inductively computed by applying the Forward algorithm (§2). Given input sequence $\mathbf{x}$ longer than one, let $\mathcal{C}[\![\mathbf{x}_{:0}]\!] = 0$, then $\mathcal{C}[\![\mathbf{x}_{:t+1}]\!] =$

$$\mathcal{C}[\![\mathbf{x}_{:t}]\!]\,\phi_2(x_{t+1}) + \beta_t\,\mu_2(x_{t+1}), \tag{11}$$

where

$$\beta_t = \beta_{t-1}\,\phi_1(x_t) + \mu_1(x_t), \tag{12}$$

and $\beta_0 = 0$. Unrolling β_t in time, we get $\beta_t =$

$$\sum_{j=1}^{t-1}\left(\mu_1(x_j)\prod_{k=j+1}^{t}\phi_1(x_k)\right) + \mu_1(x_t). \tag{13}$$

Due to the self-loop over state q_1, β_t can be seen as a weighted sum of the μ_1 terms up to x_t (Equation 13). The second product term in Equation 11 then provides multiplicative interactions between μ_2, and the weighted sum of μ_1s. In this sense, it captures nonconsecutive bigram features.

At a first glance, Equations 11 and 12 resemble recurrent convolutional neural networks (RCNN; Lei et al., 2016). RCNN is inspired by nonconsecutive n-gram features and low rank tensor factorization. It is later studied from a string kernel perspective (Lei et al., 2017a). Here we review its nonlinear bigram version:

$$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \boldsymbol{\lambda}_t + \mathbf{u}_t^{(1)}, \tag{14a}$$

$$\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \boldsymbol{\lambda}_t + \mathbf{c}_{t-1}^{(1)} \odot \mathbf{u}_t^{(2)}, \tag{14b}$$

where the $\mathbf{u}_t^{(j)}$s are computed similarly to Equation 5b, and $\mathbf{c}_t^{(2)}$ is used as output for onward computation. Different strategies to computing $\boldsymbol{\lambda}_t$ were explored (Lei et al., 2015, 2016). When $\boldsymbol{\lambda}_t$ is a constant, or depends only on x_t, e.g., $\boldsymbol{\lambda}_t = \boldsymbol{\sigma}(\mathbf{W}_\lambda \mathbf{v}_t + \mathbf{b}_\lambda)$, the ith dimension of Equations 14 can be recovered from Equation 11, by letting

$$\mu_j(x_t) = [\mathbf{u}_t^{(j)}]_i,\ \phi_j(x_t) = [\boldsymbol{\lambda}_t]_i,\ j = 1, 2. \tag{15}$$

It is straightforward to generalize the above discussion to higher order cases: n-gram RCNN corresponds to WFSAs with $n + 1$ states, constructed similarly to how we build $\mathcal{C}$ from $\mathcal{B}$ (Appendix B).

Proposition 12. *For a single-layer RCNN with $\boldsymbol{\lambda}_t$ being a constant or depending only on x_t, the recurrence is rational.*

As noted later in §4.3, its recurrence may not be rational when $\boldsymbol{\lambda}_t = \boldsymbol{\sigma}(\mathbf{W}_c \mathbf{c}_{t-1} + \mathbf{W}_\lambda \mathbf{v}_t + \mathbf{b}_\lambda)$.

4.3 Beyond Elementwise Operations

So far we have discussed rational recurrences for models using elementwise recurrent updates (e.g., Equation 5c). This section uses an existing model as an example, to study a rational recurrence that is not elementwise. We focus on the input switched affine network (ISAN; Foerster et al., 2017). Aiming for efficiency and interpretability, it does not use any explicit nonlinearity; its affine transformation parameters depend only on the input:

$$\mathbf{c}_t = \mathbf{W}_{x_t} \mathbf{c}_{t-1} + \mathbf{b}_{x_t}. \tag{16}$$

Due to the matrix multiplication, the recurrence of a single-layer ISAN is not elementwise. Yet, we argue that it is rational. We will sketch the proof for a 2-dimensional case, and it is straightforward to generalize to higher dimensions (Appendix C).

We define two WFSAs, each recovering one dimension of ISAN's recurrent updates. Figure 3 diagrams one of them, $\mathcal{D}_1$. The other one, $\mathcal{D}_2$, is identical (including shared weights), except using q_3 instead of q_2 as the final state. For any nonempty input sequence $\mathbf{x} \in \Sigma^+$, the scores assigned by $\mathcal{D}_1$ and $\mathcal{D}_2$ can be inductively computed by applying the Forward algorithm. Letting $\mathcal{D}_1[\![\mathbf{x}_{:0}]\!] = \mathcal{D}_2[\![\mathbf{x}_{:0}]\!] = 0$, for $t \geq 1$

$$\begin{bmatrix}\mathcal{D}_1[\![\mathbf{x}_{:t}]\!]\\ \mathcal{D}_2[\![\mathbf{x}_{:t}]\!]\end{bmatrix} = \widetilde{\mathbf{W}}_{x_t}\begin{bmatrix}\mathcal{D}_1[\![\mathbf{x}_{:t-1}]\!]\\ \mathcal{D}_2[\![\mathbf{x}_{:t-1}]\!]\end{bmatrix} + \widetilde{\mathbf{b}}_{x_t}, \tag{17}$$

where

$$\widetilde{\mathbf{W}}_{x_t} = \begin{bmatrix} \mu_{1,1}(x_t) & \mu_{1,2}(x_t) \\ \mu_{2,1}(x_t) & \mu_{2,2}(x_t) \end{bmatrix}, \quad \widetilde{\mathbf{b}}_{x_t} = \begin{bmatrix} \eta_1(x_t) \\ \eta_2(x_t) \end{bmatrix}. \tag{18}$$

Then Equation 16, in the case of hidden size 2, is recovered by letting $\mathbf{W}_{x_t} = \widetilde{\mathbf{W}}_{x_t}$ and $\mathbf{b}_{x_t} = \widetilde{\mathbf{b}}_{x_t}$.

