Recurrent Additive Networks

Kenton Lee[†*] **Omer Levy**[†*] **Luke Zettlemoyer**[†‡]

[†]Paul G. Allen School, University of Washington, Seattle, WA
[‡]Allen Institute for Artificial Intelligence, Seattle, WA
{kentonl,omerlevy,lsz}@cs.washington.edu

Abstract

We introduce *recurrent additive networks* (RANs), a new gated RNN which is distinguished by the use of purely additive latent state updates. At every time step, the new state is computed as a gated component-wise sum of the input and the previous state, without any of the non-linearities commonly used in RNN transition dynamics. We formally show that RAN states are weighted sums of the input vectors, and that the gates only contribute to computing the weights of these sums. Despite this relatively simple functional form, experiments demonstrate that RANs outperform both LSTMs and GRUs on benchmark language modeling problems. This result shows that many of the non-linear computations in LSTMs and related networks are not essential, at least for the problems we consider, and suggests that the gates are doing more of the computational work than previously understood.

1 Introduction

Gated recurrent neural networks (GRNNs), such as long short-term memories (LSTMs) (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRUs) (Cho et al., 2014), have become ubiquitous in natural language processing (NLP). GRNN's widespread popularity is at least in part due to their ability to model crucial language phenomena such as word order (Adi et al., 2017), syntactic structure (Linzen et al., 2016), and even long-range semantic dependencies (He et al., 2017). Like simple recurrent neural networks (S-RNNs) (Elman, 1990), they are able to learn non-linear functions of arbitrary-length input sequences, while at the same time alleviating the problem of vanishing gradients (Bengio et al., 1994) by including gated additive state updates. While GRNNs work well in practice for a wide range of tasks, it is often difficult to interpret what function they have learned.

In this paper, we introduce a new GRNN architecture that is much simpler than existing approaches (e.g. fewer parameters and fewer non-linearities), produces highly interpretable outputs, and improves performance significantly on benchmark language modeling tasks. More specifically, we propose *recurrent additive networks* (RANs), which are distinguished by their use of purely additive latent state updates. At every time step, the new state is computed as a gated component-wise sum of the input and the previous state. Unlike almost all existing RNNs, there are no non-linearities in the transition dynamics.

One advantage of simplifying the dynamics this way is that we can formally characterize the space of functions RANs compute. It is easy to show that the internal state of a RAN at each time step is simply a component-wise weighted sum of the input vectors up to that time. The only non-linearities come from the gates, which specify the weights for this sum. Because all computations are component-wise, RANs can directly select which *part* of each input element to retain at each time step, and in which proportions, leading to a highly expressive yet interpretable model.

[*]The first two authors contributed equally to this paper.

Despite their relatively simplicity, RANs significantly outperform both LSTMs and GRUs on two language modeling benchmarks and yield comparable results on a third character-based language modeling task with far fewer parameters. To better understand this result, we show that it is possible to derive the RAN updates from LSTM equations by sequentially ablating two recurrent non-linearities. Experiments show that both simplifications improve performance, and suggest that additive connections, rather than the non-linear transition dynamics, are the driving force behind LSTM's success.

2 Recurrent Additive Networks

In this section, we first formally define the RAN model and then show that it represents a relatively simple class of additive functions over the input vectors.

2.1 Model Definition

We assume a sequence of input vectors $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, and define a network that produces a sequence of output vectors $\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n\}$. All recurrences over time are mediated by a sequence of state vectors $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n\}$, computed as follows for each time step t:[2]

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \boldsymbol{W}_{cx}\boldsymbol{x}_t \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ic}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fc}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= g(\boldsymbol{c}_t)
\end{aligned} \tag{1}$$

The new state $\boldsymbol{c}_t$ is simply a weighted sum where two gates, $\boldsymbol{i}_t$ (input) and $\boldsymbol{f}_t$ (forget), control the mixing of the content layer $\widetilde{\boldsymbol{c}}_t$ and the previous state $\boldsymbol{c}_{t-1}$. The *content layer* $\widetilde{\boldsymbol{c}}_t$ is a linear transformation $\boldsymbol{W}_x$ over the input $\boldsymbol{x}_t$, which is useful when the number of input dimensions d_i is different from the number of hidden dimensions d_h (e.g. embedding one-hot vectors). We also include a non-recurrent *output layer* $\boldsymbol{h}_t$ computed with a function g of the internal state $\boldsymbol{c}_t$. In our experiments, we use both $g = \tanh$ and the identity function. The weight matrices $\boldsymbol{W}_*$ are free trainable parameters.

