机器读心术之神经网络与深度学习 第12周

法律声明

关注炼数成金企业微信

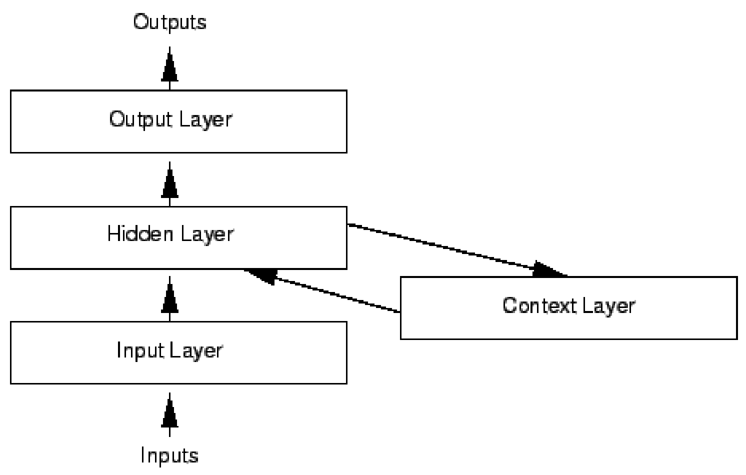
- 提供全面的数据价值资讯，涵盖商业智能与数据分析、大数据、企业信息化、数字化技术等，各种高性价比课程信息，赶紧掏出您的手机关注吧！

主要内容

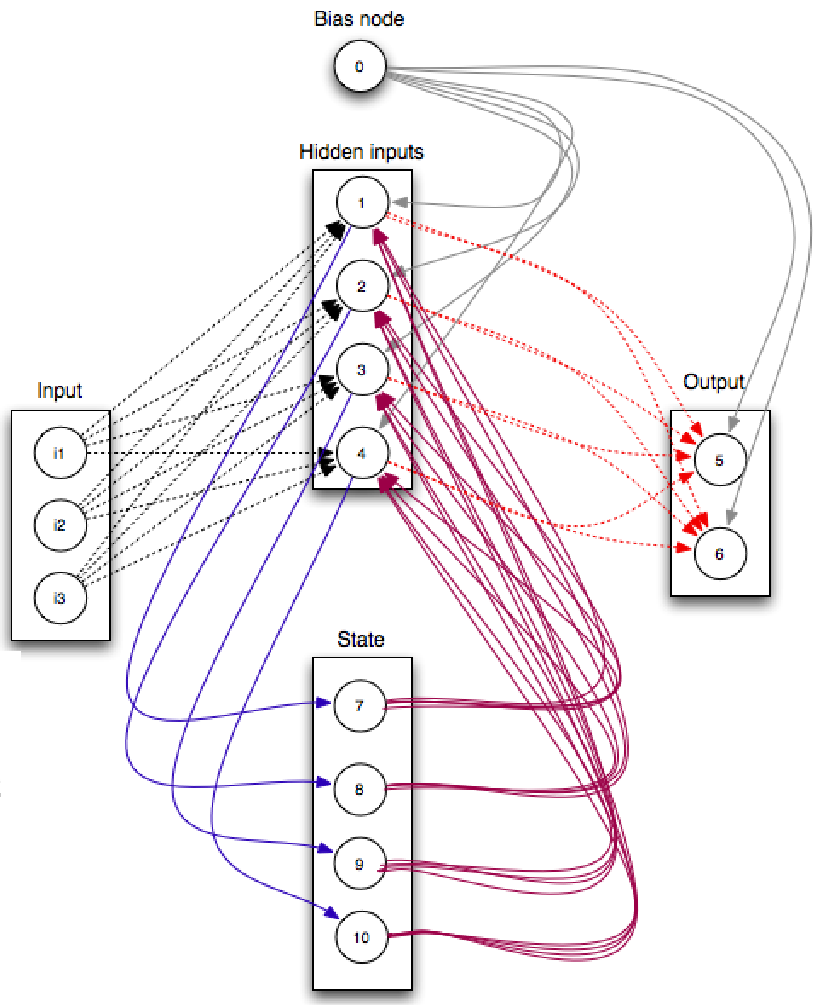
RNN

- RNN= Recurrent Neural Network
- Jordan，Pineda．Williams，Elman等于上世纪80年代末提出的一种神经网络结构模型。这种网络的本质特征是在处理单元之间既有内部的反馈连接又有前馈连接。从系统观点看，它是一个反馈动力系统，在计算过程中体现过程动态特性，比前馈神经网络具有更强的动态行为和计算能力。循环神经网络现已成为国际上神经网络专家研究的重要对象之一。
- 这种网络的内部状态可以展示动态时序行为。不同于前馈神经网络的是，RNN可以利用它内部的记忆来处理任意时序的输入序列，这让它可以更容易处理如不分段的手写识别、语音识别等。

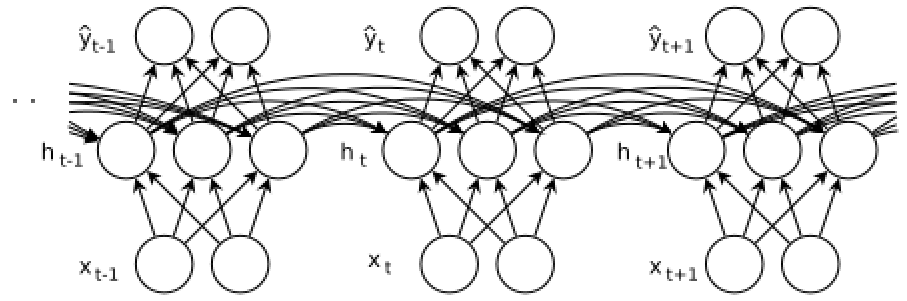
用图表述RNN



Outputs
Output Layer
Hidden Layer
Context Layer
Input Layer
Inputs



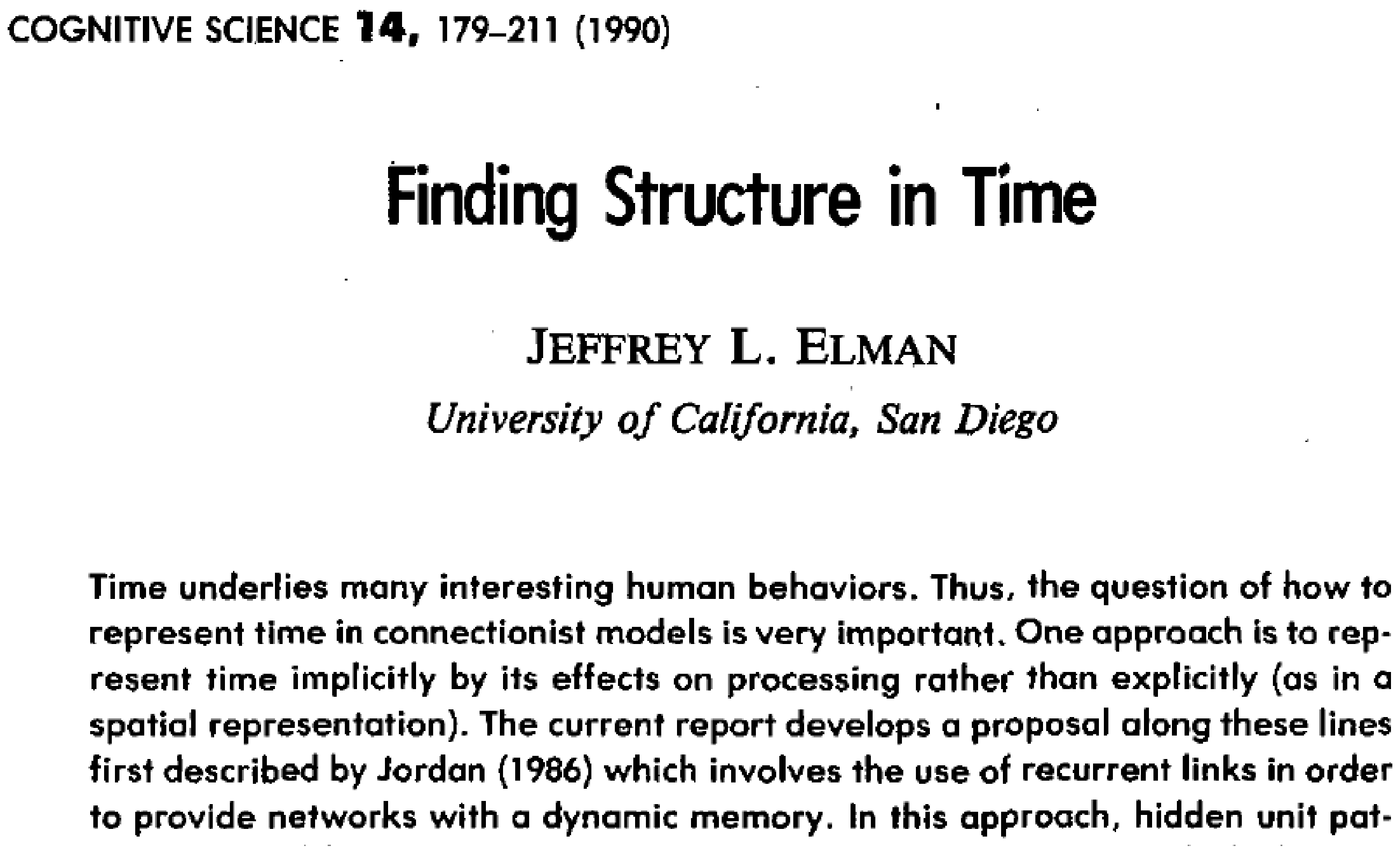
Bias node
0
Hidden inputs
1
2
3
4
Input
i1
i2
i3
Output
5
6
State
7
8
9
10



ŷt-1
ŷt
ŷt+1
ht-1
ht
ht+1
xt-1
xt
xt+1

Elman网络

- 参考论文



COGNITIVE SCIENCE **14,** 179–211 (1990)

Finding Structure in Time

JEFFREY L. ELMAN

University of California, San Diego

Time underlies many interesting human behaviors. Thus, the question of how to represent time in connectionist models is very important. One approach is to represent time implicitly by its effects on processing rather than explicitly (as in a spatial representation). The current report develops a proposal along these lines first described by Jordan (1986) which involves the use of recurrent links in order to provide networks with a dynamic memory. In this approach, hidden unit pat-

