How to Construct Deep Recurrent Neural Networks

Razvan Pascanu[1], Caglar Gulcehre[1], Kyunghyun Cho[2], and Yoshua Bengio[1]

[1]Département d'Informatique et de Recherche Opérationelle, Université de Montréal,
{pascanur, gulcehrc}@iro.umontreal.ca, yoshua.bengio@umontreal.ca
[2]Department of Information and Computer Science, Aalto University School of Science,
kyunghyun.cho@aalto.fi

Abstract

In this paper, we explore different ways to extend a recurrent neural network (RNN) to a *deep* RNN. We start by arguing that the concept of depth in an RNN is not as clear as it is in feedforward neural networks. By carefully analyzing and understanding the architecture of an RNN, however, we find three points of an RNN which may be made deeper; (1) input-to-hidden function, (2) hidden-to-hidden transition and (3) hidden-to-output function. Based on this observation, we propose two novel architectures of a deep RNN which are orthogonal to an earlier attempt of stacking multiple recurrent layers to build a deep RNN (Schmidhuber, 1992; El Hihi and Bengio, 1996). We provide an alternative interpretation of these deep RNNs using a novel framework based on neural operators. The proposed deep RNNs are empirically evaluated on the tasks of polyphonic music prediction and language modeling. The experimental result supports our claim that the proposed deep RNNs benefit from the depth and outperform the conventional, shallow RNNs.

1 Introduction

Recurrent neural networks (RNN, see, e.g., Rumelhart *et al.*, 1986) have recently become a popular choice for modeling variable-length sequences. RNNs have been successfully used for various task such as language modeling (see, e.g., Graves, 2013; Pascanu *et al.*, 2013a; Mikolov, 2012; Sutskever *et al.*, 2011), learning word embeddings (see, e.g., Mikolov *et al.*, 2013a), online handwritten recognition (Graves *et al.*, 2009) and speech recognition (Graves *et al.*, 2013).

In this work, we explore *deep* extensions of the basic RNN. Depth for feedforward models can lead to more expressive models (Pascanu *et al.*, 2013b), and we believe the same should hold for recurrent models. We claim that, unlike in the case of feedforward neural networks, the *depth* of an RNN is ambiguous. In one sense, if we consider the existence of a composition of several nonlinear computational layers in a neural network being deep, RNNs are already deep, since any RNN can be expressed as a composition of multiple nonlinear layers when unfolded in time.

Schmidhuber (1992); El Hihi and Bengio (1996) earlier proposed another way of building a deep RNN by stacking multiple recurrent hidden states on top of each other. This approach potentially allows the hidden state at each level to operate at different timescale (see, e.g., Hermans and Schrauwen, 2013). Nonetheless, we notice that there are some other aspects of the model that may still be considered *shallow*. For instance, the transition between two consecutive hidden states at a single level is shallow, when viewed separately.This has implications on what kind of transitions this model can represent as discussed in Section 3.2.3.

Based on this observation, in this paper, we investigate possible approaches to extending an RNN into a deep RNN. We begin by studying which parts of an RNN may be considered shallow. Then, for each shallow part, we propose an alternative *deeper* design, which leads to a number of deeper variants of an RNN. The proposed deeper variants are then empirically evaluated on two sequence modeling tasks.

The layout of the paper is as follows. In Section 2 we briefly introduce the concept of an RNN. In Section 3 we explore different concepts of *depth* in RNNs. In particular, in Section 3.3.1–3.3.2 we propose two novel variants of deep RNNs and evaluate them empirically in Section 5 on two tasks: polyphonic music prediction (Boulanger-Lewandowski *et al.*, 2012) and language modeling. Finally we discuss the shortcomings and advantages of the proposed models in Section 6.

2 Recurrent Neural Networks

A recurrent neural network (RNN) is a neural network that simulates a discrete-time dynamical system that has an input $\mathbf{x}_t$, an output $\mathbf{y}_t$ and a hidden state $\mathbf{h}_t$. In our notation the subscript t represents time. The dynamical system is defined by

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{1}$$

$$\mathbf{y}_t = f_o(\mathbf{h}_t), \tag{2}$$

where f_h and f_o are a state transition function and an output function, respectively. Each function is parameterized by a set of parameters; $\boldsymbol{\theta}_h$ and $\boldsymbol{\theta}_o$.

Given a set of N training sequences $D = \left\{ \left((\mathbf{x}_1^{(n)}, \mathbf{y}_1^{(n)}), \dots, (\mathbf{x}_{T_n}^{(n)}, \mathbf{y}_{T_n}^{(n)}) \right) \right\}_{n=1}^{N}$, the parameters of an RNN can be estimated by minimizing the following cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T_n} d(\mathbf{y}_t^{(n)}, f_o(\mathbf{h}_t^{(n)})), \tag{3}$$

where $\mathbf{h}_t^{(n)} = f_h(\mathbf{x}_t^{(n)}, \mathbf{h}_{t-1}^{(n)})$ and $\mathbf{h}_0^{(n)} = \mathbf{0}$. $d(\mathbf{a}, \mathbf{b})$ is a predefined divergence measure between $\mathbf{a}$ and $\mathbf{b}$, such as Euclidean distance or cross-entropy.