The content layer $\widetilde{\boldsymbol{c}}_t$ and the output function g are presented above to be consistent with other GRNN notation. However, the content layer $\widetilde{\boldsymbol{c}}_t$ in RANs is very simple and only serves to allow different input vector and state vector dimensions. Similarly, the output function g is not recurrent and should be considered external to the central recurrent computations. When these are not used, the RAN can be reduced to an even simpler form:

$$\begin{aligned}
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ic}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fc}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \boldsymbol{x}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1}
\end{aligned} \tag{2}$$

Unlike LSTMs and GRUs, RANs only use additive connections to update the latent state $\boldsymbol{c}_t$. RANs also use relatively fewer parameters. For example, the number of parameters (omitting biases) in an LSTM are $4d_h^2 + 4d_h d_i$, but only $2d_h^2 + 3d_h d_i$ in a RAN. In Section 4, we explore their relationships more closely, showing that RANs are a significantly simplified variation of both LSTMs and GRUs. These simplifications, perhaps surprisingly, improve performance for language modeling (Section 3). They also lead to a highly interpretable model that can be careful analyzed, as we present in more detail in the rest of this section.

2.2 Analysis

Another advantage of the relative simplicity of RANs is that we can formally characterize the space of functions that are used to compute the hidden states $\boldsymbol{c}_t$. In particular, each state is a *component-wise*

[2]Throughout this paper, we omit bias terms for brevity. However, bias terms are always present. Operations such as $\boldsymbol{W}_x\boldsymbol{x}_t$ should always be interpreted as $\boldsymbol{W}_x\boldsymbol{x}_t + \boldsymbol{b}_x$.

weighted sum of the input with the form:

$$\begin{aligned}\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \boldsymbol{x}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} \\ &= \sum_{j=1}^{t} \Big(\boldsymbol{i}_j \circ \prod_{k=j+1}^{t} \boldsymbol{f}_k \Big) \circ \boldsymbol{x}_t \\ &= \sum_{j=1}^{t} \boldsymbol{w}_j^t \circ \boldsymbol{x}_t \end{aligned} \tag{3}$$

when considering the simpler RAN described in equation set (2).[3] Each weight $\boldsymbol{w}_j^t$ is a product of the input gate $\boldsymbol{i}_j$ (when its respective input $\boldsymbol{x}_j$ was read) and every subsequent forget gate $\boldsymbol{f}_k$. An interesting property of these weights is that, like the gates, they are also soft component-wise binary filters. This also produces a highly interpretable model, where each component of each state can be directly traced back to the inputs that contributed the most to its sum.

3 Language Modeling Experiments

We compare the RANs' performance to LSTMs, GRUs, and S-RNNs on three benchmark language modeling tasks.

Benchmarks We use three language modeling benchmarks: Penn Treebank (PTB) (Marcus et al., 1993), Google's billion-word benchmark (BWB) (Chelba et al., 2014), and the Text8 character-based language modeling benchmark.[4] For PTB and BWB, our vocabulary contained the top 10,000 words in each training set, while other tokens were marked as `<unk>`. Text8 uses 27 character types as input (lowercase a–z and space).

Models We compare various recurrent architectures: simple RNN (S-RNN), GRU, LSTM, and RAN. For RANs, we test two variants: a linear RAN with an identity output function (*identity*), and a non-linear RAN with a $\tanh$ output function ($\tanh$).

To control for external variables, we implement a very lightweight language model around each of the five different RNNs. We represent each input token w_t (word or character) as a 256-dimension embedding $\boldsymbol{x}_t$ ($d_i = 256$). These word embeddings $\boldsymbol{x}_t$ are provided to the RNN to produce the final outputs $\boldsymbol{h}_t$, which always have 1024 dimensions ($d_h = 1024$). Finally, we project $\boldsymbol{h}_t$ onto a vocabulary-sized softmax layer to predict the next token w_{t+1}.