Elman网络模型

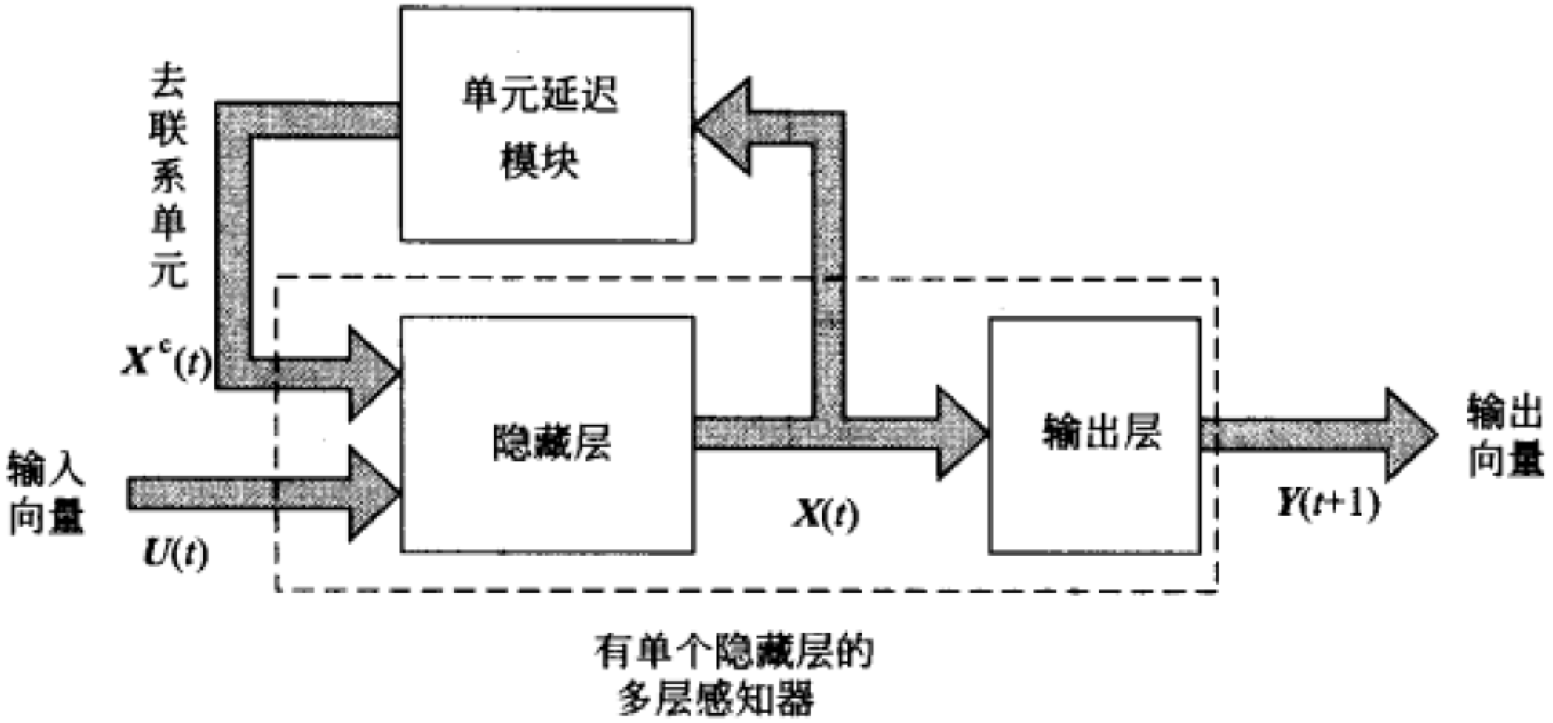
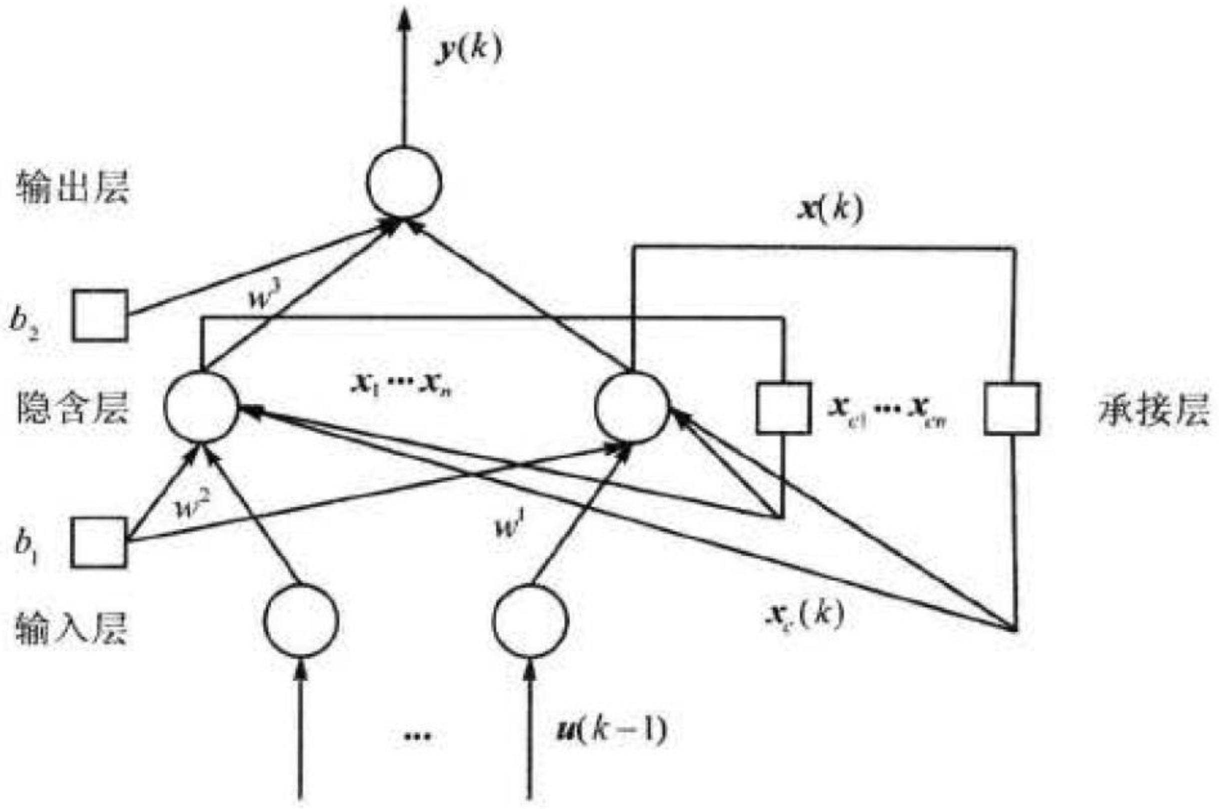


图 6.16　Elman 网络模型

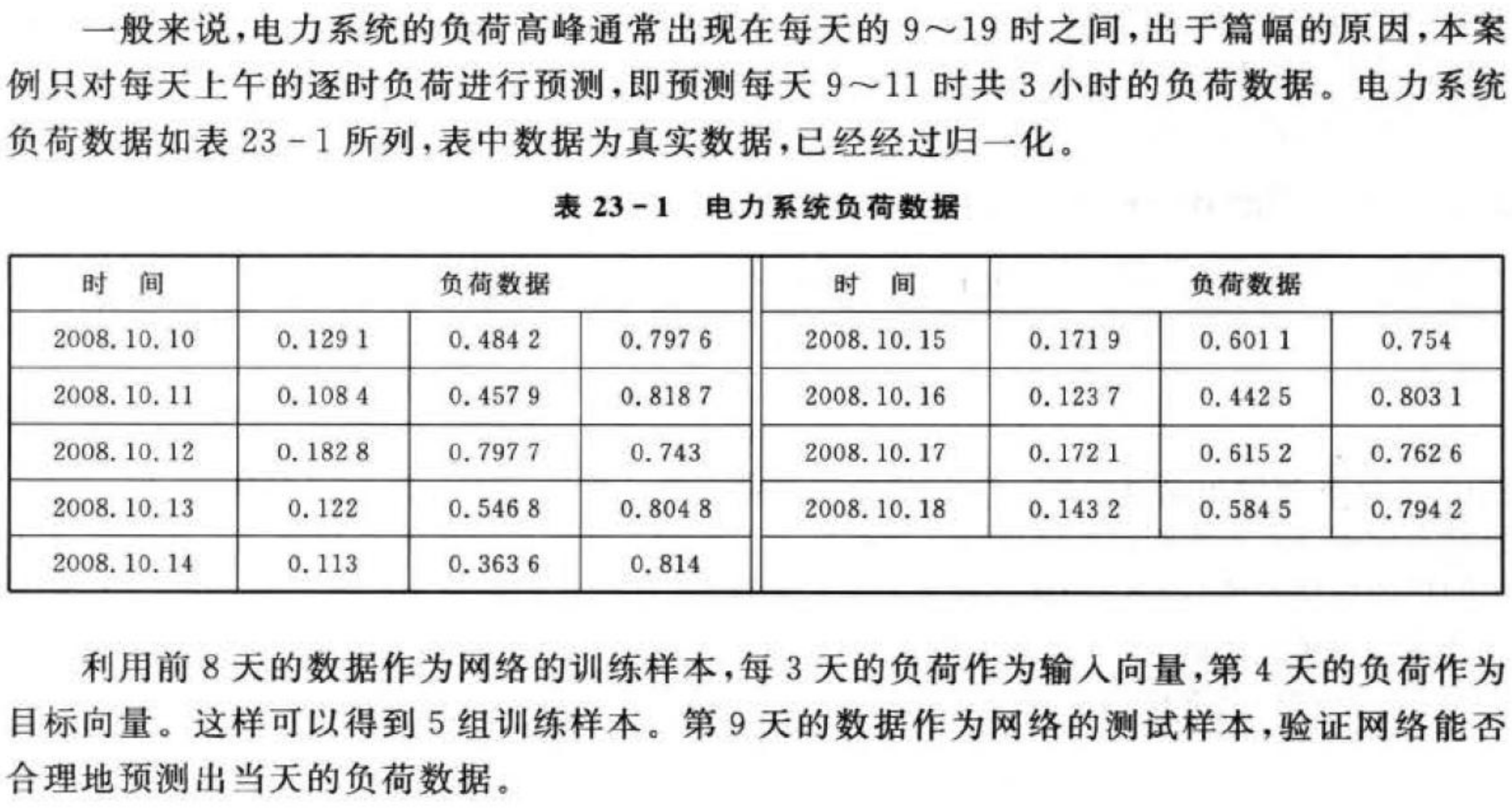
Elman网络模型



$y(k)$
输出层
$x(k)$
b_2
w^3
隐含层
$x_1 \cdots x_n$
$x_{c1} \cdots x_{cn}$
承接层
b_1
w^2
w^1
输入层
$x_c(k)$
$\cdots$
$u(k-1)$

使用Elman网络进行电力负荷预测

- 《Matlab神经网络43个案例》第197页



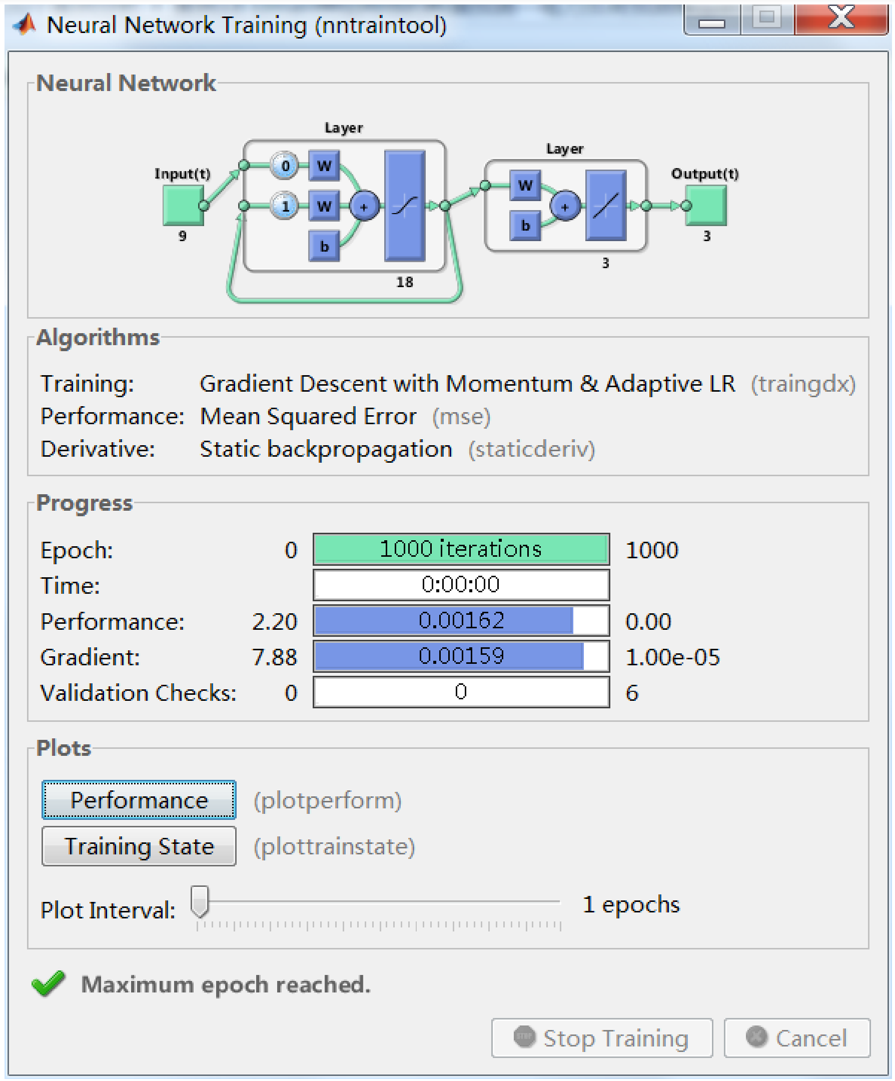
一般来说，电力系统的负荷高峰通常出现在每天的 9～19 时之间，出于篇幅的原因，本案例只对每天上午的逐时负荷进行预测，即预测每天 9～11 时共 3 小时的负荷数据。电力系统负荷数据如表 23－1 所列，表中数据为真实数据，已经经过归一化。