2.1 Conventional Recurrent Neural Networks

A conventional RNN is constructed by defining the transition function and the output function as

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) = \phi_h \left(\mathbf{W}^\top \mathbf{h}_{t-1} + \mathbf{U}^\top \mathbf{x}_t \right) \tag{4}$$

$$\mathbf{y}_t = f_o(\mathbf{h}_t, \mathbf{x}_t) = \phi_o \left(\mathbf{V}^\top \mathbf{h}_t \right), \tag{5}$$

where $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are respectively the transition, input and output matrices, and ϕ_h and ϕ_o are element-wise nonlinear functions. It is usual to use a saturating nonlinear function such as a logistic sigmoid function or a hyperbolic tangent function for ϕ_h. An illustration of this RNN is in Fig. 2 (a).

The parameters of the conventional RNN can be estimated by, for instance, stochastic gradient descent (SGD) algorithm with the gradient of the cost function in Eq. (3) computed by backpropagation through time (Rumelhart *et al.*, 1986).

3 Deep Recurrent Neural Networks

3.1 Why Deep Recurrent Neural Networks?

Deep learning is built around a hypothesis that a deep, hierarchical model can be exponentially more efficient at representing some functions than a shallow one (Bengio, 2009). A number of recent theoretical results support this hypothesis (see, e.g., Le Roux and Bengio, 2010; Delalleau and Bengio, 2011; Pascanu *et al.*, 2013b). For instance, it has been shown by Delalleau and Bengio (2011) that a deep sum-product network may require exponentially less units to represent the same function compared to a shallow sum-product network. Furthermore, there is a wealth of empirical evidences supporting this hypothesis (see, e.g., Goodfellow *et al.*, 2013; Hinton *et al.*, 2012b,a). These findings make us suspect that the same argument should apply to recurrent neural networks.

3.2 Depth of a Recurrent Neural Network

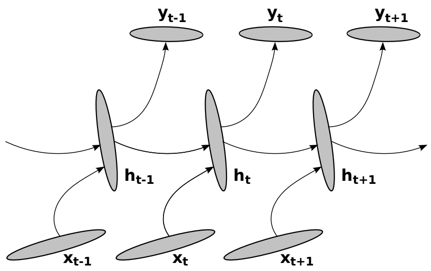


Figure 1: A conventional recurrent neural network unfolded in time.

The *depth* is defined in the case of feedforward neural networks as having multiple nonlinear layers between input and output. Unfortunately this definition does not apply trivially to a recurrent neural network (RNN) because of its temporal structure. For instance, any RNN when unfolded in time as in Fig. 1 is deep, because a computational path between the input at time $k < t$ to the output at time t crosses several nonlinear layers.

temporal structure 时间结构

A close analysis of the computation carried out by an RNN (see Fig. 2 (a)) at each time step individually, however, shows that certain transitions are not deep, but are only results of a linear projection followed by an element-wise nonlinearity. It is clear that the hidden-to-hidden ($\mathbf{h}_{t-1} \rightarrow \mathbf{h}_t$), hidden-to-output ($\mathbf{h}_t \rightarrow \mathbf{y}_t$) and input-to-hidden ($\mathbf{x}_t \rightarrow \mathbf{h}_t$) functions are all *shallow* in the sense that there exists no intermediate, nonlinear hidden layer.

We can now consider different types of depth of an RNN by considering those transitions separately. We may make the hidden-to-hidden transition deeper by having one or more intermediate nonlinear layers between two consecutive hidden states ($\mathbf{h}_{t-1}$ and $\mathbf{h}_t$). At the same time, the hidden-to-output function can be made deeper, as described previously, by plugging, multiple intermediate nonlinear layers between the hidden state $\mathbf{h}_t$ and the output $\mathbf{y}_t$. Each of these choices has a different implication.

3.2.1 Deep Input-to-Hidden Function

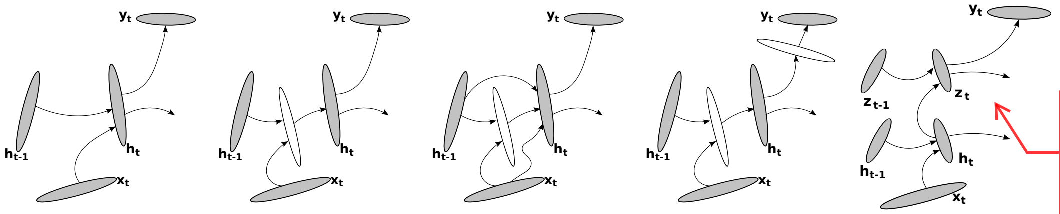
A model can exploit more non-temporal structure from the input by making the input-to-hidden function deep. Previous work has shown that higher-level representations of deep networks tend to better disentangle the underlying factors of variation than the original input (Goodfellow *et al.*, 2009; Glorot *et al.*, 2011b) and flatten the manifolds near which the data concentrate (Bengio *et al.*, 2013). We hypothesize that such higher-level representations should make it easier to learn the temporal structure between successive time steps because the relationship between abstract features can generally be expressed more easily. This has been, for instance, illustrated by the recent work (Mikolov *et al.*, 2013b) showing that word embeddings from neural language models tend to be related to their temporal neighbors by simple algebraic relationships, with the same type of relationship (adding a vector) holding over very different regions of the space, allowing a form of analogical reasoning.