Proposition 13. *The recurrence of a single-layer ISAN is rational.*

Corollary 14. *For a single-layer Elman network, in the absence of any nonlinearity, the recurrence is rational.*

Discussion. It is known that an Elman network can approximate any recursively computable partial function (Siegelmann and Sontag, 1995). On the other hand, in their single-layer cases, WFSAs (and thus models with rational recurrences) are restricted to rational series (Schützenberger, 1961). Therefore, we hypothesize that models like Elman networks, LSTMs, and GRUs, where the recurrences depend on previous states through affine transformations followed by nonlinearities, are not rational.

This work does not intend to propose rational recurrences as a concept general enough to include most existing RNNs. Rather, we wish to study a more constrained class of methods to better understand the connections between WFSAs and RNNs. Therefore in Definition 8, we restrict the semirings to be "simple," in the sense that both operations take constant time and space. Such a restriction aims to exclude the possibility of hiding arbitrarily complex computations inside the semiring, which might allow RNNs to satisfy the definition in a trivial and unilluminating way.

Such theoretical limitations might be less severe than they appear, since it is not yet entirely clear what they correspond to in practice, especially when multiple vertical layers of these models are used (Leshno and Schocken, 1993). We defer to future work the further study of the connections between WFSAs and Elman-style RNNs.

Closing this section, Table 1 summarizes the discussed recurrent neural architectures and their corresponding WFSAs.

	Models	Recurrence Function	WFSA
§4.1	SRU, SCRN T-RNN, QRNN	$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t$	$\mathcal{B}$
§4.2	RCNN	$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \boldsymbol{\lambda}_t + \mathbf{u}_t^{(1)}$ $\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \boldsymbol{\lambda}_t + \mathbf{c}_{t-1}^{(1)} \odot \mathbf{u}_t^{(2)}$	$\mathcal{C}$
§4.3	ISAN	$\mathbf{c}_t = \mathbf{W}_{x_t}\mathbf{c}_{t-1} + \mathbf{b}_{x_t}$	$\mathcal{D}_1, \mathcal{D}_2$

Table 1: Recurrent neural network architectures discussed in §4 and their corresponding WFSAs. §4.1: SRU (Lei et al., 2017b), SCRN (Mikolov et al., 2014), T-RNN and its gated variants (Balduzzi and Ghifary, 2016), and QRNN (Bradbury et al., 2017); §4.2: RCNN (Lei et al., 2016); §4.3: ISAN (Foerster et al., 2017).

5 Deriving Neural Models from WFSAs

Rational recurrences provide a new view of several recently proposed neural models. Based on such observations, this section aims to explore potential approaches to designing neural architectures in a more interpretable and intuitive way: by deriving them from WFSAs. §5.1 studies an interpolation of unigram and bigram features by combining 2-state and 3-state WFSAs (Figures 1 and 2). We then explore alternative semirings (§5.2), an approach orthogonal to what we've discussed so far.

We note that our goal is not to devise new state-of-the-art architectures. Rather, we illustrate a new design process for neural architectures that draws inspiration from WFSAs. That said, in our experiments (§6), one of our new architectures performs as well as or better than strong baselines.

5.1 Aggregating Different Length Patterns

We start by presenting a straightforward extension to 2-state and 3-state rational models: one combining both. It is inspired by many classical NLP models, where unigram features and higher-order ones are interpolated.

Figure 4 diagrams a 4-state WFSA $\mathcal{F}$. Compared to $\mathcal{C}$ (Figure 2), $\mathcal{F}$ uses q_1 as a second final state, aiming to capture both unigram and bigram patterns, since a path is allowed to stop at q_1 after consuming one input. The final states are weighted by ρ_1 and ρ_2 respectively. Another notable modification is the additional state q_3, which is used to create a "shortcut" to reach q_2, together with an ε-transition. Specifically, starting from q_0, a path can now take the ε-transition and reach q_3, and then take a transition with weight μ_2 to reach q_2. Recall from §2, that ε-transitions do not consume any input, yet they can still be weighted by a (parameterized) function γ *not* depending on the inputs. The ε-transition allows for skipping the

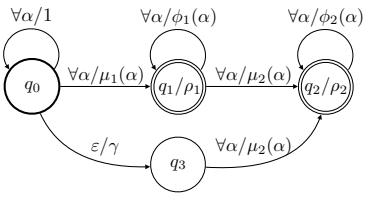

Figure 4: A WFSA $\mathcal{F}$ that combines both unigram and bigram features (§5.1). Two final states q_1 and q_2 are used, with weights ρ_1 and ρ_2, respectively.

Model	Unigram	Bigram	Semiring
RRNN($\mathcal{B}$)	✓		real
RRNN($\mathcal{B}$)$_{\text{m+}}$	✓		max-plus
RRNN($\mathcal{C}$)		✓	real
RRNN($\mathcal{F}$)	✓	✓	real

Table 2: Rational recurrent neural architectures compared in the experiments (§6.1).

first word in a bigram. It can be discouraged by using $\gamma \in (0, 1)$, just as we do in our experiments.