表 23－1　电力系统负荷数据

时　间	负荷数据			时　间	负荷数据		
2008.10.10	0.129 1	0.484 2	0.797 6	2008.10.15	0.171 9	0.601 1	0.754
2008.10.11	0.108 4	0.457 9	0.818 7	2008.10.16	0.123 7	0.442 5	0.803 1
2008.10.12	0.182 8	0.797 7	0.743	2008.10.17	0.172 1	0.615 2	0.762 6
2008.10.13	0.122	0.546 8	0.804 8	2008.10.18	0.143 2	0.584 5	0.794 2
2008.10.14	0.113	0.363 6	0.814				

利用前 8 天的数据作为网络的训练样本，每 3 天的负荷作为输入向量，第 4 天的负荷作为目标向量。这样可以得到 5 组训练样本。第 9 天的数据作为网络的测试样本，验证网络能否合理地预测出当天的负荷数据。

MATLAB运行



Neural Network

Algorithms

Training: Gradient Descent with Momentum & Adaptive LR (traingdx)
Performance: Mean Squared Error (mse)
Derivative: Static backpropagation (staticderiv)

Progress

Epoch:	0	1000 iterations	1000
Time:		0:00:00	
Performance:	2.20	0.00162	0.00
Gradient:	7.88	0.00159	1.00e-05
Validation Checks:	0	0	6

Plots

Performance (plotperform)
Training State (plottrainstate)
Plot Interval: 1 epochs

✓ Maximum epoch reached.

Stop Training Cancel

MATLAB运行

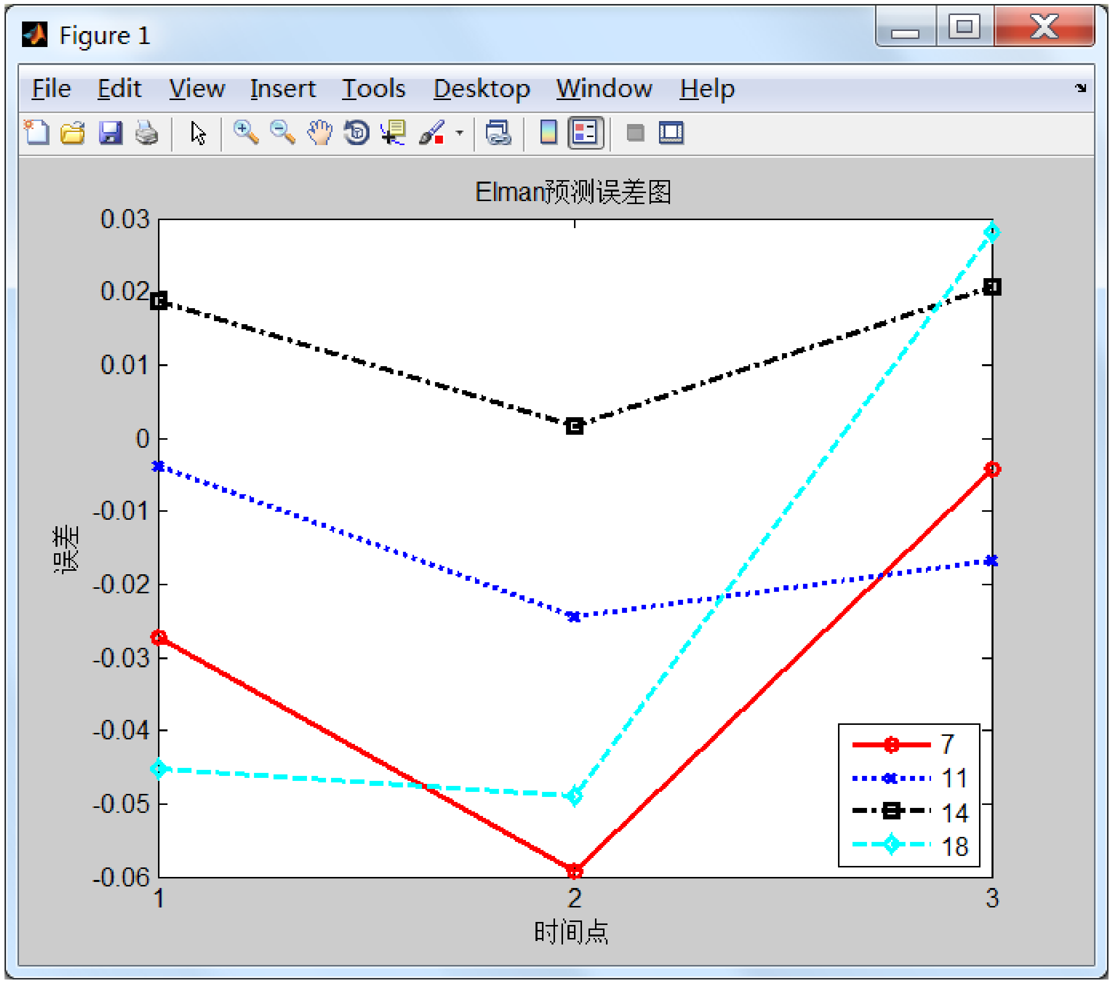
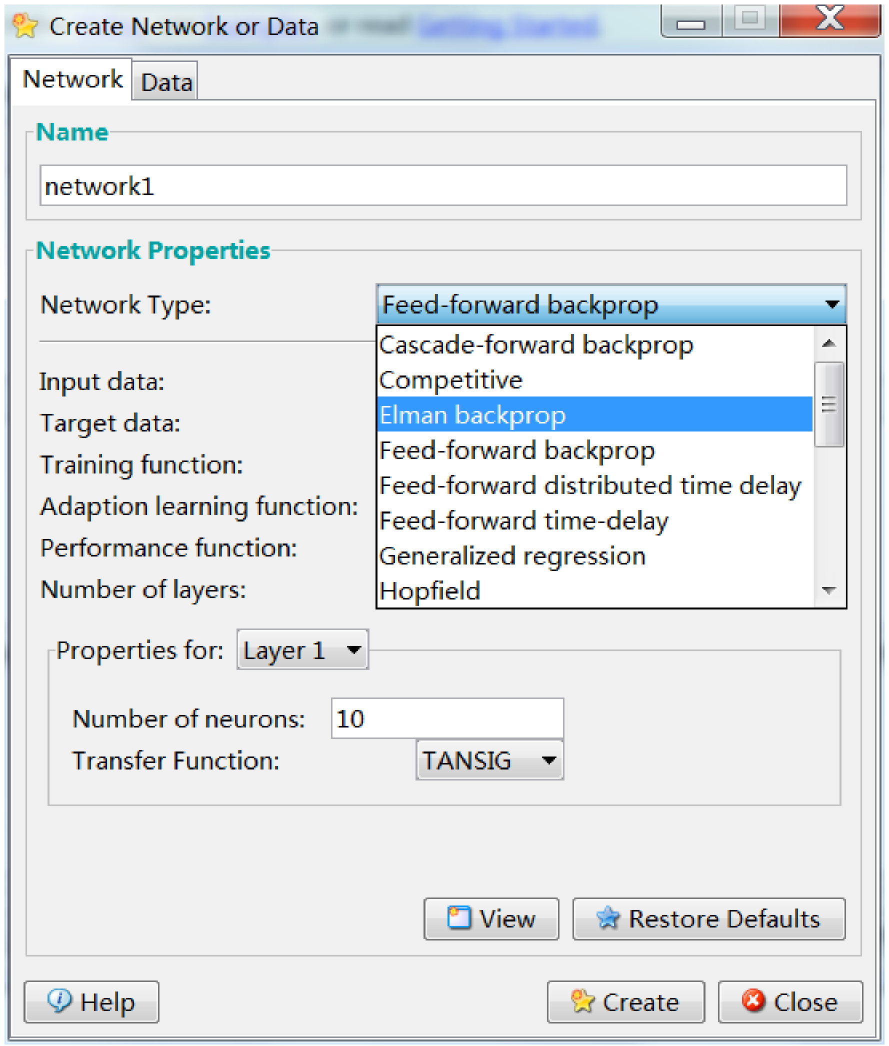


Figure 1
File Edit View Insert Tools Desktop Window Help
Elman预测误差图
误差
时间点
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
1
2
3
7
11
14
18

用MATLAB神经网络工具箱建立Elman网络


Create Network or Data

Network | Data

Name

network1

Network Properties

Network Type: Feed-forward backprop

- Cascade-forward backprop
- Competitive
- Elman backprop
- Feed-forward backprop
- Feed-forward distributed time delay
- Feed-forward time-delay
- Generalized regression
- Hopfield

Input data:

Target data:

Training function:

Adaption learning function:

Performance function:

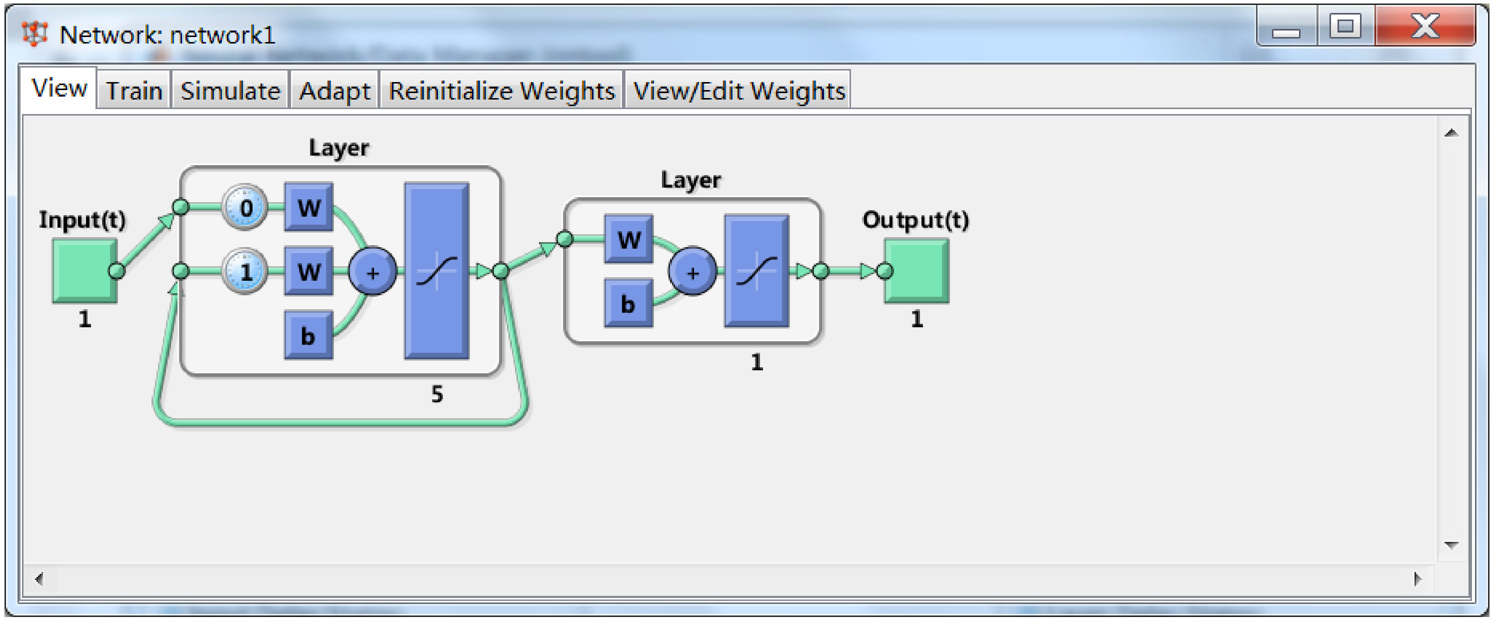
Number of layers:

Properties for: Layer 1

Number of neurons: 10

Transfer Function: TANSIG

用MATLAB神经网络工具箱建立Elman网络



NARX

- 韩立群书第147页

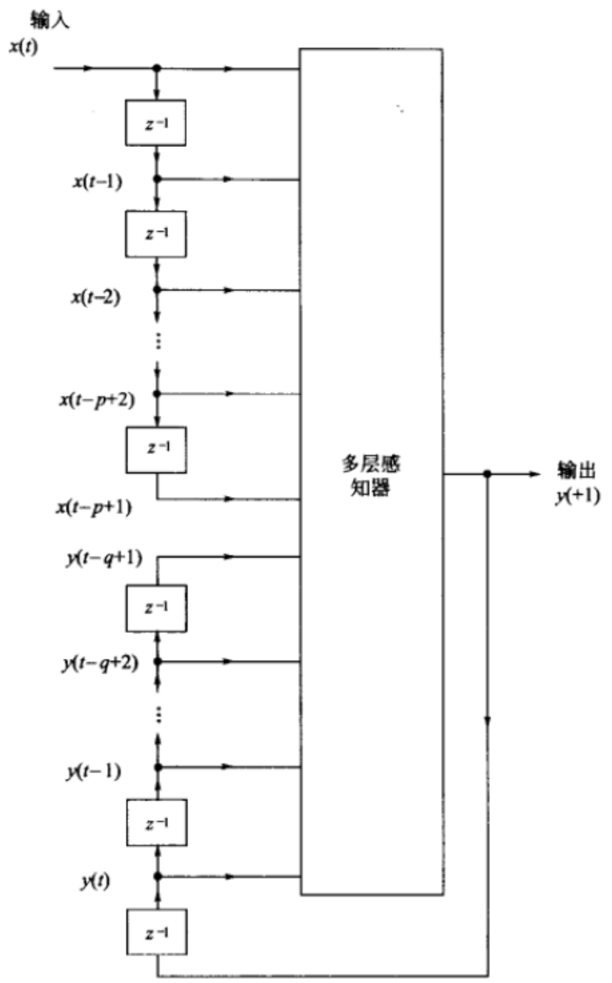


图 6.15 一种通用递归网络模型

NARX

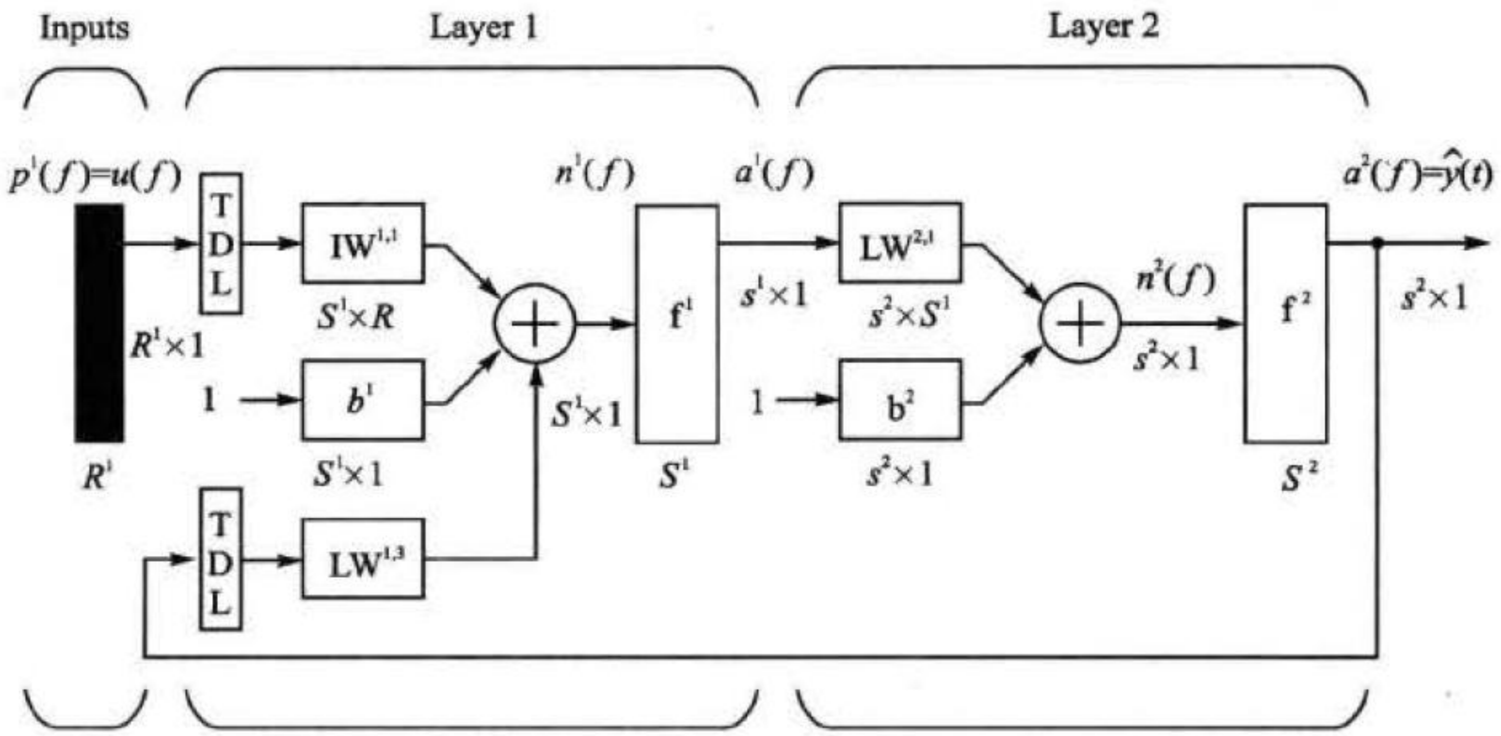
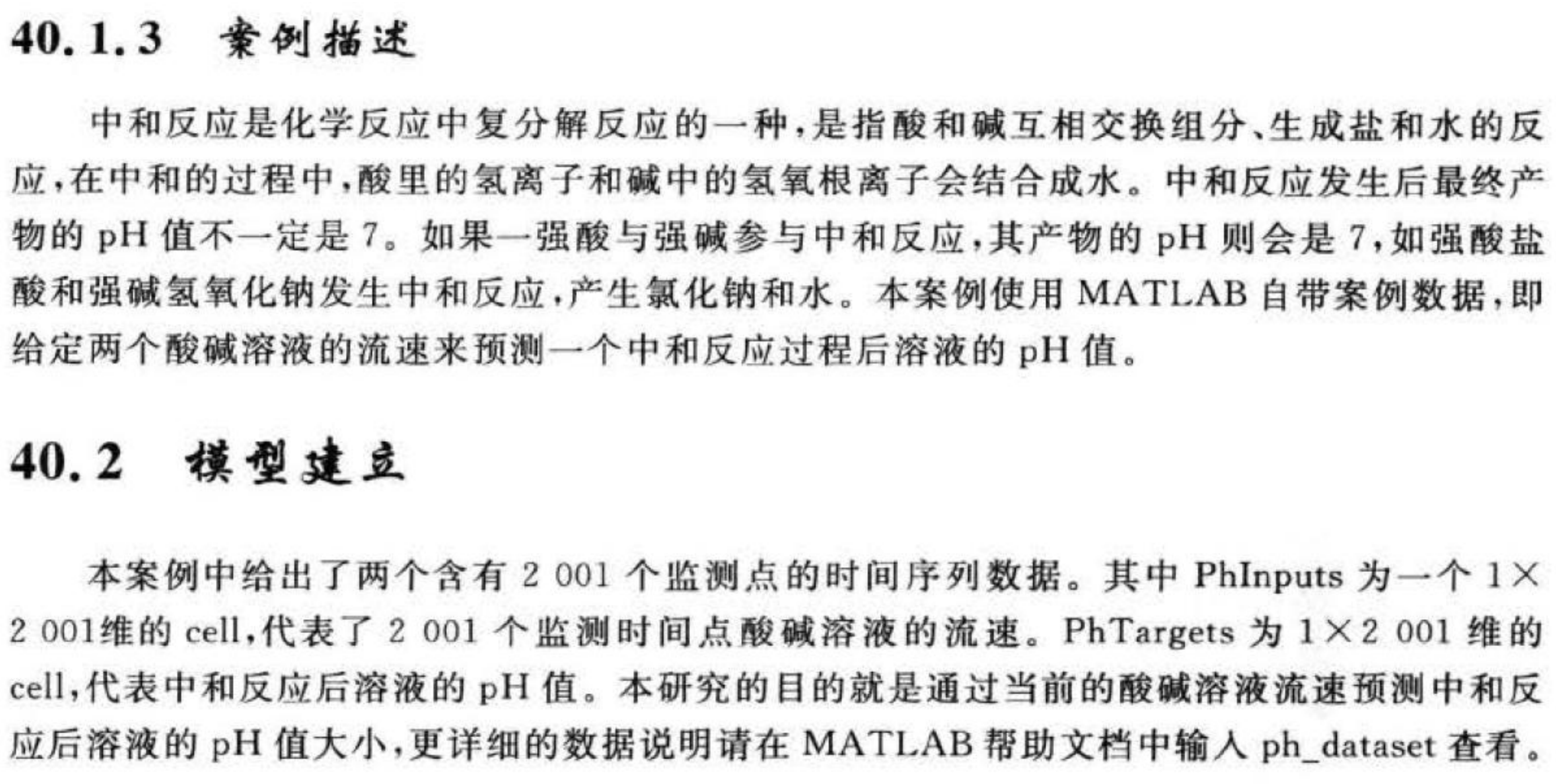