This approach of making the input-to-hidden function deeper is in the line with the standard practice of replacing input with extracted features in order to improve the performance of a machine learning model (see, e.g., Bengio, 2009). Recently, Chen and Deng (2013) reported that a better speech recognition performance could be achieved by employing this strategy, although they did not jointly train the deep input-to-hidden function together with other parameters of an RNN.

3.2.2 Deep Hidden-to-Output Function

A deep hidden-to-output function can be useful to disentangle the factors of variations in the hidden state, making it easier to predict the output. This allows the hidden state of the model to be more compact and may result in the model being able to summarize the history of previous inputs more efficiently. Let us denote an RNN with this deep hidden-to-output function a deep output RNN (DO-RNN).

Instead of having feedforward, intermediate layers between the hidden state and the output, Boulanger-Lewandowski *et al.* (2012) proposed to replace the output layer with a conditional gen-



the transition between the consecutive hidden states is still shallow

(a) RNN (b) DT-RNN (b*) DT(S)-RNN (c) DOT-RNN (d) Stacked RNN

Figure 2: Illustrations of four different recurrent neural networks (RNN). (a) A conventional RNN. (b) Deep Transition (DT) RNN. (b*) DT-RNN with shortcut connections (c) Deep Transition, Deep Output (DOT) RNN. (d) Stacked RNN

erative model such as restricted Boltzmann machines or neural autoregressive distribution estimator (Larochelle and Murray, 2011). In this paper we only consider feedforward intermediate layers.

3.2.3 Deep Hidden-to-Hidden Transition

The third knob we can play with is the depth of the hidden-to-hidden transition. The state transition between the consecutive hidden states effectively adds a new input to the summary of the previous inputs represented by the fixed-length hidden state. Previous work with RNNs has generally limited the architecture to a shallow operation; affine transformation followed by an element-wise nonlinearity. Instead, we argue that this procedure of constructing a new summary, or a hidden state, from the combination of the previous one and the new input should be highly nonlinear. This nonlinear transition could allow, for instance, the hidden state of an RNN to rapidly adapt to quickly changing modes of the input, while still preserving a useful summary of the past. This may be impossible to be modeled by a function from the family of generalized linear models. However, this highly nonlinear transition can be modeled by an MLP with one or more hidden layers which has an universal approximator property (see, e.g., Hornik *et al.*, 1989).

An RNN with this deep transition will be called a deep transition RNN (DT-RNN) throughout remainder of this paper. This model is shown in Fig. 2 (b).

This approach of having a deep transition, however, introduces a potential problem. As the introduction of deep transition increases the number of nonlinear steps the gradient has to traverse when propagated back in time, it might become more difficult to train the model to capture long-term dependencies (Bengio *et al.*, 1994). One possible way to address this difficulty is to introduce shortcut connections (see, e.g., Raiko *et al.*, 2012) in the deep transition, where the added shortcut connections provide shorter paths, skipping the intermediate layers, through which the gradient is propagated back in time. We refer to an RNN having deep transition with shortcut connections by DT(S)-RNN (See Fig. 2 (b*)).

Furthermore, we will call an RNN having both a deep hidden-to-output function and a deep transition a deep output, deep transition RNN (DOT-RNN). See Fig. 2 (c) for the illustration of DOT-RNN. If we consider shortcut connections as well in the hidden to hidden transition, we call the resulting model DOT(S)-RNN.

An approach similar to the deep hidden-to-hidden transition has been proposed recently by Pinheiro and Collobert (2014) in the context of parsing a static scene. They introduced a recurrent convolutional neural network (RCNN) which can be understood as a recurrent network whose the transition between consecutive hidden states (and input to hidden state) is modeled by a convolutional neural network. The RCNN was shown to speed up scene parsing and obtained the state-of-the-art result in Stanford Background and SIFT Flow datasets. Ko and Dieter (2009) proposed deep transitions for Gaussian Process models. Earlier, Valpola and Karhunen (2002) used a deep neural network to model the state transition in a nonlinear, dynamical state-space model.

3.2.4 Stack of Hidden States

An RNN may be extended deeper in yet another way by stacking multiple recurrent hidden layers on top of each other (Schmidhuber, 1992; El Hihi and Bengio, 1996; Jaeger, 2007; Graves, 2013).

We call this model a stacked RNN (sRNN) to distinguish it from the other proposed variants. The goal of a such model is to encourage each recurrent level to operate at a different timescale.

It should be noticed that the DT-RNN and the sRNN extend the conventional, shallow RNN in different aspects. If we look at each recurrent level of the sRNN separately, it is easy to see that the transition between the consecutive hidden states is still shallow. As we have argued above, this limits the family of functions it can represent. For example, if the structure of the data is sufficiently complex, incorporating a new input frame into the summary of what had been seen up to now might be an arbitrarily complex function. In such a case we would like to model this function by something that has universal approximator properties, as an MLP. The model can not rely on the higher layers to do so, because the higher layers do not feed back into the lower layer. On the other hand, the sRNN can deal with multiple time scales in the input sequence, which is not an obvious feature of the DT-RNN. The DT-RNN and the sRNN are, however, orthogonal in the sense that it is possible to have both features of the DT-RNN and the sRNN by stacking multiple levels of DT-RNNs to build a stacked DT-RNN which we do not explore more in this paper.

