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 perform on par with LSTMs 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) and produces highly interpretable outputs, while matching the robust performance of LSTMs 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, non-linearities affect the recurrent state only by controlling the gates at each timestep.

One advantage of simplifying the transition 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. Because all computations are component-wise, RANs can directly select which *part* of each input element to retain at each time step, leading to a highly expressive yet interpretable model.

Despite their relative simplicity, RANs perform as well as LSTMs and related architectures on three language modeling benchmarks, but with far fewer parameters. To better understand this result, we

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

derive the RAN updates from LSTM equations by (1) removing the recurrent non-linearity from $\widetilde{c}_t$ (LSTM's internal S-RNN) and (2) by removing the output gate. Experiments show that we maintain the same level of performance after both simplifications, suggesting 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:

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

The new state $\boldsymbol{c}_t$ is 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 an *output layer* $\boldsymbol{h}_t$ computed with a function g of the internal state $\boldsymbol{c}_t$. In our experiments, we use $g(x) = \tanh(x)$ and the identity function $g(x) = x$. The matrices $\boldsymbol{W}_*$ and biases $\boldsymbol{b}_*$ are free trainable parameters.

The content layer $\widetilde{\boldsymbol{c}}_t$ and 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 can be the identity, merging $\boldsymbol{h}_t$ with $\boldsymbol{c}_t$. When the content and output layers are trivial, the RAN can be reduced to an even simpler form:

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

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, perform just as well as LSTMs on 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 and in Section 5.

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 weighted sum of the input* with the form:

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

when considering the simpler RAN described in equation set (2).[2] 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 on three benchmark language modeling tasks. For each dataset, we use previously reported hyperparameter settings that were tuned for LSTMs in order to ensure a fair comparison (Section 3.1). Our experiments show that RANs perform on par with LSTMs on all three benchmarks (Section 3.2). The code and settings to replicate these experiments is publicly available.[3]

3.1 Experiment Setup

Penn TreeBank The Penn Treebank (PTB) (Marcus et al., 1993) is a popular language-modeling benchmark, containing approximately 1M tokens over a vocabulary of 10K words. We used the implementation of Zaremba et al. (2014) while replacing any invocation of LSTMs with RANs. We tested two configurations: *medium*, which uses two layers of 650-dimension RANs, and *large*, which uses two layers of 1500-dimension RANs. Word embedding size is set to match the recurrent layers' size, and dropout (Srivastava et al., 2014) is used throughout the network. Both settings use stochastic gradient descent (SGD) to optimize the model, each with a unique hyperparameter setting to gradually decrease the learning rate.[4]

Billion-Word Benchmark Google's billion-word benchmark (BWB) (Chelba et al., 2014) is about a thousand times larger than PTB, and uses a more diverse vocabulary of 800K words. Using the implementation of Józefowicz et al. (2016), we tested the *LSTM-2048-512* configuration, which uses a single-layered LSTM of 2048 hidden dimensions and a word embedding space of 512 dimensions. In our experiments, the LSTM was replaced with a RAN, while reusing exactly the same hyperparameters (dimensions, dropout, learning rates, etc) that were originally tuned for LSTMs by Jozefowicz et al. Following their implementation, we project the hidden state at each time step down to 512 dimensions. Due to the enormous size of this dataset, we stopped training after 5 epochs.

Text8 We also evaluated on a character-based language modeling benchmark, Text8.[5] This dataset is made of the first 100M characters in English Wikipedia after some filtering process. The vocabulary contains only 27 characters (lowercase a–z and space). We adapted the implementation of Zilly et al. (2017) by taking hyperparameter settings from Chung et al. (2017), which had an LSTM-like architecture. Their setting used 128-dimension character embeddings, followed by 3 LSTM layers of 1024, 1024, and 2048 dimensions respectively, which we adapted to RANs of the same dimensions. We present only tanh RANs for this benchmark because identity RANs were unstable with these hyperparameters.

3.2 Results

We first compare the performance of RANs to that of LSTMs on the word-based language modeling benchmarks, PTB (Table 1) and BWB (Table 2). In all configurations, the change in performance is below 1% relative difference between LSTMs and tanh RANs. It appears that despite using less than two-thirds the parameters, RANs can perform on par with LSTMs. The BWB result is particularly interesting, because it demonstrates RANs' ability to to leverage big data like LSTMs.

[2]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.

[3]`http://www.github.com/kentonl/ran`

[4]The only modifications we made from the original setting was to use lower initial learning rates to improve stability. Exact values are provided in the code.

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

Configuration	Model	Perplexity	# RNN Parameters
Medium	LSTM (Zaremba et al., 2014)	82.7	6.77M
	tanh RAN	81.9	4.23M
	identity RAN	85.5	4.23M
Large	LSTM (Zaremba et al., 2014)	78.4	36.02M
	tanh RAN	78.5	22.52M
	identity RAN	83.2	22.52M

Table 1: The performance of RAN and LSTM on the Penn TreeBank (PTB) benchmark, measured by perplexity. We also display the number of RNN parameters for comparison.