图 40－2 NARX 神经网络详细结构图

利用NARX处理时间序列数据

ATAGURU 炼数成金

- 《Matlab神经网络43个案例》第356页



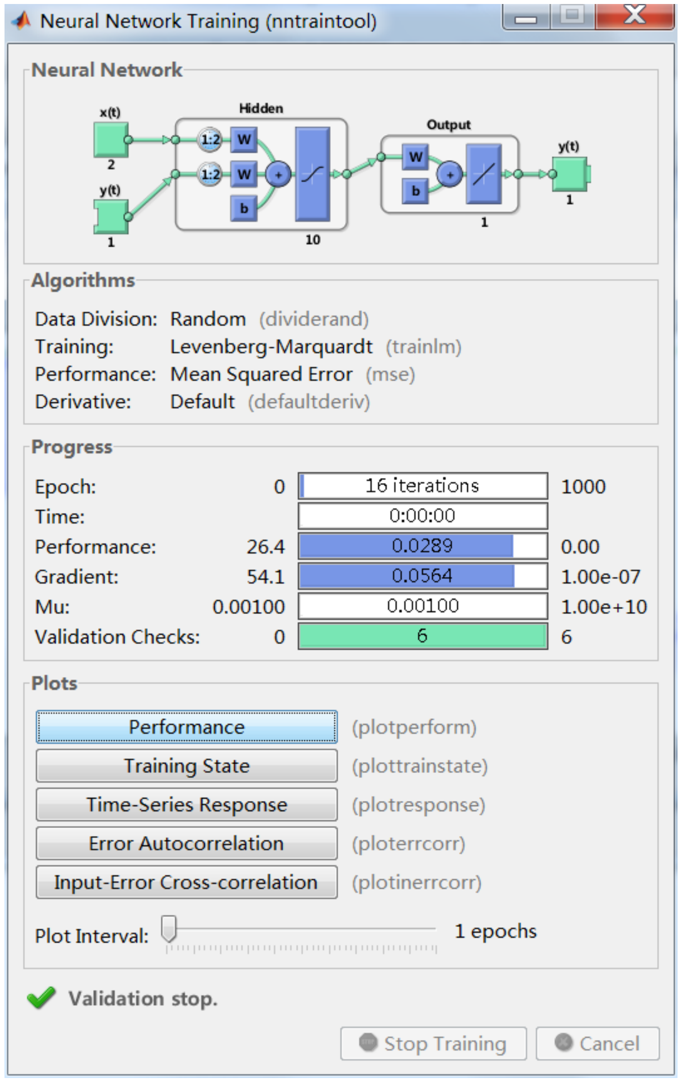
40.1.3 案例描述

中和反应是化学反应中复分解反应的一种，是指酸和碱互相交换组分、生成盐和水的反应，在中和的过程中，酸里的氢离子和碱中的氢氧根离子会结合成水。中和反应发生后最终产物的 pH 值不一定是 7。如果一强酸与强碱参与中和反应，其产物的 pH 则会是 7，如强酸盐酸和强碱氢氧化钠发生中和反应，产生氯化钠和水。本案例使用 MATLAB 自带案例数据，即给定两个酸碱溶液的流速来预测一个中和反应过程后溶液的 pH 值。

40.2 模型建立

本案例中给出了两个含有 2 001 个监测点的时间序列数据。其中 PhInputs 为一个 1×2 001维的 cell，代表了 2 001 个监测时间点酸碱溶液的流速。PhTargets 为 1×2 001 维的 cell，代表中和反应后溶液的 pH 值。本研究的目的就是通过当前的酸碱溶液流速预测中和反应后溶液的 pH 值大小，更详细的数据说明请在 MATLAB 帮助文档中输入 ph_dataset 查看。

MATLAB运行


Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division:	Random (dividerand)
Training:	Levenberg-Marquardt (trainlm)
Performance:	Mean Squared Error (mse)
Derivative:	Default (defaultderiv)

Progress

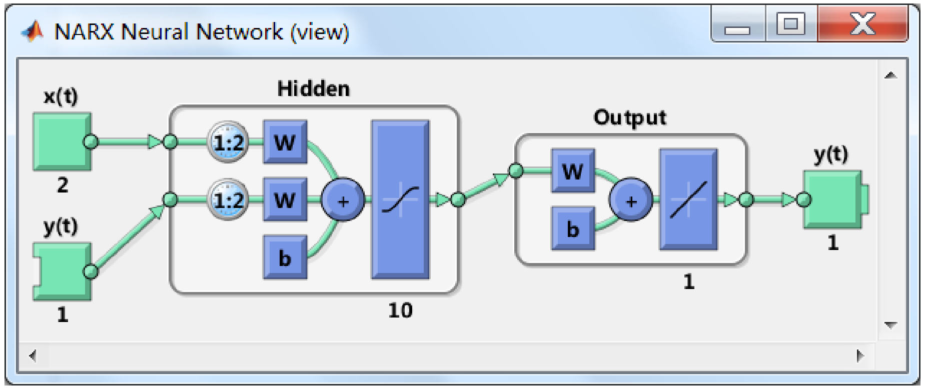
Epoch:	0	16 iterations	1000
Time:		0:00:00	
Performance:	26.4	0.0289	0.00
Gradient:	54.1	0.0564	1.00e-07
Mu:	0.00100	0.00100	1.00e+10
Validation Checks:	0	6	6

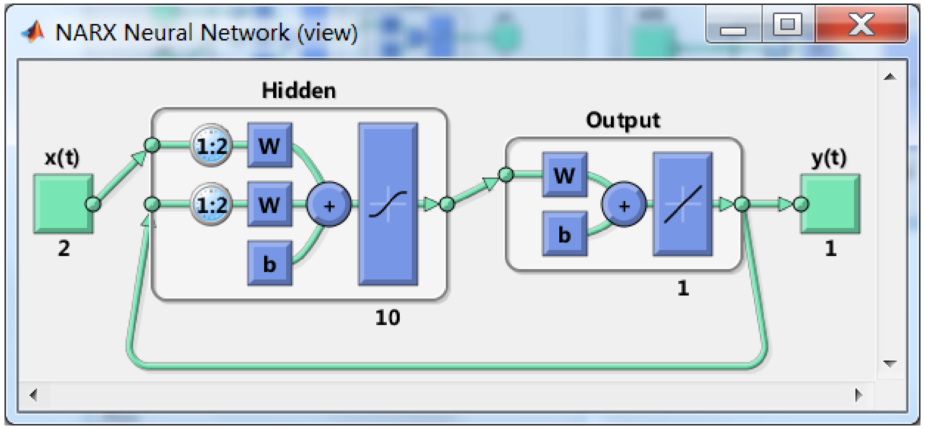
Plots

Performance	(plotperform)
Training State	(plottrainstate)
Time-Series Response	(plotresponse)
Error Autocorrelation	(ploterrcorr)
Input-Error Cross-correlation	(plotinerrcorr)

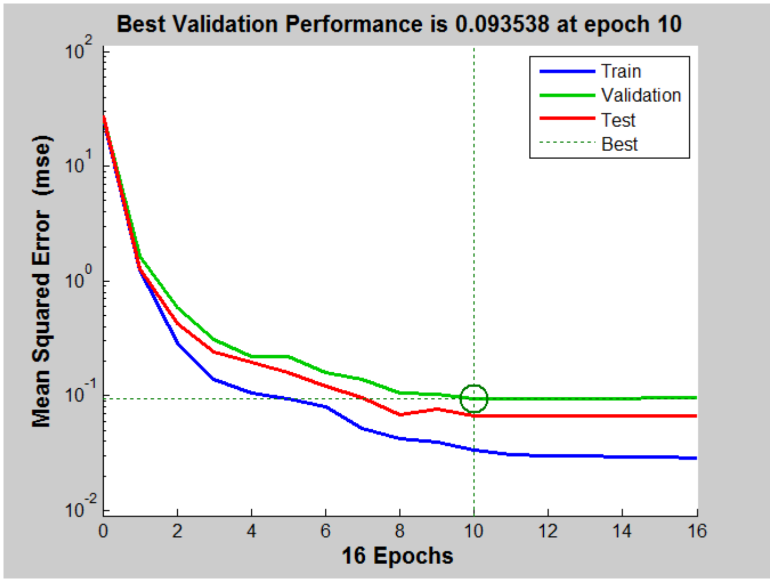
Plot Interval: 1 epochs

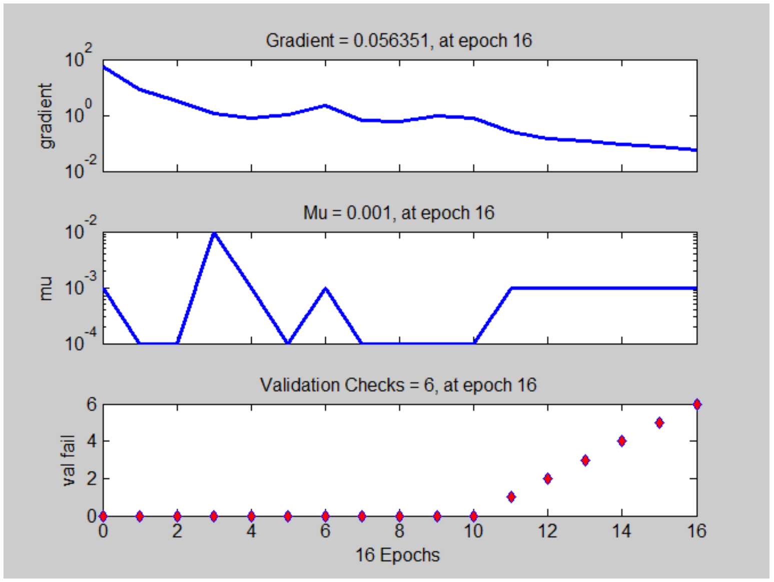
✓ Validation stop.

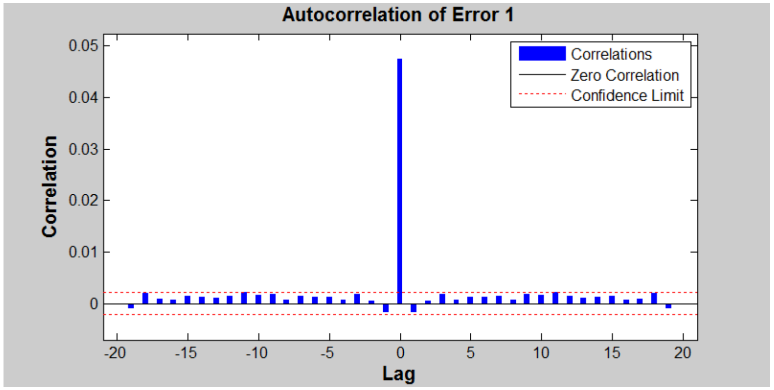


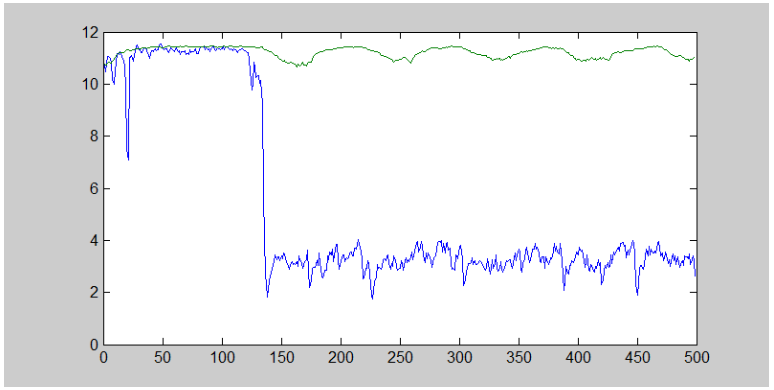


结果


Best Validation Performance is 0.093538 at epoch 10
Mean Squared Error (mse)
16 Epochs
Train
Validation
Test
Best