3.3 Formal descriptions of deep RNNs

Here we give a more formal description on how the deep transition recurrent neural network (DT-RNN) and the deep output RNN (DO-RNN) as well as the stacked RNN are implemented.

3.3.1 Deep Transition RNN

We noticed from the state transition equation of the dynamical system simulated by RNNs in Eq. (1) that there is no restriction on the form of f_h. Hence, we propose here to use a multilayer perceptron to approximate f_h instead.

In this case, we can implement f_h by L intermediate layers such that

$$\mathbf{h}_t = f_h(\mathbf{x}_t, \mathbf{h}_{t-1}) = \phi_h\left(\mathbf{W}_L^\top \phi_{L-1}\left(\mathbf{W}_{L-1}^\top \phi_{L-2}\left(\cdots \phi_1\left(\mathbf{W}_1^\top \mathbf{h}_{t-1} + \mathbf{U}^\top \mathbf{x}_t\right)\right)\right)\right),$$

where ϕ_l and $\mathbf{W}_l$ are the element-wise nonlinear function and the weight matrix for the l-th layer. This RNN with a multilayered transition function is a deep transition RNN (DT-RNN).

An illustration of building an RNN with the deep state transition function is shown in Fig. 2 (b). In the illustration the state transition function is implemented with a neural network with a single intermediate layer.

This formulation allows the RNN to learn a non-trivial, highly nonlinear transition between the consecutive hidden states.

like Network In Network

3.3.2 Deep Output RNN

Similarly, we can use a multilayer perceptron with L intermediate layers to model the output function f_o in Eq. (2) such that

$$\mathbf{y}_t = f_o(\mathbf{h}_t) = \phi_o\left(\mathbf{V}_L^\top \phi_{L-1}\left(\mathbf{V}_{L-1}^\top \phi_{L-2}\left(\cdots \phi_1\left(\mathbf{V}_1^\top \mathbf{h}_t\right)\right)\right)\right),$$

where ϕ_l and $\mathbf{V}_l$ are the element-wise nonlinear function and the weight matrix for the l-th layer. An RNN implementing this kind of multilayered output function is a deep output recurrent neural network (DO-RNN).

Fig. 2 (c) draws a deep output, deep transition RNN (DOT-RNN) implemented using both the deep transition and the deep output with a single intermediate layer each.

3.3.3 Stacked RNN

The stacked RNN (Schmidhuber, 1992; El Hihi and Bengio, 1996) has multiple levels of transition functions defined by

$$\mathbf{h}_t^{(l)} = f_h^{(l)}(\mathbf{h}_t^{(l-1)}, \mathbf{h}_{t-1}^{(l)}) = \phi_h\left(\mathbf{W}_l^\top \mathbf{h}_{t-1}^{(l)} + \mathbf{U}_l^\top \mathbf{h}_t^{(l-1)}\right),$$

where $\mathbf{h}_t^{(l)}$ is the hidden state of the l-th level at time t. When $l = 1$, the state is computed using $\mathbf{x}_t$ instead of $\mathbf{h}_t^{(l-1)}$. The hidden states of all the levels are recursively computed from the bottom level $l = 1$.

Once the top-level hidden state is computed, the output can be obtained using the usual formulation in Eq. (5). Alternatively, one may use all the hidden states to compute the output (Hermans and Schrauwen, 2013). Each hidden state at each level may also be made to depend on the input as well (Graves, 2013). Both of them can be considered approaches using shortcut connections discussed earlier.

The illustration of this stacked RNN is in Fig. 2 (d).

4 Another Perspective: Neural Operators

In this section, we briefly introduce a novel approach with which the already discussed deep transition (DT) and/or deep output (DO) recurrent neural networks (RNN) may be built. We call this approach which is based on building an RNN with a set of predefined neural operators, an operator-based framework.

In the operator-based framework, one first defines a set of operators of which each is implemented by a multilayer perceptron (MLP). For instance, a *plus* operator $\oplus$ may be defined as a function receiving two vectors $\mathbf{x}$ and $\mathbf{h}$ and returning the summary $\mathbf{h}'$ of them:

$$\mathbf{h}' = \mathbf{x} \oplus \mathbf{h},$$

where we may constrain that the dimensionality of $\mathbf{h}$ and $\mathbf{h}'$ are identical. Additionally, we can define another operator $\triangleright$ which *predicts* the most likely output symbol $\mathbf{x}'$ given a summary $\mathbf{h}$, such that

$$\mathbf{x}' = \triangleright \mathbf{h}$$

It is possible to define many other operators, but in this paper, we stick to these two operators which are sufficient to express all the proposed types of RNNs.

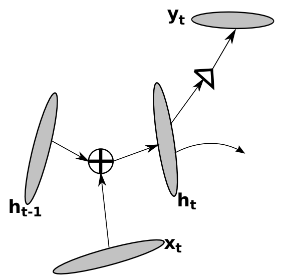


Figure 3: A view of an RNN under the operator-based framework: $\oplus$ and $\triangleright$ are the *plus* and *predict* operators, respectively.