Deriving the neural architecture. As in §3, we relate hidden states of an RNN to the scores assigned by WFSAs to input strings. We then derive the neural architecture with a dynamic program. Here we keep the discussion self-contained by explicitly overviewing the procedure. It is a direct application of the Forward algorithm (§2), though now in a form that deals with the ε-transition. Such an approach applies, of course, to more general cases, as noted by Schwartz et al. (2018).

Given an input string $\mathbf{x} \in \Sigma^+$, let $z_t^{(j)}$ denote the total score of all paths landing in state q_j *just* after consuming x_t. Let $z_0^{(j)} = 0$, then for $t \geq 1$,

$$
\begin{aligned}
z_t^{(0)} &= 1 \\
z_t^{(1)} &= z_{t-1}^{(1)}\, \phi_1(x_t) + z_{t-1}^{(0)}\, \mu_1(x_t) \\
z_t^{(3)} &= z_t^{(0)}\, \gamma \\
z_t^{(2)} &= z_{t-1}^{(2)}\, \phi_2(x_t) + (z_{t-1}^{(1)} + z_{t-1}^{(3)})\, \mu_2(x_t) \\
\mathcal{F}[\![\mathbf{x}_{:t}]\!] &= \rho_1\, z_t^{(1)} + \rho_2\, z_t^{(2)}.
\end{aligned}
$$

We now collect d of these WFSAs to construct an RNN, and we parameterize their weight functions with the technique we've been using:

$$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \mathbf{f}_t^{(1)} + \mathbf{u}_t^{(1)}, \tag{19a}$$

$$\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \mathbf{f}_t^{(2)} + (\mathbf{c}_{t-1}^{(1)} + \mathbf{r}) \odot \mathbf{u}_t^{(2)}, \tag{19b}$$

$$\mathbf{c}_t = \mathbf{p}^{(1)} \odot \mathbf{c}_t^{(1)} + \mathbf{p}^{(2)} \odot \mathbf{c}_t^{(2)}, \tag{19c}$$

where

$$\mathbf{f}_t^{(j)} = \boldsymbol{\sigma}\big(\mathbf{W}_f^{(j)} \mathbf{v}_t + \mathbf{b}_f^{(j)}\big), \tag{20a}$$

$$\mathbf{u}_t^{(j)} = (\mathbf{1} - \mathbf{f}_t^{(j)}) \odot \boldsymbol{g}\big(\mathbf{W}_u^{(j)} \mathbf{v}_t + \mathbf{b}_u^{(j)}\big), \tag{20b}$$

$$\mathbf{p}^{(j)} = \boldsymbol{\sigma}(\mathbf{b}_p^{(j)}), \quad \mathbf{r} = \boldsymbol{\sigma}(\mathbf{b}_r). \tag{20c}$$

The $\mathbf{p}$ vectors correspond to the final state weights ρ_1 and ρ_2. Despite the similarities, $\mathbf{p}$ are different from output gates (Bradbury et al., 2017), since the former do not depend on the input, and are parameterized (through a sigmoid) by two leanred vectors $\mathbf{b}_p^{(j)}$. The same applies to $\mathbf{r}$ and $\mathbf{b}_r$, which correspond to the weights for ε-transitions γ.

5.2 Alternative Semirings

Our new understanding of rational recurrences allows us to consider a different kind of extension: replacing the semiring. We introduce an example, which modifies Example 6 by replacing its real (plus-times) semiring with the max-plus semiring $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$:

Example 15.

$$\mathbf{f}_t = \log \boldsymbol{\sigma}\big(\mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\big), \tag{21a}$$

$$\mathbf{u}_t = \boldsymbol{g}\big(\mathbf{W}_u \mathbf{v}_t + \mathbf{b}_u\big), \tag{21b}$$

$$\mathbf{c}_t = \max\{\mathbf{f}_t + \mathbf{c}_{t-1}, \mathbf{u}_t\}. \tag{21c}$$

Example 15 does not use the forget gate when computing $\mathbf{u}_t$ (Equation 21b), which is different from its plus-times counterpart, where $\mathbf{u}_t = (\mathbf{1} - \mathbf{f}_t) \odot \boldsymbol{g}\big(\mathbf{W}_u \mathbf{v}_t + \mathbf{b}_u\big)$. The reason is that, unlike the real semiring, the max-plus semiring lacks a well-defined negation. Possible alternatives include taking the log of a separate input gate, or using $\log(\mathbf{1} - \mathbf{f}_t)$, which we leave for future work.

Example 15 can be seen as replacing sum-pooling with max-pooling. Both max and sum-pooling have been used successfully in vision and NLP models. Intuitively, max-pooling "detects" the occurrence of a pattern while sum-pooling "counts" the occurrence of a pattern. One advantage of max operator is that the model's decisions can be back-traced and interpreted, as argued by Schwartz et al. (2018). Such a technique is applicable to all the models with rational recurrences.

6 Experiments

This section evaluates four rational RNNs on language modeling (§6.2) and text categorization (§6.3). Our goal is to compare the behaviors of models derived from different WFSAs, showing that our understanding of WFSAs allows us to

improve existing rational models.

6.1 Compared Models

Our comparisons focus on the *recurrences* of the models, i.e., how the hidden states $\mathbf{c}_t$ are computed (e.g., Equations 5c and 19c). Therefore we follow Lei et al. (2017b) and use $\mathbf{u}_t^{(j)} = \mathbf{W}_u^{(j)}\mathbf{v}_t^{(j)}$ across all compared models, listed below and as well as in Table 2:

- RRNN($\mathcal{B}$), with real semiring (§4.1);
- RRNN($\mathcal{B}$)$_{\text{m+}}$, with max-plus semiring (§5.2);
- RRNN($\mathcal{C}$), with real semiring (§4.2);
- RRNN($\mathcal{F}$), with real semiring (§5.1).