Training We applied 50% dropout (Srivastava et al., 2014) immediately before, after, and inside each of the RNNs (using variational dropout (Gal and Ghahramani, 2016)). We optimized the cross-entropy loss using Adam (Kingma and Ba, 2014), using a batch size of 512. We ran 100 epochs on PTB, 1 epoch on BWB (due to its size), and 20 epochs on Text8.

Results Table 1 shows the performance for different architectures. In both word-based benchmarks (PTB and BWB), non-linear RAN substantially outperforms LSTM and GRU, yielding approximately 10%-20% relative perplexity reduction. On the character-based task, the relative difference between the best model (LSTM) to non-linear RAN is less than 1%; however, RAN reaches its result with about half as many parameters as LSTM. Overall, it appears that RANs are "doing more for less", and that there is a real advantage in using them for language modeling tasks.

Perhaps an even more remarkable result is the fact that *linear* RANs – which use the identity as the output function g and whose only non-linearities are in the gates – are still performing comparably or even better than LSTMs and GRUs. This observation begs the question: how is it possible that a simple weighted sum outperforms LSTMs? In Section 4, we show that LSTMs (and similarly GRUs) are implicitly computing some form of component-wise weighted sum as well, and that this computation is key to their success.

[3]The state is also a linear function of the inputs in the more general form (equation set (1)). However, it is a weighted sum of linearly transformed inputs, instead of a weighted sum of the input vectors themselves.

[4]`http://mattmahoney.net/dc/textdata`

	Simple RNN	**GRU**	**LSTM**	**RAN (identity)**	**RAN (tanh)**
PTB	352.6	147.1	157.7	130.9	**121.0**
BWB	141.0	70.90	70.14	69.96	**62.34**
Text8	6.126	4.023	**3.741**	3.874	3.778
# RNN Parameters	1.31M	3.94M	5.25M	2.89M	2.89M

Table 1: The performance of each RNN on language modeling benchmarks, measured by perplexity. We also display the number of RNN parameters (excluding input and output embeddings) for comparison.

4 The Role of Recurrent Additive Networks in Long Short-Term Memory

The need for LSTMs is typically motivated by the fact that they can ease the vanishing gradient problem found in simple RNNs (S-RNNs) (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997). By introducing cell states that are controlled by a set of gates, LSTMs enable shortcuts through which gradients can flow easily when learning with backpropagation. This mechanism enables learning of long-distance dependencies while preserving the expressive power of recurrent non-linearities provided by S-RNNs.

Rather than viewing the gating mechanism as simply an auxiliary mechanism to address a *learning* problem, we present an alternate of view of LSTMs that emphasizes the *modeling* strengths of the gates. We argue that it is possible to reinterpret LSTMs as a hybrid of two other recurrent architectures: (1) S-RNNs and (2) RANs. More specifically, LSTMs can be seen as computing a content layer using an S-RNN, which is then aggregated into a weighted sum using a RAN.

We show empirically that the recurrent non-linearity provided by S-RNN is not required for language modeling, and that the RAN is in fact performing the heavy lifting. To better understand this result, we derive the RAN using two modifications of the LSTMs: a simplification of the content layer (Section 4.2), and a simplification of the output layer (Section 4.3). Figures 1(a)-1(d) illustrate the derivation process, on which we will elaborate more formally below. We also provide experimental evidence, shown in Table 2, that these simplifications maintain or improve model performance.