It is clear to see that the plus operator $\oplus$ and the predict operator $\triangleright$ correspond to the transition function and the output function in Eqs. (1)–(2). Thus, at each step, an RNN can be thought as performing the plus operator to update the hidden state given an input ($\mathbf{h}_t = \mathbf{x}_t \oplus \mathbf{h}_{t-1}$) and then the predict operator to compute the output ($\mathbf{y}_t = \triangleright \mathbf{h}_t = \triangleright(\mathbf{x}_t \oplus \mathbf{h}_{t-1})$). See Fig. 3 for the illustration of how an RNN can be understood from the operator-based framework.

Each operator can be parameterized as an MLP with one or more hidden layers, hence a neural operator, since we cannot simply expect the operation will be linear with respect to the input vector(s). By using an MLP to implement the operators, the proposed deep transition, deep output RNN (DOT-RNN) naturally arises.

This framework provides us an insight on how the constructed RNN be regularized. For instance, one may regularize the model such that the plus operator $\oplus$ is commutative. However, in this paper, we do not explore further on this approach.

Note that this is different from (Mikolov *et al.*, 2013a) where the learned embeddings of words happened to be suitable for algebraic operators. The operator-based framework proposed here is rather geared toward *learning* these operators directly.

5 Experiments

We train four types of RNNs described in this paper on a number of benchmark datasets to evaluate their performance. For each benchmark dataset, we try the task of predicting the next symbol.

The task of predicting the next symbol is equivalent to the task of modeling the distribution over a sequence. For each sequence $(\mathbf{x}_1, \dots, \mathbf{x}_T)$, we decompose it into

$$p(\mathbf{x}_1, \dots, \mathbf{x}_T) = p(\mathbf{x}_1) \prod_{t=2}^{T} p(\mathbf{x}_t \mid \mathbf{x}_1, \dots, \mathbf{x}_{t-1}),$$

and each term on the right-hand side will be replaced with a single timestep of an RNN. In this setting, the RNN predicts the probability of the next symbol $\mathbf{x}_t$ in the sequence given the all previous symbols $\mathbf{x}_1, \dots \mathbf{x}_{t-1}$. Then, we train the RNN by maximizing the log-likelihood.

We try this task of modeling the joint distribution on three different tasks; polyphonic music prediction, character-level and word-level language modeling.

We test the RNNs on the task of polyphonic music prediction using three datasets which are Nottingham, JSB Chorales and MuseData (Boulanger-Lewandowski *et al.*, 2012). On the task of character-level and word-level language modeling, we use Penn Treebank Corpus (Marcus *et al.*, 1993).

5.1 Model Descriptions

We compare the conventional recurrent neural network (RNN), deep transition RNN with shortcut connections in the transition MLP (DT(S)-RNN), deep output/transition RNN with shortcut connections in the hidden to hidden transition MLP (DOT(S)-RNN) and stacked RNN (sRNN). See Fig. 2 (a)–(d) for the illustrations of these models.

			RNN	DT(S)-RNN	DOT(S)-RNN	sRNN 2 layers
Music	Notthingam	# units	600	400,400	400,400,400	400
		# parameters	465K	585K	745K	550K
	JSB Chorales	# units	200	400,400	400,400,400	400
		# parameters	75K	585K	745K	550K
	MuseData	# units	600	400,400	400,400,400	600
		# parameters	465K	585K	745K	1185K
Language	Char-level	# units	600	400,400	400,400,600	400
		# parameters	420K	540K	790K	520K
	Word-level	# units	200	200,200	200,200,200	400
		# parameters	4.04M	6.12M	6.16M	8.48M

Table 1: The sizes of the trained models. We provide the number of hidden units as well as the total number of parameters. For DT(S)-RNN, the two numbers provided for the number of units mean the size of the hidden state and that of the intermediate layer, respectively. For DOT(S)-RNN, the three numbers are the size of the hidden state, that of the intermediate layer between the consecutive hidden states and that of the intermediate layer between the hidden state and the output layer. For sRNN, the number corresponds to the size of the hidden state at each level

The size of each model is chosen from a limited set $\{100, 200, 400, 600, 800\}$ to minimize the validation error for each polyphonic music task (See Table. 1 for the final models). In the case of language modeling tasks, we chose the size of the models from $\{200, 400\}$ and $\{400, 600\}$ for word-level and character-level tasks, respectively. In all cases, we use a logistic sigmoid function as an element-wise nonlinearity of each hidden unit. Only for the character-level language modeling we used rectified linear units (Glorot *et al.*, 2011a) for the intermediate layers of the output function, which gave lower validation error.

5.2 Training

We use stochastic gradient descent (SGD) and employ the strategy of clipping the gradient proposed by Pascanu *et al.* (2013a). Training stops when the validation cost stops decreasing.

Polyphonic Music Prediction: For Nottingham and MuseData datasets we compute each gradient step on subsequences of at most 200 steps, while we use subsequences of 50 steps for JSB Chorales.

We do not reset the hidden state for each subsequence, unless the subsequence belongs to a different song than the previous subsequence.