We also compare to an LSTM baseline. Aiming to control for comfounding factors, we do not use highway connections in any of the models.[6] In the interest of space, the full architectures and hyperparameters are detailed in Appendices D and E.

6.2 Language Modeling

Dataset and implementation. We experiment with the Penn Treebank corpus (PTB; Marcus et al., 1993). We use the preprocessing and splits from Mikolov et al. (2010), resulting in a vocabulary size of 10K and 1M tokens.

Following standard practice, we treat the training data as one long sequence, split into mini batches, and train using BPTT truncated to 35 time steps (Williams and Peng, 1990). The input embeddings and output softmax weights are tied (Press and Wolf, 2017).

Results. Following Collins et al. (2017) and Melis et al. (2018), we compare models controlling for parameter budget. Table 3 summarizes language modeling perplexities on PTB test set. The middle block compares all models with two layers and 10M trainable parameters. RRNN($\mathcal{B}$) and RRNN($\mathcal{C}$) achieve roughly the same performance; interpolating both unigram and bigram features, RRNN($\mathcal{F}$) outperforms others by more than 2.9 test perplexity. For the three-layer and 24M setting (the bottom block), we observe similar trends, except that RRNN($\mathcal{C}$) slightly underperforms RRNN($\mathcal{B}$). Here RRNN($\mathcal{F}$) outperforms others by more than 2.1 perplexity.

Using a max-plus semiring, RRNN($\mathcal{B}$)$_{\text{m+}}$ *underperforms* RRNN($\mathcal{B}$) under both settings. Possible reasons could be the suboptimal design choice for computing input representations in the former (§5.2). Finally, most compared models outperform the LSTM baselines, whose numbers are taken from Lei et al. (2017b).[7]

Model	ℓ	# Params.	Dev.	Test
LSTM	2	24M	73.3	71.4
LSTM	3	24M	78.8	76.2
RRNN($\mathcal{B}$)	2	10M	73.1	69.2
RRNN($\mathcal{B}$)$_{\text{m+}}$	2	10M	75.1	71.7
RRNN($\mathcal{C}$)	2	10M	72.5	69.5
RRNN($\mathcal{F}$)	2	10M	**69.5**	**66.3**
RRNN($\mathcal{B}$)	3	24M	68.7	65.2
RRNN($\mathcal{B}$)$_{\text{m+}}$	3	24M	70.8	66.9
RRNN($\mathcal{C}$)	3	24M	70.0	67.0
RRNN($\mathcal{F}$)	3	24M	**66.0**	**63.1**

Table 3: Language modeling perplexity on PTB test set (lower is better). LSTM numbers are taken from Lei et al. (2017b). ℓ denotes the number of layers. Bold font indicates best performance.

Split	Amazon	SST	subj	CR
Train	20K	6.9K	8K	3.0K
Dev.	5K	0.9K	1K	0.4K
Test	25K	1.8K	1K	0.4K

Table 4: Number of instances in the text classification datasets (§6.3).

6.3 Text Classification

Implementation. We use unidirectional 2-layer architectures for all compared models. To build the classifiers, we feed the final RNN hidden states into a 2-layer tanh-MLP. Further implementation details are described in Appendix E.

Datasets. We experiment with four binary text classification datasets, described below.

- **Amazon** (electronic product review corpus; McAuley and Leskovec, 2013).[8] We focus on the positive and negative reviews.
- **SST** (Stanford sentiment treebank; Socher et al., 2013).[9] We focus on the binary classification task. SST provides labels for syntactic phrases; we experiment with a more realistic setup, and

[6] Thus RRNN($\mathcal{B}$) is essentially an SRU without highway connections. We denote it differently, to note its differences from the original implementation (Lei et al., 2017b). Similarly, we do not denote RRNN($\mathcal{C}$) as RCNN (Lei et al., 2016).

[7] Melis et al. (2018) point out that carefully tuning LSTMs can achieve much stronger performance, at the cost of exceptionally large amounts of computational resources for tuning.

[8] http://riejohnson.com/cnn_data.html

[9] nlp.stanford.edu/sentiment/index.html

Model	Amazon	SST	subj	CR
LSTM	$91.2_{\pm 0.3}$	$85.1_{\pm 0.6}$	$93.3_{\pm 0.6}$	$82.4_{\pm 1.5}$
RRNN($\mathcal{B}$)	$92.4_{\pm 0.1}$	$85.8_{\pm 0.3}$	$93.9_{\pm 0.4}$	$84.1_{\pm 1.0}$
RRNN($\mathcal{C}$)	$\mathbf{92.8}_{\pm 0.2}$	$84.8_{\pm 0.4}$	$93.8_{\pm 0.6}$	$84.5_{\pm 0.9}$
RRNN($\mathcal{B}$)$_{\text{m+}}$	$89.2_{\pm 3.1}$	$84.9_{\pm 0.4}$	$92.6_{\pm 0.5}$	$84.3_{\pm 0.5}$
RRNN($\mathcal{F}$)	$92.7_{\pm 0.2}$	$\mathbf{86.5}_{\pm 0.6}$	$\mathbf{94.8}_{\pm 0.5}$	$\mathbf{85.1}_{\pm 0.5}$

Table 5: Text classification test accuracy averaged over 5 runs. $\pm$ denotes standard deviation, and bold font indicates best averaged performance.

consider only complete sentences at either training or evaluating time.

- **subj** (Subjectivity dataset; Pang and Lee, 2004). As **subj** doesn't come with official splits, we randomly split it to train (80%), development (10%), and test (10%) sets.
- **CR** (customer reviews dataset; Hu and Liu, 2004).[10] As with **subj**, we randomly split this dataset using the same ratio.