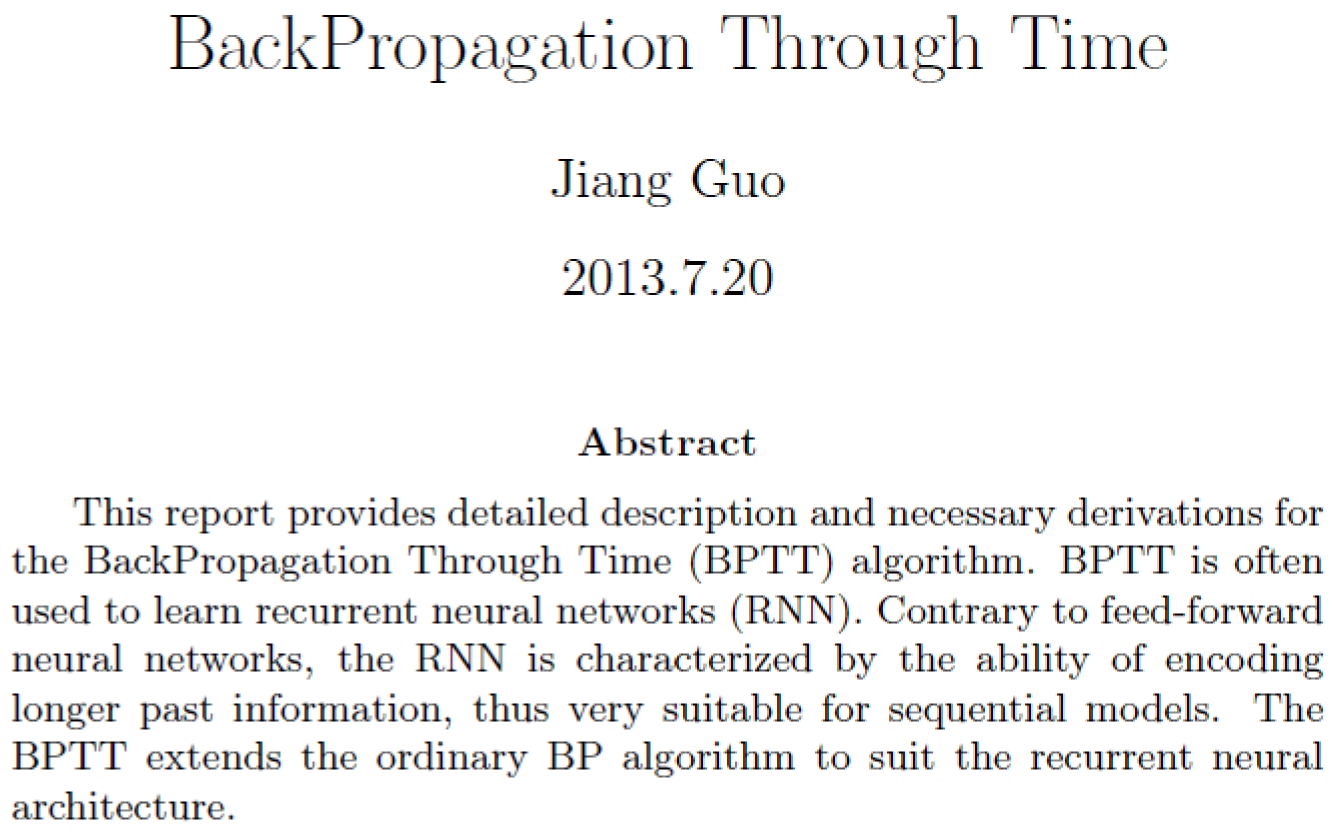

Gradient = 0.056351, at epoch 16
gradient
Mu = 0.001, at epoch 16
mu
Validation Checks = 6, at epoch 16
val fail
16 Epochs


Autocorrelation of Error 1
Correlation
Lag
Correlations
Zero Correlation
Confidence Limit



BPTT

- 参考论文



BackPropagation Through Time

Jiang Guo

2013.7.20

Abstract

This report provides detailed description and necessary derivations for the BackPropagation Through Time (BPTT) algorithm. BPTT is often used to learn recurrent neural networks (RNN). Contrary to feed-forward neural networks, the RNN is characterized by the ability of encoding longer past information, thus very suitable for sequential models. The BPTT extends the ordinary BP algorithm to suit the recurrent neural architecture.

展开的RNN

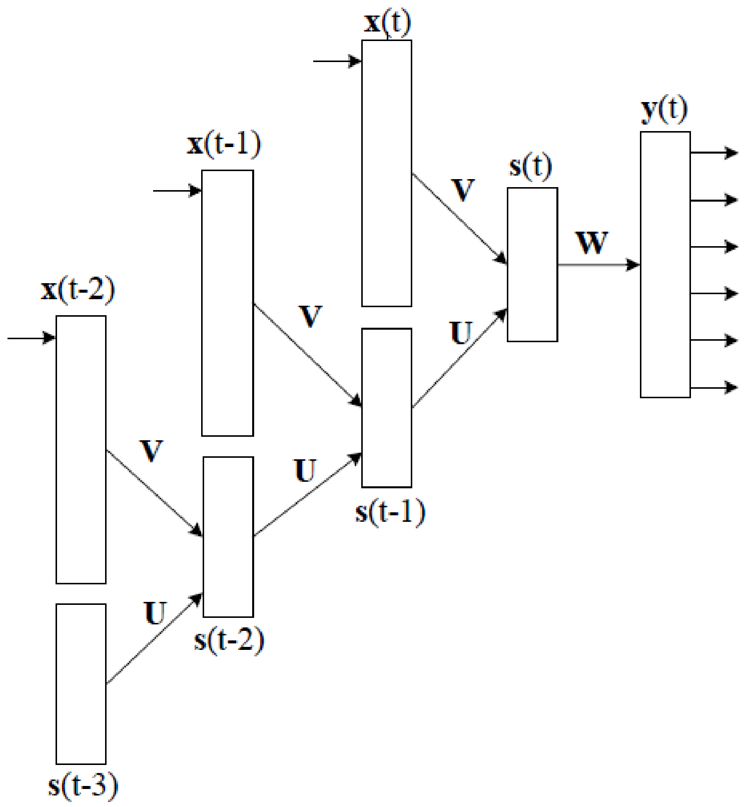
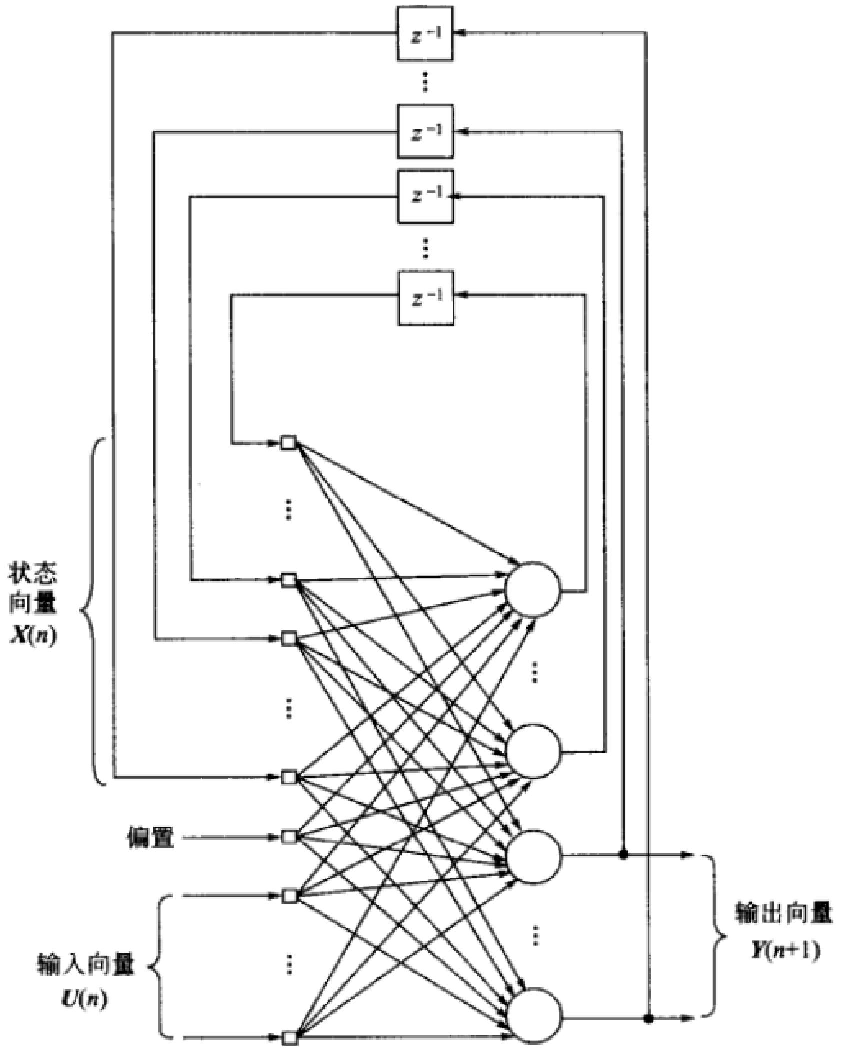


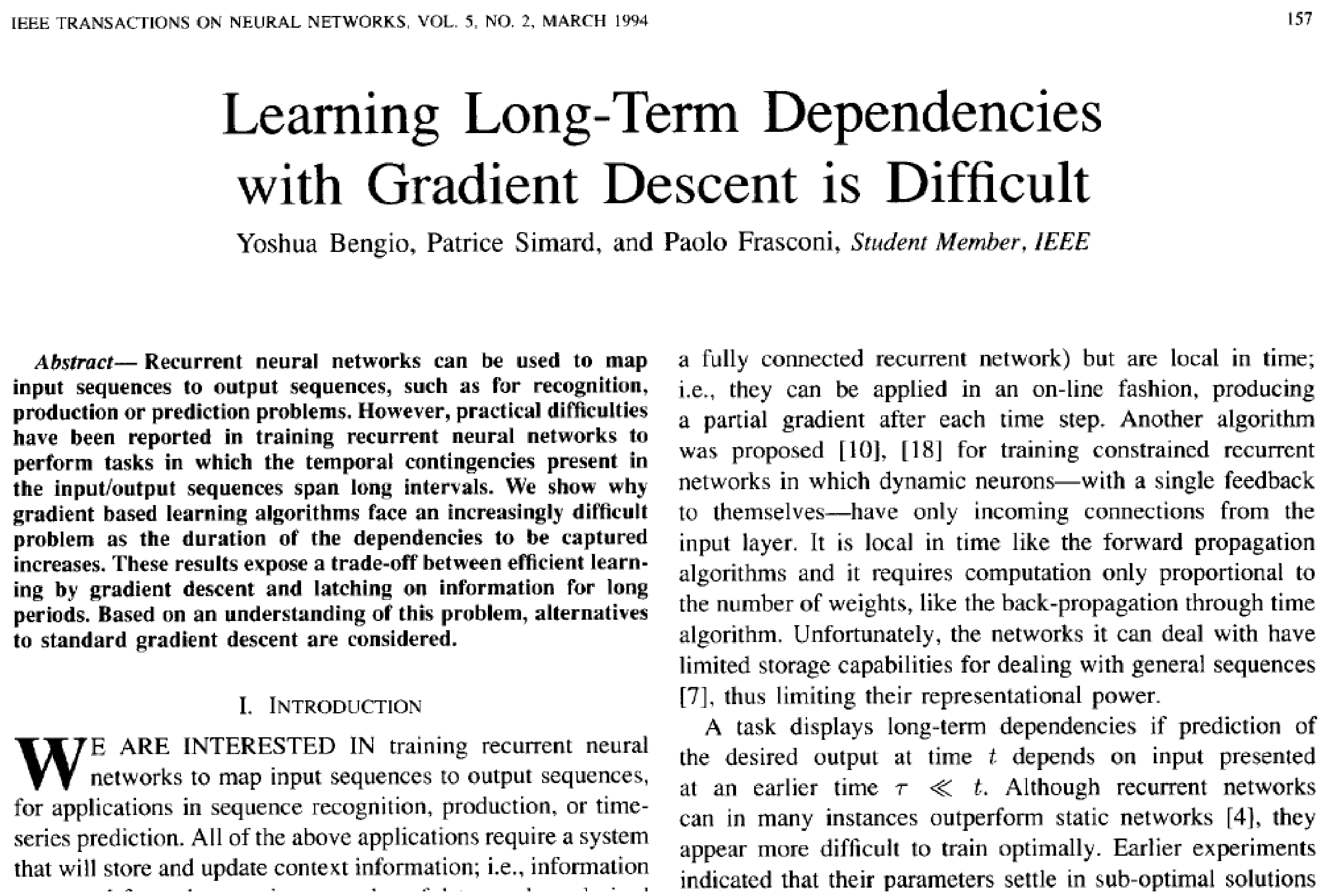
Figure 2: An unfolded recurrent neural network.