Model	Perplexity	# RNN Parameters
LSTM (Józefowicz et al., 2016)	47.5	9.46M
tanh RAN	47.9	6.30M
identity RAN	47.2	6.30M

Table 2: The performance of RAN and LSTM on Google's billion-word benchmark (BWB), measured by perplexity after 5 epochs. We also display the number of RNN parameters for comparison.

Perhaps an even more remarkable result is the fact that identity RANs – which, excluding the gates, compute a *linear* function of the input – are also performing comparably to LSTMs on the BWB. This observation begs the question: how is it possible that a simple weighted sum performs just as well as an LSTM? 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.

Finally, we compare the performance of RANs to a variety of LSTM variants on the character-based language modeling task Text8 (Table 3). While the different results vary both in model and in hyperparameters, the same trend in which RANs perform similarly to LSTMs emerges yet again.

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 (Section 4.1). To better understand this composition, we show how a RAN can be derived by simplifying an LSTM (Section 4.2) or a GRU (Section 4.3). Given our empirical observations in Section 3, which show that RANs perform comparably with LSTMs, it appears 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.

Model	BPC
td-LSTM (Zhang et al., 2016)	1.49
MI-LSTM (Wu et al., 2016)	1.44
mLSTM (Krause et al., 2017)	1.40
tanh RAN	1.38
BatchNorm LSTM (Cooijmans et al., 2017)	1.36
LayerNorm HM-LSTM (Chung et al., 2017)	1.29
RHN (Zilly et al., 2017)	1.27

Table 3: The performance of RAN and recently-published LSTM variants on the Text8 character-based language modeling benchmark, measured by bits per character (BPC).

4.1 LSTM as a Hybrid of S-RNN and RAN

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

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

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, the content layer $\widetilde{c_j}$ depends on both the current input and the previous state cell state. Therefore, LSTMs cannot express the cell state as a weighted sum of the *input* vectors.

4.2 Deriving RAN from LSTM

To derive an RAN from LSTM's equations, we perform two steps: (1) simplify the output layer by removing the output gate, and (2) simplify the content layer using only a linear projection of the input.

Simplifying the Output Layer In the first step, we simply remove the output gate, and abstract the output non-linearity $\tanh$ with g:

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_{ch}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cx}\boldsymbol{x}_t + \boldsymbol{b}_c) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t + \boldsymbol{b}_f) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= g(\boldsymbol{c}_t)
\end{aligned}$$

This step follows previous architectural ablations of LSTM (Józefowicz et al., 2015; Greff et al., 2016), which demonstrated that removing the output gate has limited effects.

Simplifying the Content Layer We proceed to remove the embedded S-RNN from the LSTM by eliminating the non-linearity of the content layer $\widetilde{\boldsymbol{c}}_t$ and its dependence on the previous output vector $\boldsymbol{h}_{t-1}$. Specifically, we replace the original content layer with a simple linear projection of the input vector $\widetilde{\boldsymbol{c}}_t = \boldsymbol{W}_x \boldsymbol{x}_t$, resulting in the RAN architecture:

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \boldsymbol{W}_{cx}\boldsymbol{x}_t \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i) \\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t + \boldsymbol{b}_f) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= g(\boldsymbol{c}_t)
\end{aligned}$$

This step emphasizes the most significant difference between RANs and other RNNs. The fact that removing the embedded S-RNN does not harm performance on our benchmarks in any significant way suggests that it is perhaps unnecessary, and that the gating mechanism has sufficient modeling capacity of its own for language modeling.

4.3 Deriving RAN from GRU

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

$$\begin{aligned}
\boldsymbol{z}_t &= \sigma(\boldsymbol{W}_{zh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{zx}\boldsymbol{x}_t + \boldsymbol{b}_z) \\
\boldsymbol{r}_t &= \sigma(\boldsymbol{W}_{rh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{rx}\boldsymbol{x}_t + \boldsymbol{b}_r) \\
\widetilde{\boldsymbol{h}}_t &= \tanh(\boldsymbol{W}_{hh}(r \circ \boldsymbol{h}_{t-1}) + \boldsymbol{W}_{hx}\boldsymbol{x}_t + \boldsymbol{b}_h) \\
\boldsymbol{h}_t &= \boldsymbol{z}_t \circ \widetilde{\boldsymbol{h}}_t + (\boldsymbol{1} - \boldsymbol{z}_t) \circ \boldsymbol{h}_{t-1}
\end{aligned} \tag{5}$$

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:

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_{ch}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cx}\boldsymbol{x}_t + \boldsymbol{b}_c) \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ic}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{oc}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t + \boldsymbol{b}_o) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + (\boldsymbol{1} - \boldsymbol{i}_t) \circ \boldsymbol{c}_{t-1} \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \boldsymbol{c}_t
\end{aligned} \tag{6}$$

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 $\boldsymbol{r}_t$. In this form, the output vector at each time step is c_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 the content layer with respect to the previous hidden state $\boldsymbol{h}_{t-1}$ rather than the previous output $\boldsymbol{c}_{t-1}$.