Table 4 summarizes the sizes of the datasets.

Results. Table 5 summarizes text classification test accuracy. We report the average performance of 5 trials different only in random seeds. RRNN($\mathcal{F}$) outperforms all other models on 3 out of the 4 datasets. For Amazon, the largest one, we do not observe significant differences between RRNN($\mathcal{F}$) and RRNN($\mathcal{C}$), while both outperform others. This may suggest that the interpolation of unigram and bigram features by RRNN($\mathcal{F}$) is especially useful in small data setups. As in the language modeling experiments, RRNN($\mathcal{B}$)$_{\text{m+}}$ underperforms all other models in most cases, and in particular RRNN($\mathcal{B}$). These results provide evidence that replacing the real semiring in rational models might be challenging. We leave further exploration to future work.

7 Related Work

Weighted finite state automata. WFSAs were once popular among many sequential tasks (Mohri et al., 2002; Kumar and Byrne, 2003; Cortes et al., 2004; Pardo and Birmingham, 2005; Moore et al., 2006, *inter alia*), and are still successful in morphology (Dreyer, 2011; Cotterell et al., 2015; Rastogi et al., 2016, *inter alia*). Compared to neural networks, WFSAs are better understood theoretically and arguably more interpretable. They were recently revisited in combination with the former in, e.g., text generation (Ghazvininejad et al., 2016, 2017; Lin et al., 2017) and automatic music accompaniment (Forsyth, 2016).

Recurrent neural networks. RNNs (Elman, 1990; Jordan, 1989) prove to be strong models for sequential data (Siegelmann and Sontag, 1995). Besides the perhaps most notable gated variants (Hochreiter and Schmidhuber, 1997; Cho et al., 2014), extensive efforts have been devoted to developing alternatives (Balduzzi and Ghifary, 2016; Miao et al., 2016; Zoph and Le, 2017; Lee et al., 2017; Lei et al., 2017a; Vaswani et al., 2017; Gehring et al., 2017, *inter alia*). Departing from the above approaches, this work derives RNN architectures drawing inspiration from WFSAs.

Another line of work studied the connections between WFSAs and RNNs in terms of modeling capacity, both empirically (Kolen, 1993; Giles et al., 1992; Weiss et al., 2018, *inter alia*) and theoretically (Cleeremans et al., 1989; Visser et al., 2001; Chen et al., 2018, *inter alia*).

8 Conclusion

We presented *rational recurrences*, a new construction to study the recurrent updates in RNNs, drawing inspiration from WFSAs. We showed that rational recurrences are in frequent use by several recently proposed recurrent neural architectures, providing new understanding of them. Based on such connections, we discussed approaches to deriving novel neural architectures from WFSAs. Our empirical results demonstrate the potential of doing so. We publicly release our implementation at `https://github.com/Noahs-ARK/rational-recurrences`.

Acknowledgments

We thank Jason Eisner, Luheng He, Tao Lei, Omer Levy, members of the ARK lab at the University of Washington, and researchers at the Allen Institute for Artificial Intelligence for their helpful comments on an early version of this work, and the anonymous reviewers for their valuable feedback. We also thank members of the Aristo team at the Allen Institute for Artificial Intelligence for their support with the Beaker experimentation system. This work was supported in part by NSF grant IIS-1562364 and by the NVIDIA Corporation through the donation of a Tesla GPU.

[10] `http://www.cs.uic.edu/?liub/FBS/sentiment-analysis.html`

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

David Balduzzi and Muhammad Ghifary. 2016. Strongly-typed recurrent neural networks. In *Proc. of ICML*.

Leonard E. Baum and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563.

Jean Berstel, Jr. and Christophe Reutenauer. 1988. *Rational Series and Their Languages*. Springer-Verlag, Berlin, Heidelberg.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-recurrent neural network. In *Proc. of ICLR*.

Yining Chen, Sorcha Gilroy, Kevin Knight, and Jonathan May. 2018. Recurrent neural networks as weighted language recognizers. In *Proc. of NAACL*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of EMNLP*.

Axel Cleeremans, David Servan-Schreiber, and James L. McClelland. 1989. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381.

Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. 2017. Capacity and trainability in recurrent neural networks. In *Proc. of ICLR*.

Corinna Cortes, Patrick Haffner, and Mehryar Mohri. 2004. Rational kernels: Theory and algorithms. *Journal of Machine Learning Research*, 5:1035–1062.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2015. Modeling word forms using latent underlying morphs and phonology. *TACL*, 3:433–447.

Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Enhanced sentiment learning using twitter hashtags and smileys. In *Proc. of COLING*.

Markus Dreyer. 2011. *A Non-parametric Model for the Discovery of Inflectional Paradigms from Plain Text Using Graphical Models over Strings*. Ph.D. thesis, Johns Hopkins University.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proc. of NAACL*.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proc. of ACL*.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Jakob N. Foerster, Justin Gilmer, Jan Chorowski, Jascha Sohl-Dickstein, and David Sussillo. 2017. Intelligible language modeling with input switched affine networks. In *Proc. of ICML*.

Jonathan P. Forsyth. 2016. *Automatic musical accompaniment using finite state machines*. Ph.D. thesis, New York University.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *Proc. of ICML*.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating topical poetry. In *Proc. of EMNLP*.

Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. 2017. Hafez: an interactive poetry generation system. In *Proc. of ACL, System Demonstrations*.

C. Lee Giles, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.

Yoav Goldberg. 2016. A primer on neural network models for natural language processing. *JAIR*, 57:345–420.

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

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proc. of KDD*.

