Neural Networks 23 (2010) 239–243

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Temporal-Kernel Recurrent Neural Networks

Ilya Sutskever *, Geoffrey Hinton

Department of Computer Science, University of Toronto, Canada

ARTICLE INFO

Article history:
Received 30 April 2008
Revised and accepted 26 October 2009

Keywords:
Recurrent Neural Networks
Fixed points
Long-term dependencies
Backpropagation through time
Supervised learning

ABSTRACT

A Recurrent Neural Network (RNN) is a powerful connectionist model that can be applied to many challenging sequential problems, including problems that naturally arise in language and speech. However, RNNs are extremely hard to train on problems that have long-term dependencies, where it is necessary to remember events for many timesteps before using them to make a prediction.

In this paper we consider the problem of training RNNs to predict sequences that exhibit significant long-term dependencies, focusing on a serial recall task where the RNN needs to remember a sequence of characters for a large number of steps before reconstructing it. We introduce the Temporal-Kernel Recurrent Neural Network (TKRNN), which is a variant of the RNN that can cope with long-term dependencies much more easily than a standard RNN, and show that the TKRNN develops short-term memory that successfully solves the serial recall task by representing the input string with a stable state of its hidden units.

1. Introduction

Recurrent Neural Networks (RNNs) are connectionist models that operate in discrete time using feedback connections. An RNN has a set of units, each taking a real value in each timestep, and a set of weighted connections between its units. The input units are set by the environment and the output units are computed using the connection weights and the hidden units.

RNNs have nonlinear dynamics, allowing them to behave in a highly complex manner. In principle, the states of the hidden units can store information through time in the form of a distributed representation and this distributed representation can be used many timesteps later to predict subsequent input vectors.

RNNs are appealing because of their range of potential applications: they can be applied to almost any problem with sequential structure, including the problems that arise naturally in speech, control, and natural language processing. Since RNNs can represent highly complex functions of sequences, these problems are likely to be solvable with some RNN, so a learning algorithm that can find this RNN would be very useful in practice.

Unfortunately, RNNs have proved to be difficult to learn with gradient descent, especially when the sought for RNN must use its units to store events for more than a few timesteps. Whenever events in the far past are relevant for predicting the current timestep, the problem is said to exhibit long-term dependencies. It is known (Bengio, Simard, & Frasconi, 1994; Hochreiter, 1991) that gradient descent has great difficulty in learning weights that make use of long-term dependencies, so the resulting RNNs are typically no more useful than a simple moving window.

In this paper, we address the problem of learning RNNs that successfully predict sequences that exhibit long-term dependencies. In particular, we focus on a serial recall task in which an arbitrary sequence of characters must be stored for a variable length of time until a cue is presented. After the cue is presented, the RNN must reproduce the stored sequence. The variable time delay makes it very hard to solve this problem using delay lines, so the RNN must learn to convert the arbitrary sequence to a stable distributed pattern of activity and then convert this stable pattern back into the appropriate sequence when the recall cue arrives.

Our main contribution is a new family of RNNs, the Temporal-Kernel Recurrent Neural Network (TKRNN), in which every unit is an efficient leaky integrator, which makes it easier to "notice" long-term dependencies: we demonstrate that the TKRNN can learn to use its units to store 35 bits of information for at least 50 timesteps in an immediate serial recall task (e.g., Botvinick & Plaut, 2006). The TKRNN learns to represent its input with a stable state of its hidden units, which is relevant to a line of research that uses the fixed points of biologically-plausible neural networks to represent useful information, such as shape (Amit, 1995) or eye position (Seung, 1996; Camperi & Wang, 1998), for extended periods of time. Our experiments show that the TKRNN's architecture is suitable for such problems, and that its performance is comparable to that of the Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) when applied to the same tasks.

* Corresponding author.
E-mail addresses: ilya@cs.utoronto.ca (I. Sutskever), hinton@cs.utoronto.ca (G. Hinton).