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 two straightforward steps. First, the non-linear recurrence of the content layer $\widetilde{\boldsymbol{c}}_t$ is replaced by a simple linear projection $\widetilde{\boldsymbol{c}}_t = \boldsymbol{W}_{cx}\boldsymbol{x}_t$, as we did for LSTMs. This step also makes $\boldsymbol{h}_t$ redundant. Second, we repurpose the output gate $\boldsymbol{o}_t$ as a forget gate by applying it to the previous cell state $\boldsymbol{c}_{t-1}$ rather than the current cell state $\boldsymbol{c}_t$:

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

This step results in a RAN with an identity output function g:

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \boldsymbol{W}_{cx}\boldsymbol{x}_t \\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ic}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i) \\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{oc}\boldsymbol{c}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t + \boldsymbol{b}_o) \\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{o}_t \circ \boldsymbol{c}_{t-1}
\end{aligned}$$

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 $\boldsymbol{1}$. As a result, GRUs are computing weighted *averages* rather than weighted *sums* of their content layer. In development experiments, we found that this

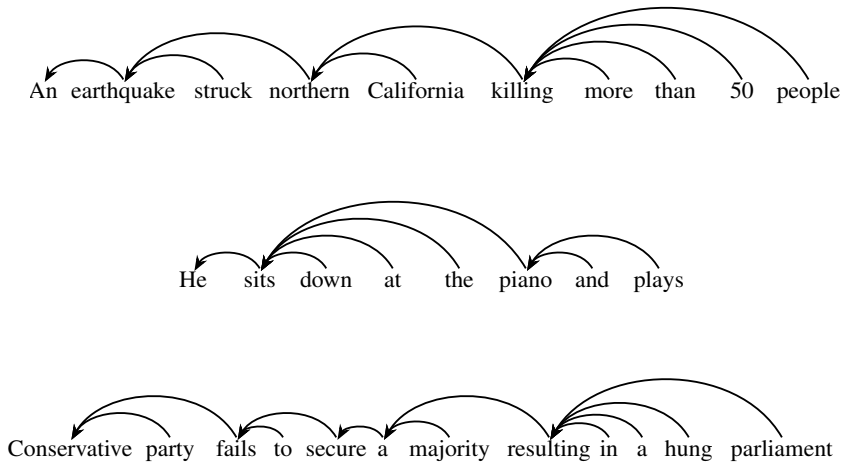

Figure 1: Partial visualizations of the weights in the weighted sum computed by an RAN ($\boldsymbol{w}_j^t$ in equation set (3)). The arrows indicate for each word t which previous word has the highest weighted component (v_t in equation (7)).

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 a 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:

$$v_t = \operatorname*{argmax}_{j=1}^{t-1} \left(\max_{m=1}^{d_h} (\boldsymbol{w}_j^t(m)) \right) \tag{7}$$

In Figure 1, 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 4.3 for an alternative formulation). Greff et al. (2016) conducted an LSTM ablation study that probed the importance of each component independently, while others (Józefowicz et al., 2015; Zoph and Le, 2017) took an automatic approach to the task of architecture design, finding 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. Specifically, this is the first study to remove the recurrent non-linearity (the embedded simple RNN) from such models.

Concurrently, Lei et al. (2017) arrived at a similar space of models as RANs by deriving recurrent neural networks from string kernels, which they call kernel neural networks (KNNs). Our studies complement each other in two ways. First, Lei et al. show the connection between RANs/KNNs and kernels, while we show the connection to LSTMs and GRUs. Second, we conduct somewhat different experiments; we compare RANs to LSTMs in standard settings that were tuned for LSTMs, while Lei et al. show that RANs can achieve state-of-the-art performance when the network's design is more flexible and can be fitted with the latest advances in language modeling.

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. Vaswani et al. (2017) further showed that state-of-the-art machine translation can be achieved using only self-attention, without the use of recurrent non-linearities found in LSTMs. Recently, Arora et al. (2017) proposed a theory-driven approach to assign scalar weights to elements in a bag of words. While these methods assign scalar weights to each input explicitly, RANs 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 similar performance to LSTMs on several language modeling benchmarks. 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 perhaps unnecessary for 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, Tencent, and Nvidia, and an Allen Distinguished Investigator Award. We also thank Yoav Goldberg, Benjamin Heinzerling, Tao Lei, 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`.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. In *ICLR*, 2017.

Tim Cooijmans, Nicolas Ballas, César Laurent, and Aaron C. Courville. Recurrent batch normalization. In *ICLR*, 2017.

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

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.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Ben Krause, Iain Murray, Steve Renals, and Liang Lu. Multiplicative LSTM for sequence modelling. In *ICLR Workshop*, 2017.

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

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan Salakhutdinov. On multiplicative integration with recurrent neural networks. In *NIPS*, 2016.

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

Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan Salakhutdinov, and Yoshua Bengio. Architectural complexity measures of recurrent neural networks. In *NIPS*, 2016.

Julian G. Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *ICLR*, 2017.

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