The cutoff threshold for the gradients is set to 1. The hyperparameter for the learning rate schedule[1] is tuned manually for each dataset. We set the hyperparameter β to 2330 for Nottingham, 1475 for MuseData and 100 for JSB Chroales. They correspond to two epochs, a single epoch and a third of an epoch, respectively.

The weights of the connections between any pair of hidden layers are sparse, having only 20 non-zero incoming connections per unit (see, e.g., Sutskever *et al.*, 2013). Each weight matrix is rescaled to have a unit largest singular value (Pascanu *et al.*, 2013a). The weights of the connections between the input layer and the hidden state as well as between the hidden state and the output layer are initialized randomly from the white Gaussian distribution with its standard deviation fixed to 0.1 and 0.01, respectively. In the case of deep output functions (DOT(S)-RNN), the weights of the connections between the hidden state and the intermediate layer are sampled initially from the white Gaussian distribution of standard deviation 0.01. In all cases, the biases are initialized to 0.

To regularize the models, we add white Gaussian noise of standard deviation 0.075 to each weight parameter every time the gradient is computed (Graves, 2011).

Language Modeling: We used the same strategy for initializing the parameters in the case of language modeling. For character-level modeling, the standard deviations of the white Gaussian distributions for the input-to-hidden weights and the hidden-to-output weights, we used 0.01 and 0.001, respectively, while those hyperparameters were both 0.1 for word-level modeling. In the case of DOT(S)-RNN, we sample the weights of between the hidden state and the rectifier intermediate layer of the output function from the white Gaussian distribution of standard deviation 0.01. When using rectifier units (character-based language modeling) we fix the biases to 0.1.

In language modeling, the learning rate starts from an initial value and is halved each time the validation cost does not decrease significantly (Mikolov *et al.*, 2010). We do not use any regularization for the character-level modeling, but for the word-level modeling we use the same strategy of adding weight noise as we do with the polyphonic music prediction.

For all the tasks (polyphonic music prediction, character-level and word-level language modeling), the stacked RNN and the DOT(S)-RNN were initialized with the weights of the conventional RNN and the DT(S)-RNN, which is similar to layer-wise pretraining of a feedforward neural network (see, e.g., Hinton and Salakhutdinov, 2006). We use a ten times smaller learning rate for each parameter that was pretrained as either RNN or DT(S)-RNN.

	RNN	DT(S)-RNN	DOT(S)-RNN	sRNN	DOT(S)-RNN*
Notthingam	3.225	3.206	3.215	3.258	2.95
JSB Chorales	8.338	8.278	8.437	8.367	7.92
MuseData	6.990	6.988	6.973	6.954	6.59

Table 2: The performances of the four types of RNNs on the polyphonic music prediction. The numbers represent negative log-probabilities on test sequences. (*) We obtained these results using DOT(S)-RNN with L_p units in the deep transition, maxout units in the deep output function and dropout (Gulcehre *et al.*, 2013).

5.3 Result and Analysis

5.3.1 Polyphonic Music Prediction

The log-probabilities on the test set of each data are presented in the first four columns of Tab. 2. We were able to observe that in all cases one of the proposed deep RNNs outperformed the conventional, shallow RNN. Though, the suitability of each deep RNN depended on the data it was trained on. The best results obtained by the DT(S)-RNNs on Notthingam and JSB Chorales are close to, but

[1] We use at each update τ, the following learning rate $\eta_\tau = \frac{1}{1+\frac{\max(0,\tau-\tau_0)}{\beta}}$, where τ_0 and β indicate respectively when the learning rate starts decreasing and how quickly the learning rate decreases. In the experiment, we set τ_0 to coincide with the time when the validation error starts increasing for the first time.

worse than the result obtained by RNNs trained with the technique of fast dropout (FD) which are 3.09 and 8.01, respectively (Bayer *et al.*, 2013).

In order to quickly investigate whether the proposed deeper variants of RNNs may also benefit from the recent advances in feedforward neural networks, such as the use of non-saturating activation functions[2] and the method of dropout. We have built another set of DOT(S)-RNNs that have the recently proposed L_p units (Gulcehre *et al.*, 2013) in deep transition and maxout units (Goodfellow *et al.*, 2013) in deep output function. Furthermore, we used the method of dropout (Hinton *et al.*, 2012b) instead of weight noise during training. Similarly to the previously trained models, we searched for the size of the models as well as other learning hyperparameters that minimize the validation performance. We, however, did not pretrain these models.

The results obtained by the DOT(S)-RNNs having L_p and maxout units trained with dropout are shown in the last column of Tab. 2. On every music dataset the performance by this model is significantly better than those achieved by all the other models as well as the best results reported with recurrent neural networks in (Bayer *et al.*, 2013). This suggests us that the proposed variants of deep RNNs also benefit from having non-saturating activations and using dropout, just like feedforward neural networks. We reported these results and more details on the experiment in (Gulcehre *et al.*, 2013).

We, however, acknowledge that the model-free state-of-the-art results for the both datasets were obtained using an RNN combined with a conditional generative model, such as restricted Boltzmann machines or neural autoregressive distribution estimator (Larochelle and Murray, 2011), in the output (Boulanger-Lewandowski *et al.*, 2012).