0893-6080/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2009.10.009

2. Standard Recurrent Neural Networks: Definitions

In this section we formally define the standard RNN (Werbos, 1990; Rumelhart, Hinton, & Williams, 1986). The RNN is a neural network that operates in time. At each time t, the value of the RNN's input units is given by $\mathbf{x}_t$, and the RNN computes the values of its hidden units ($\mathbf{y}_t$) and output units ($\mathbf{z}_t$) by the equations

$$\mathbf{y}_t = f\left(W_{\mathbf{y}\to\mathbf{y}}\mathbf{y}_{t-1} + W_{\mathbf{x}\to\mathbf{y}}\mathbf{x}_t\right) \tag{1}$$

$$\mathbf{z}_t = g\left(W_{\mathbf{z}\to\mathbf{y}}\mathbf{y}_t + W_{\mathbf{x}\to\mathbf{z}}\mathbf{x}_t\right) \tag{2}$$

where $W_{\mathbf{z}\to\mathbf{y}}$ is a matrix of size $n_z \times n_y$ of the weights of the connections between the hidden units $\mathbf{y}$ and the output units $\mathbf{z}$. A common choice for the functions $f : \mathbb{R}^n \to \mathbb{R}^n$ and $g : \mathbb{R}^n \to \mathbb{R}^n$ is the sigmoid and the softmax functions, which are

$$f(\mathbf{x})^{(i)} = \frac{1}{1+\exp(-\mathbf{x}^{(i)})} \tag{3}$$

$$g(\mathbf{x})^{(i)} = \frac{\exp(\mathbf{x}^{(i)})}{\sum_{j=1}^{n}\exp(\mathbf{x}^{(j)})} \tag{4}$$

where in these equations $\mathbf{x}$ is a generic n-dimensional vector and $\mathbf{x}^{(i)}$ is its i'th coordinate. Other definitions of f and g are also possible. Thus, given a setting of the RNN's parameters, these equations completely determine the values of the RNN's hidden ($\{\mathbf{y}_t\}_t$) and output ($\{\mathbf{z}_t\}_t$) units for any sequence of input vectors $\{\mathbf{x}_t\}_t$.

An RNN can be trained to approximate a highly complex function on sequences using a collection of training input sequences $\{\mathbf{x}_t\}_t$ and their corresponding desired outputs $\{\mathbf{v}_t\}_t$. The problem of learning the RNN's connection weights is formulated as a problem of minimizing a cost function C that measures the RNN's deviation from perfect behavior:

$$C = \sum_{t=1}^{T} C_t = \sum_{t=1}^{T} c(\mathbf{z}_t, \mathbf{v}_t) \tag{5}$$

where T is the length of the sequence, and $c(\mathbf{z}_t, \mathbf{v}_t)$ is a measure of distance between the desired output and the actual output (we use the cross-entropy $c(\mathbf{x}, \mathbf{z}) = -\sum_i \mathbf{z}^{(i)} \log \mathbf{x}^{(i)}$).

3. Temporal-Kernel Recurrent Neural Networks

The backpropagation through time algorithm (BPTT) (Werbos, 1990; Rumelhart et al., 1986) can efficiently compute the gradient of the RNNs cost function (Eq. (5)), so it may seem that RNNs should be easy to learn. However, if the RNN must learn to remember events in the far past in order to make accurate predictions about the present where the relevant events are always separated by many timesteps, the RNN learned by BPTT will fail to use its hidden units to store the important relevant information from the past.

A theoretical analysis (Bengio et al., 1994; Hochreiter, 1991) shows that learning is hard because the past is separated from the present with a large number of nonlinearities, causing the gradient to get "diluted" and uninformative as it flows backwards through time. If we allow the gradient to skip timesteps as it flows backwards, it can influence the past more directly and be less diluted, which is easily achieved by adding direct connections between units that are separated in time: if k is not too large ($<k_0$), we connect $\mathbf{y}_{t-k}$ and $\mathbf{y}_t$ with connections whose weights are independent of t. By adding these connections, we essentially obtain the NARX RNN (Lin, Horne, Tino, & Giles, 2000).