4.1 LSTM as a Hybrid of S-RNN and RAN

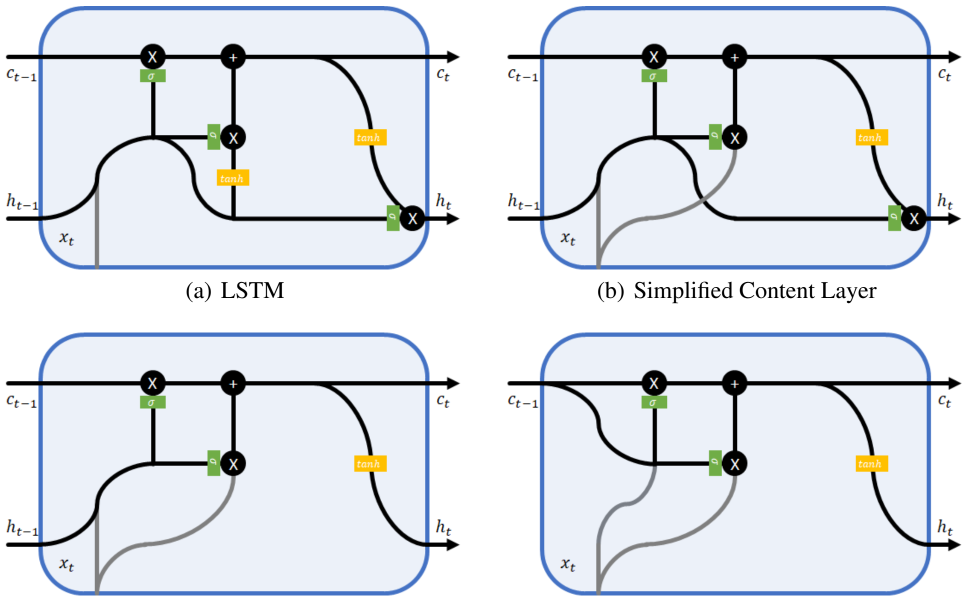
We first demonstrate the two LSTM sub-components of LSTM by dissecting its definition:

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_{ch}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cx}\boldsymbol{x}_t) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{oh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)
\end{aligned} \tag{4}$$

We refer to $\widetilde{\boldsymbol{c}}_t$ as the content layer, which like S-RNNs is a non-linear recurrent layer. The cell state $\boldsymbol{c}_t$ behaves like a RAN, using input and forget gates to compute a weighted sum of the current content layer $\widetilde{\boldsymbol{c}}_t$ and the previous cell state $\boldsymbol{c}_{t-1}$. In fact, just as in RANs, we can express each cell state as a component-wise weighted sum of the all previous content layers:

$$\begin{aligned}
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} \\
&= \sum_{j=0}^{t} \left(\boldsymbol{i}_j \circ \prod_{k=j+1}^{t} \boldsymbol{f}_k\right) \circ \widetilde{\boldsymbol{c}}_j \\
&= \sum_{j=0}^{t} \boldsymbol{w}_j^t \circ \widetilde{\boldsymbol{c}}_j
\end{aligned}$$

However, unlike RANs, we can no longer express the cell state as a weighted sum of the *input* vectors, due to the S-RNN inspired non-linear recurrence.



(a) LSTM (b) Simplified Content Layer

(c) No Output Gate (d) Simplified Output Layer (RAN)

Figure 1: A series of models that progressively transform from an LSTM (a) to an RAN (d). The first step (a to b) simplifies the content layer by removing its recurrence and non-linearity. The second step, which simplifies the output layer, removes the output gate (b to c) and then extracts the remaining $\tanh$ non-linearity from the recurrence by rewiring the gates' inputs (c to d).

4.2 Simplifying the Content Layer

We first examine the necessity of using an embedded S-RNN to compute the content layer. In this step of the derivation, we remove the non-linearity of the content layer $\widetilde{\boldsymbol{c}}_t$ and its dependence on the previous output vector $\boldsymbol{h}_{t-1}$, and replace it with a simple linear projection of the input vector $\widetilde{\boldsymbol{c}}_t = \boldsymbol{W}_x \boldsymbol{x}_t$:

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \boldsymbol{W}_{cx}\boldsymbol{x}_t \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{oh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t)
\end{aligned} \tag{5}$$

This modification results in a content layer $\widetilde{\boldsymbol{c}}_t$ that is identical to the content layer in a RAN. We can also consider this step as the removal of the S-RNN from the LSTM.

In our experiments, we observe that this simplification (*Simplified Content Layer* in Table 2) produces a large improvement in word-based language modeling and near-identical performance in character-based language modeling. This result suggests that the embedded S-RNN is not only unnecessary—it might also be an impediment.