RTRL

- 参考韩力群书第149页



传统RNN面对的困难



Learning Long-Term Dependencies with Gradient Descent is Difficult

Yoshua Bengio, Patrice Simard, and Paolo Frasconi, *Student Member, IEEE*

***Abstract*— Recurrent neural networks can be used to map input sequences to output sequences, such as for recognition, production or prediction problems. However, practical difficulties have been reported in training recurrent neural networks to perform tasks in which the temporal contingencies present in the input/output sequences span long intervals. We show why gradient based learning algorithms face an increasingly difficult problem as the duration of the dependencies to be captured increases. These results expose a trade-off between efficient learning by gradient descent and latching on information for long periods. Based on an understanding of this problem, alternatives to standard gradient descent are considered.**

I. Introduction

WE ARE INTERESTED IN training recurrent neural networks to map input sequences to output sequences, for applications in sequence recognition, production, or time-series prediction. All of the above applications require a system that will store and update context information; i.e., information

a fully connected recurrent network) but are local in time; i.e., they can be applied in an on-line fashion, producing a partial gradient after each time step. Another algorithm was proposed [10], [18] for training constrained recurrent networks in which dynamic neurons—with a single feedback to themselves—have only incoming connections from the input layer. It is local in time like the forward propagation algorithms and it requires computation only proportional to the number of weights, like the back-propagation through time algorithm. Unfortunately, the networks it can deal with have limited storage capabilities for dealing with general sequences [7], thus limiting their representational power.

A task displays long-term dependencies if prediction of the desired output at time t depends on input presented at an earlier time $\tau \ll t$. Although recurrent networks can in many instances outperform static networks [4], they appear more difficult to train optimally. Earlier experiments indicated that their parameters settle in sub-optimal solutions

梯度消失

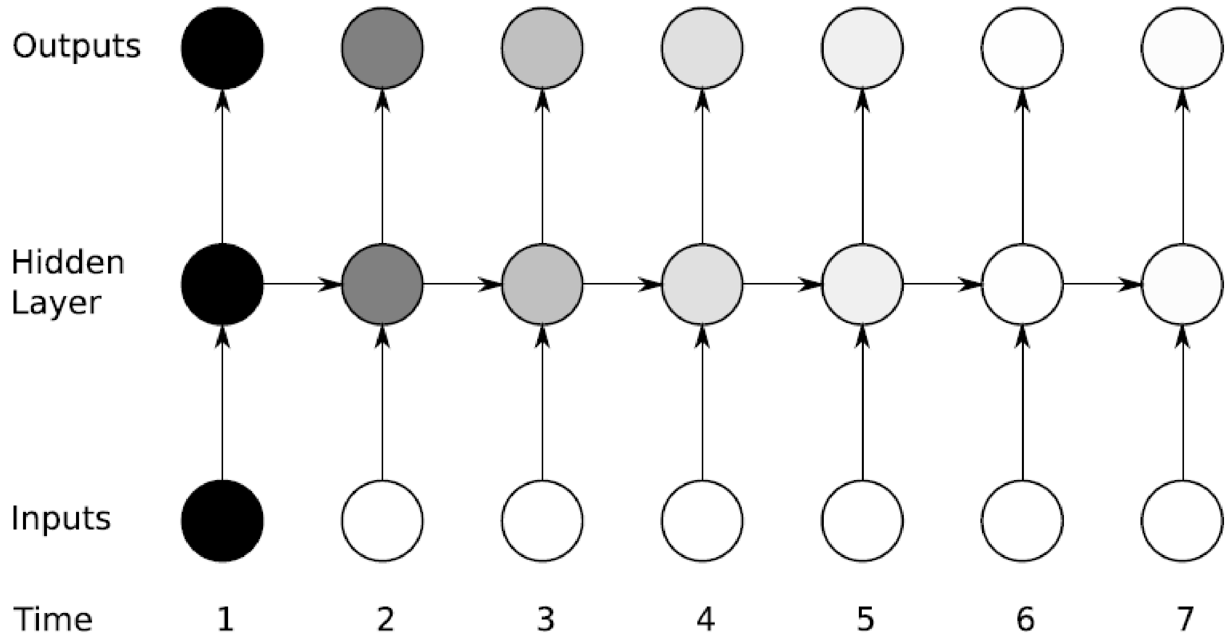
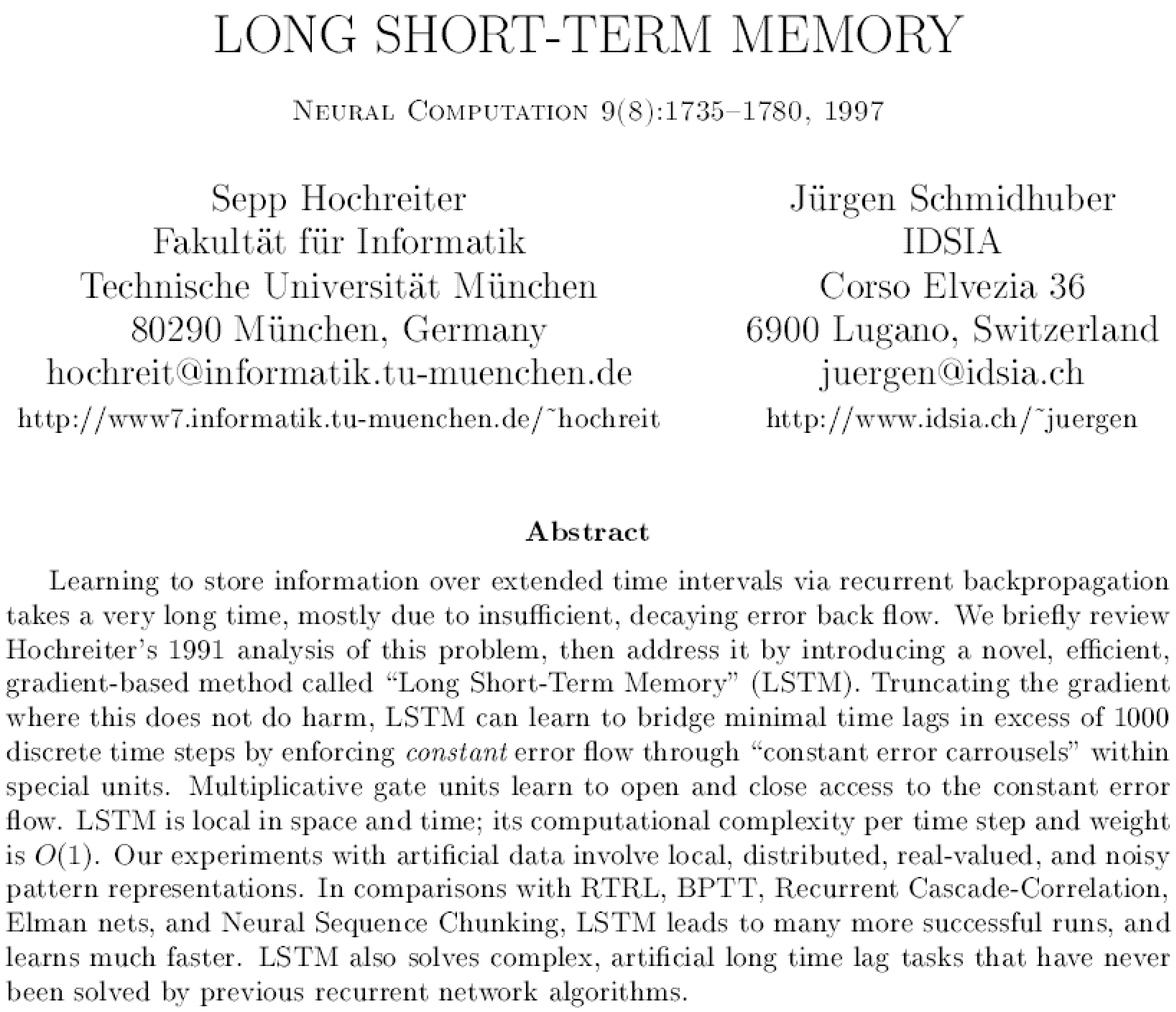


Fig. 4.1 The vanishing gradient problem for RNNs. The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity). The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

LSTM网络



LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735–1780, 1997

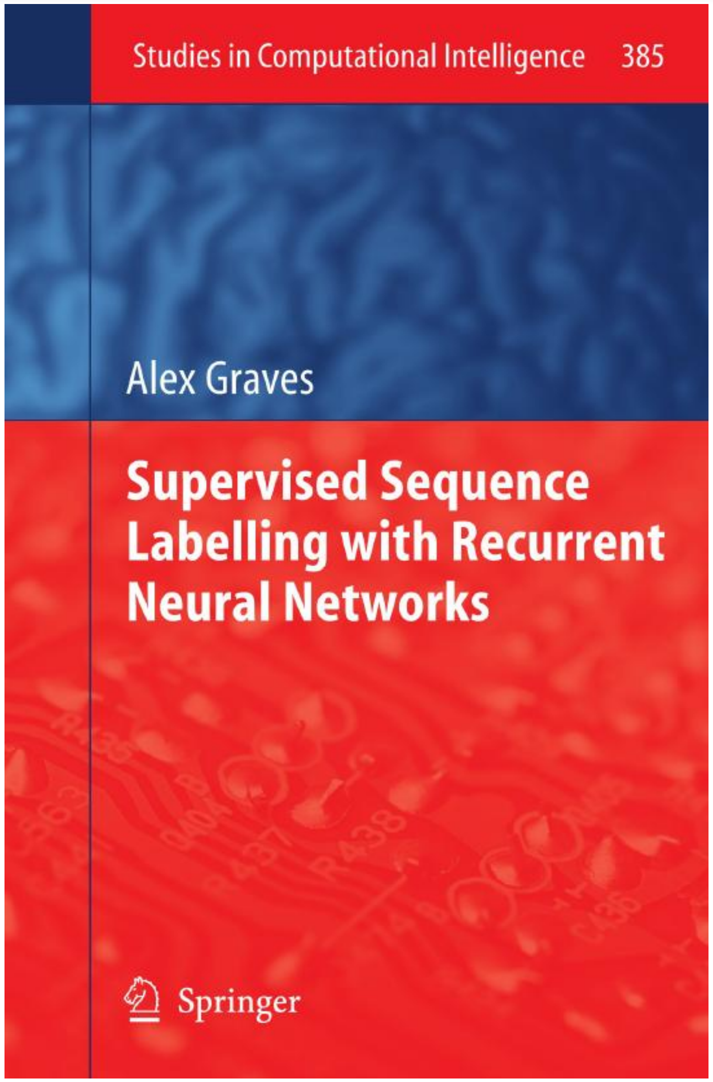
Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