	RNN	DT(S)-RNN	DOT(S)-RNN	sRNN	$*$	$\star$
Character-Level	1.414	1.409	**1.386**	1.412	1.41[1]	1.24[3]
Word-Level	117.7	112.0	**107.5**	110.0	123[2]	117[3]

Table 3: The performances of the four types of RNNs on the tasks of language modeling. The numbers represent bit-per-character and perplexity computed on test sequence, respectively, for the character-level and word-level modeling tasks. $*$ The previous/current state-of-the-art results obtained with shallow RNNs. $\star$ The previous/current state-of-the-art results obtained with RNNs having long-short term memory units.

5.3.2 Language Modeling

On Tab. 3, we can see the perplexities on the test set achieved by the all four models. We can clearly see that the deep RNNs (DT(S)-RNN, DOT(S)-RNN and sRNN) outperform the conventional, shallow RNN significantly. On these tasks DOT(S)-RNN outperformed all the other models, which suggests that it is important to have highly nonlinear mapping from the hidden state to the output in the case of language modeling.

The results by both the DOT(S)-RNN and the sRNN for word-level modeling surpassed the previous best performance achieved by an RNN with 1000 long short-term memory (LSTM) units (Graves, 2013) as well as that by a shallow RNN with a larger hidden state (Mikolov *et al.*, 2011), even when both of them used dynamic evaluation[3]. The results we report here are without dynamic evaluation.

For character-level modeling the state-of-the-art results were obtained using an optimization method Hessian-free with a specific type of RNN architecture called mRNN (Mikolov *et al.*, 2012a) or a regularization technique called adaptive weight noise (Graves, 2013). Our result, however, is better than the performance achieved by conventional, shallow RNNs without any of those advanced

[2] Note that it is not trivial to use non-saturating activation functions in conventional RNNs, as this may cause the explosion of the activations of hidden states. However, it is perfectly safe to use non-saturating activation functions at the intermediate layers of a deep RNN with deep transition.

[1] Reported by Mikolov *et al.* (2012a) using mRNN with Hessian-free optimization technique.

[2] Reported by Mikolov *et al.* (2011) using the dynamic evaluation.

[3] Reported by Graves (2013) using the dynamic evaluation and weight noise.

[3] Dynamic evaluation refers to an approach where the parameters of a model are updated as the validation/test data is predicted.

regularization methods (Mikolov *et al.*, 2012b), where they reported the best performance of 1.41 using an RNN trained with the Hessian-free learning algorithm (Martens and Sutskever, 2011).

6 Discussion

In this paper, we have explored a novel approach to building a deep recurrent neural network (RNN). We considered the structure of an RNN at each timestep, which revealed that the relationship between the consecutive hidden states and that between the hidden state and output are *shallow*. Based on this observation, we proposed two alternative designs of *deep* RNN that make those *shallow* relationships be modeled by deep neural networks. Furthermore, we proposed to make use of shortcut connections in these deep RNNs to alleviate a problem of difficult learning potentially introduced by the increasing depth.

We empirically evaluated the proposed designs against the conventional RNN which has only a single hidden layer and against another approach of building a deep RNN (stacked RNN, Graves, 2013), on the task of polyphonic music prediction and language modeling.

The experiments revealed that the RNN with the proposed deep transition and deep output (DOT(S)-RNN) outperformed both the conventional RNN and the stacked RNN on the task of language modeling, achieving the state-of-the-art result on the task of word-level language modeling. For polyphonic music prediction, a different deeper variant of an RNN achieved the best performance for each dataset. Importantly, however, in all the cases, the conventional, shallow RNN was not able to outperform the deeper variants. These results strongly support our claim that an RNN benefits from having a deeper architecture, just like feedforward neural networks.

The observation that there is no clear winner in the task of polyphonic music prediction suggests us that each of the proposed deep RNNs has a distinct characteristic that makes it more, or less, suitable for certain types of datasets. We suspect that in the future it will be possible to design and train yet another deeper variant of an RNN that combines the proposed models together to be more robust to the characteristics of datasets. For instance, a stacked DT(S)-RNN may be constructed by combining the DT(S)-RNN and the sRNN.

In a quick additional experiment where we have trained DOT(S)-RNN constructed using non-saturating nonlinear activation functions and trained with the method of dropout, we were able to improve the performance of the deep recurrent neural networks on the polyphonic music prediction tasks significantly. This suggests us that it is important to investigate the possibility of applying recent advances in feedforward neural networks, such as novel, non-saturating activation functions and the method of dropout, to recurrent neural networks as well. However, we leave this as future research.

One practical issue we ran into during the experiments was the difficulty of training deep RNNs. We were able to train the conventional RNN as well as the DT(S)-RNN easily, but it was not trivial to train the DOT(S)-RNN and the stacked RNN. In this paper, we proposed to use shortcut connections as well as to pretrain them either with the conventional RNN or with the DT(S)-RNN. We, however, believe that learning may become even more problematic as the size and the depth of a model increase. In the future, it will be important to investigate the root causes of this difficulty and to explore potential solutions. We find some of the recently introduced approaches, such as advanced regularization methods (Pascanu *et al.*, 2013a) and advanced optimization algorithms (see, e.g., Pascanu and Bengio, 2013; Martens, 2010), to be promising candidates.