4.3 Simplifying the Output Layer

In the second step, which brings us directly to RANs, we simplify two aspects of the output layer. First, we remove the output gate. Second, we extract the remaining $\tanh$ non-linearity from the recurrent process, making it an effectively external component (like g in RANs). We do so by rewiring the recurrent connection of the gates: instead of depending on the previous output vector $\boldsymbol{h}_{t-1}$, the gates take the previous cell state $\boldsymbol{c}_{t-1}$ as the recurrent argument, which does not pass through the $\tanh$ non-linearity. These two modifications result in the RAN, which we present here again for

	Original LSTM	**Simplified Content Layer**	**RAN (tanh)**
PTB	157.7	**134.7**	**121.0**
BWB	70.14	**66.73**	**62.34**
Text8	3.741	3.797	3.778
# RNN Parameters	5.25M	4.20M	2.89M

Table 2: The performance of each ablated LSTM on language modeling benchmarks, measured by perplexity. Bold figures indicate which ablations outperform the original LSTM. We also display the number of RNN parameters (excluding input and output embeddings) for comparison.

completeness:

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \boldsymbol{W}_{cx}\boldsymbol{x}_t \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ic}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fc}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= \tanh(\boldsymbol{c}_t)
\end{aligned}$$

This simplification of the output layer results in further significant improvements on word-based language modeling, while producing near-identical results for character-based language modeling (*RAN (tanh)* in Table 2).

The final result of this derivation is that both $\tanh$ non-linearities present in LSTMs are either removed or applied in a non-recurrent, modular way. The sparing use of non-linearities allows us to better reason about the computations of the architecture. More importantly, these results suggest that the success of LSTMs is better attributed to the internal structure that is shares with RANs rather than the recurrent non-linearities associated with its internal S-RNN.

4.4 Deriving RAN from GRU

A similar analysis can be applied to other gated RNNs such as GRUs, which are typically defined as follows:

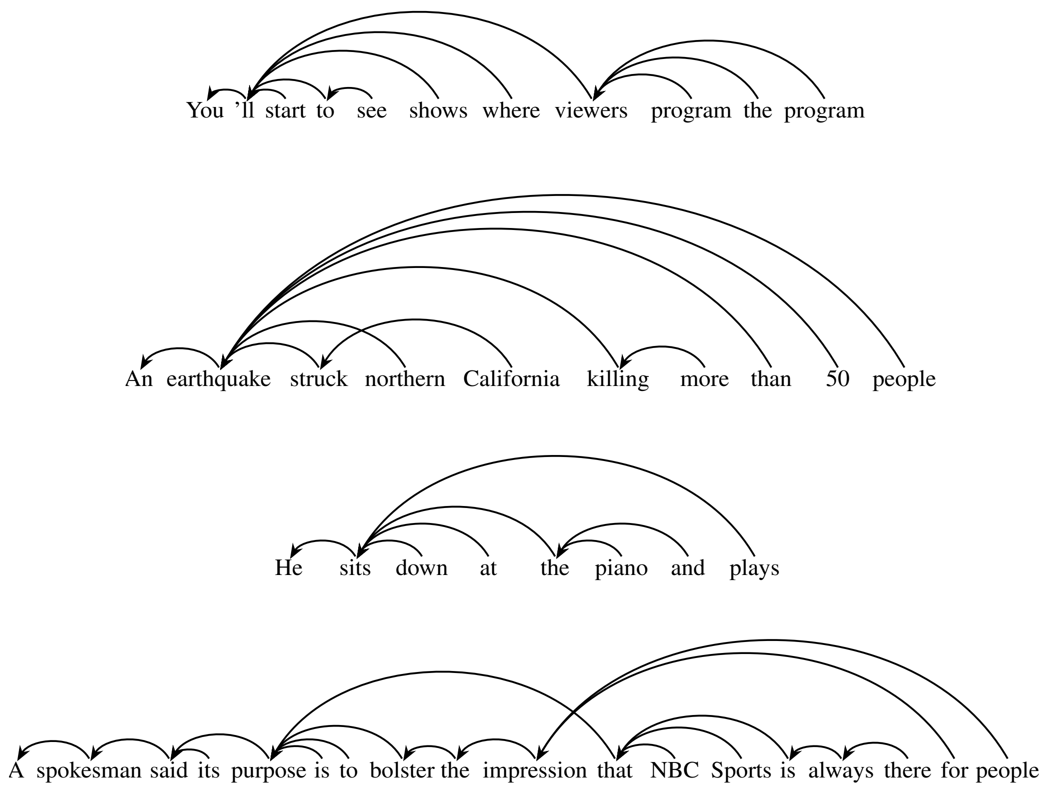
$$\begin{aligned}
\boldsymbol{z}_t &= \sigma(\boldsymbol{W}_{zh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{zx}\boldsymbol{x}_t) \\
\boldsymbol{r}_t &= \sigma(\boldsymbol{W}_{rh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{rx}\boldsymbol{x}_t) \\
\widetilde{\boldsymbol{h}}_t &= \tanh(\boldsymbol{W}_{hh}(r \circ \boldsymbol{h}_{t-1}) + \boldsymbol{W}_{hx}\boldsymbol{x}_t) \\
\boldsymbol{h}_t &= \boldsymbol{z}_t \circ \widetilde{\boldsymbol{h}}_t + (\boldsymbol{1} - \boldsymbol{z}_t) \circ \boldsymbol{h}_{t-1}
\end{aligned} \tag{6}$$