Michael I. Jordan. 1989. Serial order: A parallel, distributed processing approach. In *Advances in Connectionist Theory: Speech*. Erlbaum.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.

John F. Kolen. 1993. Fool's gold: Extracting finite state machines from recurrent network dynamics. In *Proc. of NIPS*.

Werner Kuich and Arto Salomaa, editors. 1986. *Semirings, Automata, Languages*. Springer-Verlag.

Shankar Kumar and William Byrne. 2003. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proc. of NAACL*.

Yann LeCun. 1998. Gradient-based Learning Applied to Document Recognition. In *Proc. of the IEEE*.

Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2017. Recurrent additive networks. *arXiv:1705.07393*.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2015. Molding CNNs for text: non-linear, non-consecutive convolutions. In *Proc. of EMNLP*.

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2017a. Deriving neural architectures from sequence and graph kernels. In *Proc. of ICML*.

Tao Lei, Hrishikesh Joshi, Regina Barzilay, Tommi Jaakkola, Kateryna Tymoshenko, Alessandro Moschitti, and Lluís Màrquez. 2016. Semi-supervised question retrieval with gated convolutions. In *Proc. of NAACL*.

Tao Lei, Yu Zhang, and Yoav Artzi. 2017b. Training RNNs as fast as CNNs. arXiv:1709.02755.

Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. 2018. Simple recurrent units for highly parallelizable recurrence. In *Proc. of EMNLP*.

Moshe Leshno and Shimon Schocken. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. 2018. Independently recurrent neural network (IndRNN): Building A longer and deeper RNN. In *Proc. of CVPR*.

Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, and Ming-Ting Sun. 2017. Adversarial ranking for language generation. In *Proc. of NIPS*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proc. of RecSys*.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *Proc. of ICLR*.

Yajie Miao, Jinyu Li, Yongqiang Wang, Shi-Xiong Zhang, and Yifan Gong. 2016. Simplifying long short-term memory acoustic models for fast training and decoding. In *Proc. of ICASSP*.

Tomas Mikolov, Armand Joulin, Sumit Chopra, Michaël Mathieu, and Marc'Aurelio Ranzato. 2014. Learning longer memory in recurrent neural networks. *arXiv:1412.7753*.

Tomas Mikolov, Martin Karafit, Luks Burget, Jan Cernock, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. of INTERSPEECH*.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Darren Moore, John Dines, Mathew Magimai-Doss, Jithendra Vepa, Octavian Cheng, and Thomas Hain. 2006. Juicer: A weighted finite-state transducer speech decoder. In *Proc. of MLMI*.

Bo Pang and Lillian Lee. 2004. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proc. of ACL*.

Bryan Pardo and William Birmingham. 2005. Modeling form for on-line following of musical performances. In *Proc. of AAAI*.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proc. of ACL*.

Hao Peng, Sam Thomson, and Noah A. Smith. 2018. Backpropagating through structured argmax using a spigot. In *Proc. of ACL*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proc. of EMNLP*.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proc. of EACL*.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proc. of NAACL*.

Jacques Sakarovitch. 2009. Rational and recognisable power series. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, pages 105–174. Springer Berlin Heidelberg, Berlin, Heidelberg.

Marcel Paul Schützenberger. 1961. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270.

Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. SoPa: Bridging CNNs, RNNs, and weighted finite-state machines. In *Proc. of ACL*.

Hava T. Siegelmann and Eduardo D. Sontag. 1995. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of NIPS*.

Ingmar Visser, Maartje EJ Raijmakers, and Peter CM Molenaar. 2001. Hidden markov model interpretations of neural networks. In *Connectionist Models of Learning, Development and Evolution*, pages 197–206. Springer.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proc. of ACL*.

Ronald J. Williams and Jing Peng. 1990. An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural computation*, 2(4):490–501.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv:1409.2329*.

Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *Proc. of ICLR*.

Appendices

A Proof of Proposition 11

Proof. Let's consider a single-layer QRNN with 2-window convolutions:

$$\begin{aligned}\mathbf{f}_t &= \boldsymbol{\sigma}\left(\mathbf{V}_f\mathbf{v}_{t-1} + \mathbf{W}_f\mathbf{v}_t + \mathbf{b}_f\right),\\ \mathbf{u}_t &= (\mathbf{1} - \mathbf{f}_t) \odot \boldsymbol{g}\big(\mathbf{V}_u\mathbf{v}_{t-1} + \mathbf{W}_u\mathbf{v}_t + \mathbf{b}_u\big),\\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t.\end{aligned}$$

A similar analysis applies to T-GRUs and T-LSTMs directly, and it should be straightforward to generalize the discussion to QRNNs with larger convolution windows.

Let Σ denote the alphabet, and let $\mathbf{x} = x_1x_2\dots x_n \in \Sigma^+$ be a nonempty input string. consider a WFSA over the real semiring with $2|\Sigma| + 1$ states, where q_0 is the initial state with $\lambda(q_0) = 1$; $|\Sigma|$ of them are final states $\mathcal{Q}_2 = \{q_\alpha\}_{\alpha\in\Sigma}$, with $\rho(q_\alpha) = 1$, and the remaining $|\Sigma|$ states are denoted by $\mathcal{Q}_1 = \{p_\alpha\}_{\alpha\in\Sigma}$.