Acknowledgments

We would like to thank the developers of Theano (Bergstra *et al.*, 2010; Bastien *et al.*, 2012). We also thank Justin Bayer for his insightful comments on the paper. We would like to thank NSERC, Compute Canada, and Calcul Québec for providing computational resources. Razvan Pascanu is supported by a DeepMind Fellowship. Kyunghyun Cho is supported by FICS (Finnish Doctoral Programme in Computational Sciences) and “the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170)”.

References

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. (2012). Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop.

Bayer, J., Osendorfer, C., Korhammer, D., Chen, N., Urban, S., and van der Smagt, P. (2013). On fast dropout and its applicability to recurrent networks. *arXiv:*`1311.0701 [cs.NE]`.

Bengio, Y. (2009). Learning deep architectures for AI. *Found. Trends Mach. Learn.*, **2**(1), 1–127.

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

Bengio, Y., Mesnil, G., Dauphin, Y., and Rifai, S. (2013). Better mixing via deep representations. In *ICML'13*.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *ICML'2012*.

Chen, J. and Deng, L. (2013). A new method for learning deep recurrent neural networks. *arXiv:*`1311.6091 [cs.LG]`.

Delalleau, O. and Bengio, Y. (2011). Shallow vs. deep sum-product networks. In *NIPS*.

El Hihi, S. and Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. In *NIPS 8*. MIT Press.

Glorot, X., Bordes, A., and Bengio, Y. (2011a). Deep sparse rectifier neural networks. In *AISTATS*.

Glorot, X., Bordes, A., and Bengio, Y. (2011b). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *ICML'2011*.

Goodfellow, I., Le, Q., Saxe, A., and Ng, A. (2009). Measuring invariances in deep networks. In *NIPS'09*, pages 646–654.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout networks. In *ICML'2013*.

Graves, A. (2011). Practical variational inference for neural networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2348–2356.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv:*`1308.0850 [cs.NE]`.

Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for improved unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Graves, A., Mohamed, A., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *ICASSP*.

Gulcehre, C., Cho, K., Pascanu, R., and Bengio, Y. (2013). Learned-norm pooling for deep feedforward and recurrent neural networks. *arXiv:*`1311.1780 [cs.NE]`.

Hermans, M. and Schrauwen, B. (2013). Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems 26*, pages 190–198.

Hinton, G., Deng, L., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012a). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, **29**(6), 82–97.

Hinton, G. E. and Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science*, **313**(5786), 504–507.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012b). Improving neural networks by preventing co-adaptation of feature detectors. Technical report, arXiv:1207.0580.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, 359–366.

Jaeger, H. (2007). Discovering multiscale dynamical features with hierarchical echo state networks. Technical report, Jacobs University.

Ko, J. and Dieter, F. (2009). Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*.

Larochelle, H. and Murray, I. (2011). The Neural Autoregressive Distribution Estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS'2011)*, volume 15 of JMLR: W&CP.

Le Roux, N. and Bengio, Y. (2010). Deep belief networks are compact universal approximators. *Neural Computation*, **22**(8), 2192–2207.

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

Martens, J. (2010). Deep learning via Hessian-free optimization. In L. Bottou and M. Littman, editors, *Proceedings of the Twenty-seventh International Conference on Machine Learning (ICML-10)*, pages 735–742. ACM.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with Hessian-free optimization. In *Proc. ICML'2011*. ACM.

Mikolov, T. (2012). *Statistical Language Models based on Neural Networks*. Ph.D. thesis, Brno University of Technology.

Mikolov, T., Karafiát, M., Burget, L., Cernocky, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, volume 2010, pages 1045–1048. International Speech Communication Association.

Mikolov, T., Kombrink, S., Burget, L., Cernocky, J., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *Proc. 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP 2011)*.

Mikolov, T., Sutskever, I., Deoras, A., Le, H., Kombrink, S., and Cernocky, J. (2012a). Subword language modeling with neural networks. *unpublished*.

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

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013b). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations: Workshops Track*.

Pascanu, R. and Bengio, Y. (2013). Revisiting natural gradient for deep networks. Technical report, arXiv:1301.3584.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013a). On the difficulty of training recurrent neural networks. In *ICML'2013*.

Pascanu, R., Montufar, G., and Bengio, Y. (2013b). On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv:*`1312.6098[cs.LG]`.

Pinheiro, P. and Collobert, R. (2014). Recurrent convolutional neural networks for scene labeling. In *Proceedings of The 31st International Conference on Machine Learning*, pages 82–90.

Raiko, T., Valpola, H., and LeCun, Y. (2012). Deep learning made easier by linear transformations in perceptrons. In *Proceedings of the Fifteenth Internation Conference on Artificial Intelligence and Statistics (AISTATS 2012)*, volume 22 of *JMLR Workshop and Conference Proceedings*, pages 924–932. JMLR W&CP.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, **323**, 533–536.

Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, (4), 234–242.

Sutskever, I., Martens, J., and Hinton, G. (2011). Generating text with recurrent neural networks. In L. Getoor and T. Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, pages 1017–1024, New York, NY, USA. ACM.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *ICML*.

Valpola, H. and Karhunen, J. (2002). An unsupervised ensemble learning method for nonlinear dynamic state-space models. *Neural Comput.*, **14**(11), 2647–2692.