For ease of discussion, we present the following non-standard but equivalent form of GRUs, which aligns better with the notation from LSTMs and RANs:

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_{ch}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cx}\boldsymbol{x}_t) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ic}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{oc}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + (\boldsymbol{1} - \boldsymbol{i}_t) \circ \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \boldsymbol{c}_t
\end{aligned} \tag{7}$$

In this form, the content layer $\widetilde{\boldsymbol{c}}_t$ is equivalent to $\widetilde{\boldsymbol{h}}_t$. The input gate $\boldsymbol{i}_t$ is equivalent to $\boldsymbol{z}_t$, and the output gate $\boldsymbol{o}_t$ is equivalent to the reset gate $\boldsymbol{r}_t$. A subtle detail that enables this transformation is that GRUs can be seen as maintaining separate cell and output states, similar to LSTMs. However, GRUs compute gates with respect to the previous cell state $\boldsymbol{c}_{t-1}$ rather than the previous output $\boldsymbol{h}_{t-1}$, similar to RANs.

In our alternate form, it is clear that GRUs also accumulate weighted summations $\boldsymbol{c}_t$ of previous content layers $\widetilde{\boldsymbol{c}}_t$. The derivation of RAN from here involves only two straightforward steps: (1) the non-linear recurrence of the content layer $\widetilde{\boldsymbol{c}}_t$ is replaced by a simple linear projection $\widetilde{\boldsymbol{c}}_t = \boldsymbol{W}_{cx}\boldsymbol{x}_t$,



You 'll start to see shows where viewers program the program
An earthquake struck northern California killing more than 50 people
He sits down at the piano and plays
A spokesman said its purpose is to bolster the impression that NBC Sports is always there for people

Figure 2: Partial visualizations of the weights in the weighted sum computed by an RAN ($\boldsymbol{w}_j^t$ in equation set (3)). The arrows indicate for each word t which previous word has the highest weighted component (v_t in equation (8)).

and (2) the output gate $\boldsymbol{o}_t$ is used instead as a forget gate by applying it to the previous cell state $\boldsymbol{c}_{t-1}$ rather than the current cell state $\boldsymbol{c}_t$:

$$\boldsymbol{c}_t = \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{o}_t \circ \boldsymbol{c}_{t-1}$$

The result is a RAN with an identity output function.

An interesting effect of the coupled gates in GRU is that these weights are normalized, i.e. the weights over all previous time steps sum to $\mathbf{1}$. As a result, GRUs (and our simplifications of them) are computing weighted *averages* rather than weighted *sums*. In development experiments, we found that this normalization hurt performance if added to the RAN architecture. However, it is closely related to another architecture that computes weighted averages: attention (Bahdanau et al., 2015). In fact, we can consider a GRU as a *component-wise* attention mechanism over its previous content layers and an RAN to be an unnormalized component-wise generalization of attention, which can point to many different possible inputs independently.