The transition weights τ are constructed by

$$\begin{aligned}&\tau(q_0, p_\alpha, \alpha) = 1, && \forall p_\alpha, \in \mathcal{Q}_1;\\ &\tau(p_\alpha, p_\beta, \beta) = 1, && \forall p_\alpha, p_\beta \in \mathcal{Q}_1;\\ &\tau(p_\alpha, q_\beta, \beta) = \mu_\alpha(\beta), && \forall p_\alpha \in \mathcal{Q}_1, \forall q_\beta \in \mathcal{Q}_2;\\ &\tau(q_\alpha, q_\beta, \beta) = \phi_\alpha(\beta), && \forall q_\alpha, q_\beta \in \mathcal{Q}_2.\end{aligned}$$

$\tau = 0$ otherwise. Then one dimension of the reccurent updates of a 2-window QRNN is recovered by parameterizing the weight functions as

$$\mu_{x_{t-1}}(x_t) = [\mathbf{u}_t]_i, \quad \phi_{x_{t-1}}(x_t) = [\mathbf{f}_t]_i. \tag{22}$$

The recurrent computation of a 2-window QRNN of hidden size d can then be recovered by collecting d such WFSAs. □

B Proof of Proposition 12

Proof. We present the construction of WFSAs for a single layer n-gram RCNNs of hidden size d.

Let's assume a given input sequence $\mathbf{x} \in \Sigma^+$, with $|\mathbf{x}| > n$, since otherwise one only needs include paddings, just as in a RCNN. Consider a WFSA with $n+1$ states $\mathcal{Q} = \{q_i\}_{i=0}^n$ over the real semiring. Use q_0 as the initial state with $\lambda(q_0) = 1$, and q_n as the final state with $\rho(q_n) = 1$. The transition weight function is defined by

$$\tau(q_i, q_j, \alpha) = \begin{cases}1, & i = j = 0,\\ \phi(\alpha), & j = i > 0,\\ \mu_j(\alpha), & j = i + 1,\\ 0, & \text{otherwise.}\end{cases}$$

Let $z_t^{(j)}$ denote the total score of all paths landing in state q_j *just* after consuming x_t. Let $z_t^{(j)} = 0, j = 0, \dots, n$. By the forward algorithm

$$\begin{aligned}&z_t^{(0)} = 1,\\ &z_t^{(j)} = z_{t-1}^{(j)}\,\phi(x_t) + z_{t-1}^{(j-1)}\,\mu_j(x_t), \quad j \geq 1.\end{aligned}$$

Applying similar parametrization to that in §4.2, $z_t^{(n)}$ recovers one dimension of the recurrence. Collecting d such WFSAs we recover the recurrence of a single layer n-gram RCNNs, with $\boldsymbol{\lambda}_t$ being a constant, or depending only on x_t. □

C Proof of Proposition 13

Proof. Closely following the 2-dimensional case in §4.3, let's discuss a single layer ISAN of hidden size d.

Consider a WFSA over the real semiring with $2d$ states. Let d of them, denoted by $\mathcal{Q}_2 = \{q_i\}_{i=d+1}^{2d}$ be the initial states, with $\lambda(q_i) = 1, i = 1, \dots, d$. Denote the other half $\{q_i\}_{i=1}^d$ by $\mathcal{Q}_1$. Define transition weight τ by:

$$\tau(q_i, q_j, \alpha) = \begin{cases}1, & i = j,\ q_i \in \mathcal{Q}_2,\\ \eta_j(\alpha), & i = j + d,\\ \mu_{j,i}(\alpha), & q_i, q_j \in \mathcal{Q}_1,\\ 0, & \text{otherwise.}\end{cases}$$

$\forall\alpha \in \Sigma$.

Using $q_i \in \mathcal{Q}_1$ as the final state with $\rho(q_i) = 1$, and denote the resulting WFSA by $\mathcal{G}_i$. By Forward algorithm, $\mathcal{G}_i$ recovers the ith dimension of the single layer ISAN by letting $[\mathbf{W}_{x_t}]_{i,j} = \mu_{i,j}(x_t)$, and $[\mathbf{b}_{x_t}]_i = \eta_i(x_t)$; the d-dimensional recurrent computation is recovered by a set of WFSAs $\{\mathcal{G}_i\}_{i=1}^d$ constructed similarly. □

D Compared Models

This section formally describes the models compared in the experiments (§6.1).

RRNN($\mathcal{B}$). RRNN($\mathcal{B}$) is derived from $\mathcal{B}$ (§4.1).

$$\mathbf{f}_t = \boldsymbol{\sigma}\big(\mathbf{W}_f\mathbf{v}_t + \mathbf{b}_f\big), \tag{23a}$$
$$\mathbf{u}_t = (1 - \mathbf{f}_t) \odot \mathbf{W}_u\mathbf{v}_t, \tag{23b}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{u}_t, \tag{23c}$$
$$\mathbf{o}_t = \boldsymbol{\sigma}\big(\mathbf{W}_o\mathbf{v}_t + \mathbf{b}_o\big), \tag{23d}$$
$$\mathbf{h}_t = \tanh(\mathbf{o}_t \odot \mathbf{c}_t). \tag{23e}$$

RRNN($\mathcal{B}$)$_{\mathbf{m+}}$. Also derived from $\mathcal{B}$, but uses the max-plus semiring (§5.2).

$$\mathbf{f}_t = \log \boldsymbol{\sigma}\big(\mathbf{W}_f \mathbf{v}_t + \mathbf{b}_f\big), \tag{24a}$$
$$\mathbf{u}_t = \mathbf{W}_u \mathbf{v}_t, \tag{24b}$$
$$\mathbf{c}_t = \max\{\mathbf{f}_t + \mathbf{c}_{t-1}, \mathbf{u}_t\}, \tag{24c}$$
$$\mathbf{o}_t = \log \boldsymbol{\sigma}\big(\mathbf{W}_o \mathbf{v}_t + \mathbf{b}_o\big), \tag{24d}$$
$$\mathbf{h}_t = \tanh(\mathbf{o}_t + \mathbf{c}_t). \tag{24e}$$