These additional connections allow the NARX RNN to learn long-term regularities that are k_0 times more separated in time than the standard RNN, which can be substantial when k_0 is large. However, NARX RNNs have two drawbacks. First, NARX RNNs are k_0 times slower than RNNs (per iteration), and second, NARX RNNs have k_0 times more parameters than a standard RNN with the same number of units. This is particularly costly because k_0 often needs to be large, so the NARX RNN mitigates the problem of learning at the expense of being slower and larger.

Our contribution is a new family of RNNs that has many of the advantages of the NARX RNN without its disadvantages. We introduce the Temporal-Kernel Recurrent Neural Network (TKRNN), which is an RNN with direct connections between units in all timesteps (from $\mathbf{y}_t$ to $\mathbf{y}_{t'}$ for all $t' < t$), which is as efficient as the standard RNN and has almost the same number of parameters. We also introduce the TKRNN^{+n}, which is an RNN whose weights are the sum of the weights of n TKRNNs; the TKRNN^{+n} is n times slower than the TKRNN per forward/backward pass, but it finds considerably better solutions.

The main idea of the TKRNN is to make each of its units act as a leaky integrator while keeping the forward and the backward pass efficient. The equation that governs the TKRNN's hidden units is

$$\mathbf{y}_t{}^{(i)} = f\left(\sum_{k=1}^{t}\left(\sum_{j=1}^{n_y}(\lambda^{(j)})^{k-1}W_{\mathbf{y}\to\mathbf{y}}{}^{(j,i)}\mathbf{y}_{t-k}^{(j)} + \sum_{m=1}^{n_x}(\lambda^{(m)})^{k-1}W_{\mathbf{x}\to\mathbf{y}}{}^{(m,i)}\mathbf{x}_{t-k}^{(m)}\right)\right) \tag{6}$$

where $0 < \lambda^{(j)} < 1$ is an additional parameter for each unit j that determines the extent to which the unit can directly influence units in future timesteps, or, equivalently, the extent to which units are influenced by unit j's activities in previous timesteps.[1] An analogous equation defines the output units.

The TKRNN has connections between units in all timesteps which make the gradient flow through less nonlinearities, but rather than being arbitrary, the connections' weights are factored in space and time: the weight of the connection between unit $\mathbf{y}_t^{(j)}$ and $\mathbf{y}_{t-k}^{(i)}$ is $W_{\mathbf{y}\to\mathbf{y}}{}^{(j,i)}(\lambda^{(j)})^{k-1}$, which ensures that the number of parameters is small and that the forward and the backward passes can be performed efficiently. Eq. (6) implies that the units' input is an average of the units' past activities weighted by the exponential kernel.

Previous work in which units were similarly connected through time with a kernel include (Hinton & Brown, 2000; Natarajan, Huys, Dayan, & Zemel, 2008), although they were used in a different context. Notably, the Gamma model (De Vries & Principe, 1992) is a closely related RNN architecture that uses a related family of kernels.

The TKRNN's definition causes the forward and backward passes to be as efficient as those of a standard RNN: notice that Eq. (6) can be rewritten as

$$\mathbf{y}_t{}^{(i)} = f\left(\sum_{j=1}^{n_y}W_{\mathbf{y}\to\mathbf{y}}{}^{(j,i)}\sum_{k=1}^{t}(\lambda^{(j)})^{k-1}\mathbf{y}_{t-k}^{(j)} + \sum_{m=1}^{n_x}W_{\mathbf{x}\to\mathbf{y}}{}^{(m,i)}\sum_{k=1}^{t}(\lambda^{(m)})^{k-1}\mathbf{x}_{t-k}^{(m)}\right) \tag{7}$$

so if we define

$$\mathbf{S}_t^{\mathbf{y}(j)} = \sum_{k=1}^{t}(\lambda^{(j)})^{k-1}\mathbf{y}_{t-k}^{(j)} \tag{8}$$

$$\mathbf{S}_t^{\mathbf{x}(m)} = \sum_{k=1}^{t}(\lambda^{(m)})^{k-1}\mathbf{x}_{t-k}^{(m)} \tag{9}$$

[1] We slightly abuse notation and treat the variables $\lambda^{(j)}$ and $\lambda^{(m)}$ as distinct.