5 Weight Visualization

Given the relative interpretability of the RAN predictions, we can trace the importance of each component in each of the previous elements explicitly, as shown in equation set (3). In language modeling, we can also compute which previous word had the overall strongest influence in predicting the next word at time step t:

$$v_t = \operatorname*{argmax}_{j=1}^{t-1} \big(\max_{m=1}^{d_h} (\boldsymbol{w}_j^t(m)) \big) \tag{8}$$

In Figure 2, we visualize this computation by drawing arrows from each word t to its most influential predecessor (i.e. the previous word with the highest weighted component). In these four example sentences, we see that RANs can recover long distance dependencies that make intuitive sense given the syntactic and semantic structure of the text. For example, verbs are related to their arguments, even when coordinated, and nouns in relative clauses depend on the noun they are modifying. This illustration provides only a glimpse into what the model is capturing, and perhaps future, more detailed visualizations that take the individual components into account can provide further insight into what RANs are learning in practice.

6 Related Work

Many variants of LSTMs (Hochreiter and Schmidhuber, 1997) and alternative designs for more general GRNNs have been proposed. One such variant adds peephole connections between the cell state and the gates (Gers and Schmidhuber, 2000). GRUs (Cho et al., 2014) decrease the number of parameters by coupling the input and forget gates and rewiring the gate computations (see Section 4.4 for an alternative formulation). Greff et al. (2016) conducted an LSTM ablation study that probed the importance of each component independently, while Józefowicz et al. (2015) took an automatic approach to the task of architecture design, and found additional variants of GRNNs. While we demonstrate that RANs can be seen as an ablation of LSTMs or GRUs, they are vastly simpler than the variants explored in the aforementioned studies.

Several approaches represent sequences as weighted sums of their elements. Perhaps the most popular mechanism in NLP is attention (Bahdanau et al., 2015), which assigns a normalized scalar weight to each element (typically a word vector) as a function of its compatibility with an external element. The ability to inspect attention weights has driven the use of more interpretable neural models. Self-attention (Cheng et al., 2016; Parikh et al., 2016) extends this notion by computing intra-sequence attention. Recently, Arora et al. (2017) proposed a theory-driven approach to assign scalar weights to elements in a bag of words. RANs, on the other hand, implicitly compute a component-wise weighted sum as a byproduct of a simple RNN state scheme.

7 Conclusion

We introduced recurrent additive networks (RANs), a type of gated RNNs that performs purely additive updates to its latent state. While RANs do not include the non-linear recurrent connections that are typically considered to be crucial for RNNs, they have remarkably strong performance on several language modeling benchmarks compared to other popular recurrent architectures, such as LSTMs and GRUs. RANs are also considerably more transparent than other RNNs; their limited use of non-linearities enables the state vector to be expressed as a component-wise weighted sum of previous input vectors. This also allows the individual impact of the sequence inputs on the state to be recovered explicitly, directly pointing to which factors are most influencing each part of the current state.

This work also sheds light on the inner workings of existing, more opaque models, primarily LSTMs and GRUs. We demonstrate that RAN-like components exist within LSTMs and GRUs. Furthermore, we provide empirical evidence that the use of recurrent non-linearities within those architectures is not only unhelpful, but that it can become a liability in language modeling.

While we demonstrate the robustness of RANs on several language modeling benchmarks, it is an open question whether these findings will generalize across a wider variety of tasks. However, the results do suggest that it may be possible to develop related, relatively simple, additive gated RNNs that are better building blocks for a wide range of different neural architectures. We hope that our findings prove helpful in the design of future recurrent neural networks.

Acknowledgements

The research was supported in part by DARPA under the DEFT program (FA8750-13-2-0019), the ARO (W911NF-16-1-0121), the NSF (IIS-1252835, IIS-1562364), gifts from Google and Tencent, and an Allen Distinguished Investigator Award. We also thank Yoav Goldberg, Luheng He, Aaron Jaech, Ariel Holtzman, and the UW NLP group for helpful conversations and comments on the work.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *ICLR*, 2017.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*, 2017.

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

Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. In *INTERSPEECH*, 2014.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas, November 2016. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D16-1053`.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D14-1179`.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1019–1027, 2016.

Felix A. Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *IJCNN*, 2000.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2016.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: What works and what's next. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

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

Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *ICML*, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *TACL*, 4:521–535, 2016.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330, 1993.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255, Austin, Texas, November 2016. Association for Computational Linguistics. URL `https://aclweb.org/anthology/D16-1244`.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.