RRNN($\mathcal{C}$). RRNN($\mathcal{C}$) is derived from $\mathcal{C}$ (§4.2):

$$\mathbf{f}_t^{(j)} = \boldsymbol{\sigma}\big(\mathbf{W}_f^{(j)} \mathbf{v}_t + \mathbf{b}_f^{(j)}\big), \quad j = 1, 2, \tag{25a}$$
$$\mathbf{u}_t^{(j)} = (1 - \mathbf{f}_t^{(j)}) \odot \mathbf{W}_u^{(j)} \mathbf{v}_t, \quad j = 1, 2, \tag{25b}$$
$$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \mathbf{f}_t^{(1)} + \mathbf{u}_t^{(1)}, \tag{25c}$$
$$\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \mathbf{f}_t^{(2)} + \mathbf{c}_{t-1}^{(1)} \odot \mathbf{u}_t^{(2)}, \tag{25d}$$
$$\mathbf{o}_t = \boldsymbol{\sigma}\big(\mathbf{W}_o \mathbf{v}_t + \mathbf{b}_o\big), \tag{25e}$$
$$\mathbf{h}_t = \tanh(\mathbf{o}_t \odot \mathbf{c}_t). \tag{25f}$$

RRNN($\mathcal{F}$). Derived from $\mathcal{F}$ (§5.1).

$$\mathbf{f}_t^{(j)} = \boldsymbol{\sigma}\big(\mathbf{W}_f^{(j)} \mathbf{v}_t + \mathbf{b}_f^{(j)}\big), \quad j = 1, 2, \tag{26a}$$
$$\mathbf{u}_t^{(j)} = (1 - \mathbf{f}_t^{(j)}) \odot \mathbf{W}_u^{(j)} \mathbf{v}_t, \quad j = 1, 2, \tag{26b}$$
$$\mathbf{p}^{(j)} = \boldsymbol{\sigma}(\mathbf{b}_p^{(j)}), \quad j = 1, 2, \tag{26c}$$
$$\mathbf{r} = \boldsymbol{\sigma}(\mathbf{b}_r), \tag{26d}$$
$$\mathbf{c}_t^{(1)} = \mathbf{c}_{t-1}^{(1)} \odot \mathbf{f}_t^{(1)} + \mathbf{u}_t^{(1)}, \tag{26e}$$
$$\mathbf{c}_t^{(2)} = \mathbf{c}_{t-1}^{(2)} \odot \mathbf{f}_t^{(2)} + (\mathbf{c}_{t-1}^{(1)} + \mathbf{r}) \odot \mathbf{u}_t^{(2)}, \tag{26f}$$
$$\mathbf{c}_t = \mathbf{p}^{(1)} \odot \mathbf{c}_t^{(1)} + \mathbf{p}^{(2)} \odot \mathbf{c}_t^{(2)} \tag{26g}$$
$$\mathbf{o}_t = \boldsymbol{\sigma}\big(\mathbf{W}_o \mathbf{v}_t + \mathbf{b}_o\big), \tag{26h}$$
$$\mathbf{h}_t = \tanh(\mathbf{o}_t \odot \mathbf{c}_t). \tag{26i}$$

The output gates (Equations 23d, 24d, 25e, and 26h) are optional. They are only used in language modeling experiments, where we empirically find that they improve performance.

E Experimental Setup

E.1 Implementation Details

Our implementation is based on Lei et al. (2017b)[11] and Peng et al. (2018),[12] using PyTorch.[13]

E.2 Language Modeling

For hyperparameters, we do not deviate much from the language modeling experiments in Lei et al. (2017b). We change the hidden sizes for all compared models based on the trainable parameter budget, and adjust the dropout probabilities accordingly to keep the number of remaining hidden units is roughly the same in expectation. Besides, we observe that RRNN($\mathcal{C}$) and RRNN($\mathcal{F}$) fail to converge when optimized with the SGD algorithm using 1.0 initial learning rate. And thus we use 0.5 for both models. Other hyperparameters are kept the same as Lei et al. (2017b).

Type	**Values**
Hidden size	$[100, 300]$
Vertical dropout	$[0.0, 0.5]$
Recurrent dropout	$[0.0, 0.5]$
Embedding dropout	$[0.0, 0.5]$
Learning Rate	$[10^{-2}, 10^{-4}]$
ℓ_2 regularization	$[10^{-5}, 10^{-7}]$
Gradient Clipping	$[1.0, 5.0]$

Table 6: The hyperparameters explored using random search algorithm in the text classification experiments.

E.3 Text classification

We train our models using Adam (Kingma and Ba, 2015) with a batch size of 16 (for Amazon) or 64 (for the smaller datasets). Initial learning rate and ℓ_2 regularization are hyperparameters. We use 300-dimensional GloVe 840B embeddings (Pennington et al., 2014) normalized to unit length and fixed, replacing unknown words with a special UNK token. Two layer RNNs are used in all cases. For regularization, we use three types of dropout: a recurrent variational dropout, vertical dropout and a dropout on the embedding layer.

We tune the hyperparameters of our model on the development set by running 20 epochs of random search. We then take the best development configuration, and train five models with it using different random seeds. We report the average test results. The hyperparameters values explored are summarized in Table 6. We train all models for 500 epochs, stopping early if development accuracy does not improve for 30 epochs. During training, we halve the learning rate if development accuracy does not improve for 10 epochs.

[11] https://github.com/taolei87/sru
[12] https://github.com/Noahs-ARK/SPIGOT
[13] https://pytorch.org/