参考书

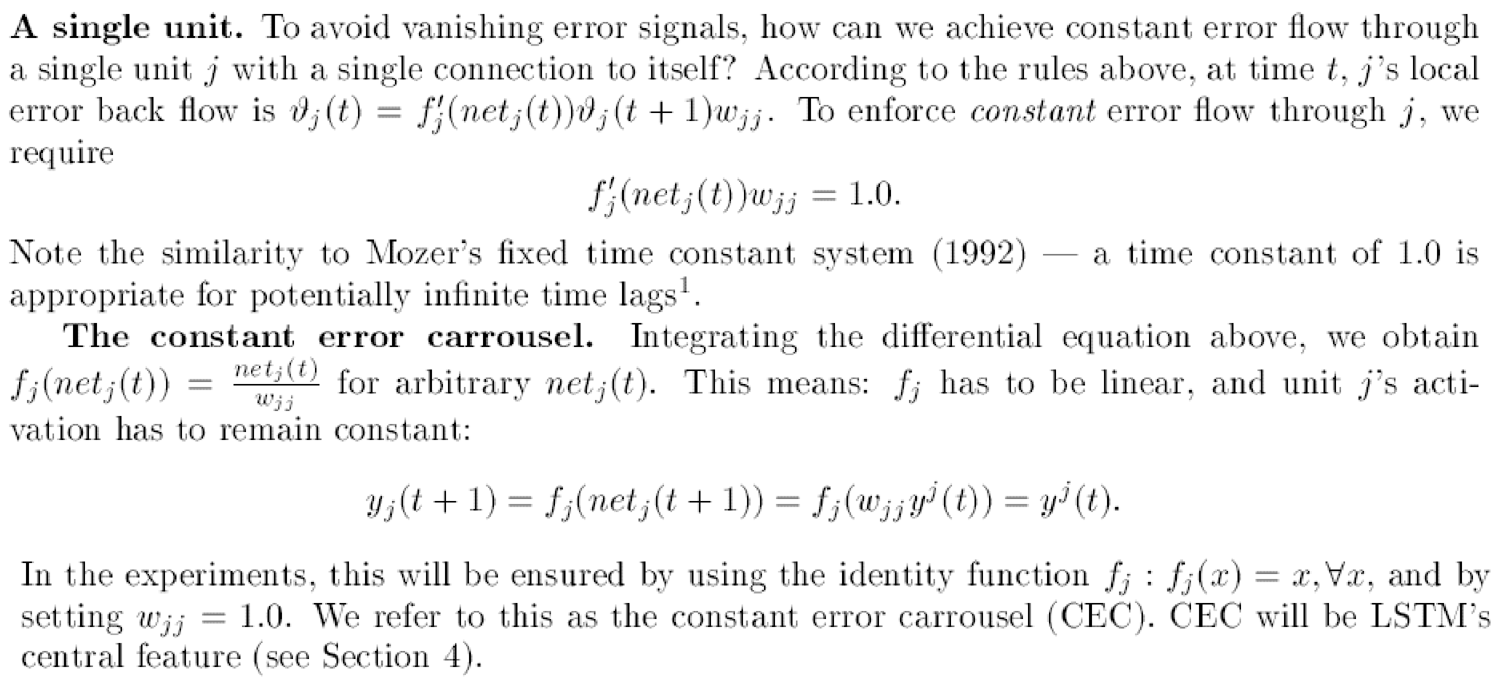

Studies in Computational Intelligence 385

Alex Graves

Supervised Sequence Labelling with Recurrent Neural Networks

Springer

对抗梯度消失：CEC



A single unit. To avoid vanishing error signals, how can we achieve constant error flow through a single unit j with a single connection to itself? According to the rules above, at time t, j's local error back flow is $\vartheta_j(t) = f'_j(net_j(t))\vartheta_j(t+1)w_{jj}$. To enforce *constant* error flow through j, we require

$$f'_j(net_j(t))w_{jj} = 1.0.$$

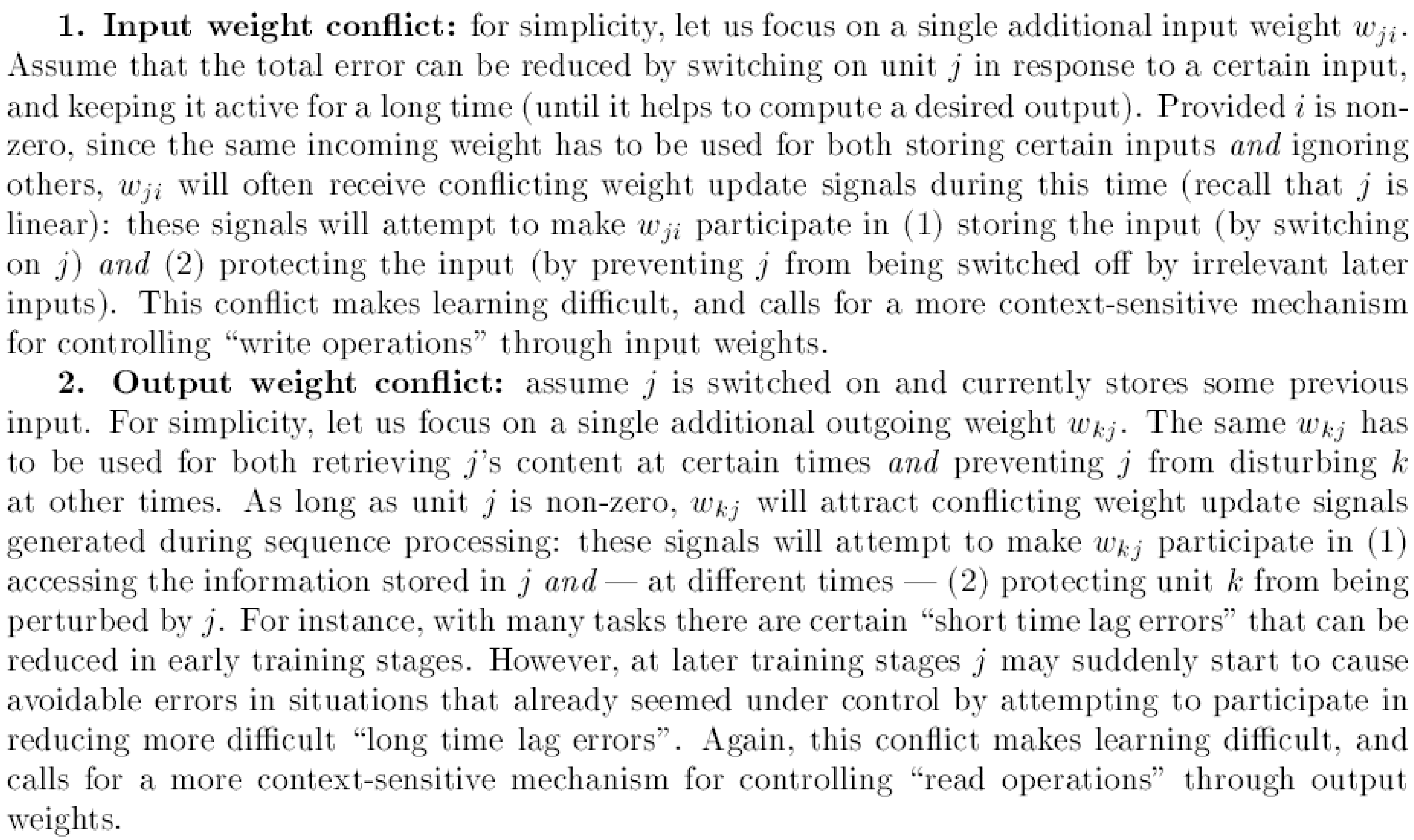
Note the similarity to Mozer's fixed time constant system (1992) — a time constant of 1.0 is appropriate for potentially infinite time lags[1].

The constant error carrousel. Integrating the differential equation above, we obtain $f_j(net_j(t)) = \frac{net_j(t)}{w_{jj}}$ for arbitrary $net_j(t)$. This means: f_j has to be linear, and unit j's activation has to remain constant:

$$y_j(t+1) = f_j(net_j(t+1)) = f_j(w_{jj}y^j(t)) = y^j(t).$$

In the experiments, this will be ensured by using the identity function $f_j : f_j(x) = x, \forall x$, and by setting $w_{jj} = 1.0$. We refer to this as the constant error carrousel (CEC). CEC will be LSTM's central feature (see Section 4).

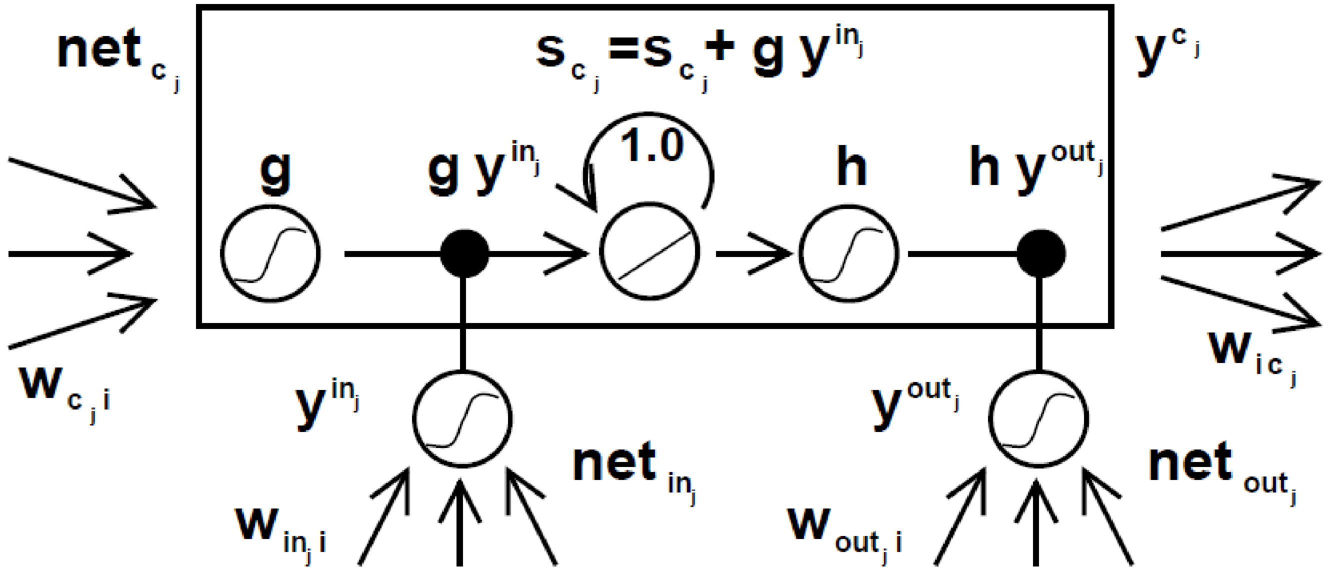
CEC带来的问题



1. Input weight conflict: for simplicity, let us focus on a single additional input weight w_{ji}. Assume that the total error can be reduced by switching on unit j in response to a certain input, and keeping it active for a long time (until it helps to compute a desired output). Provided i is non-zero, since the same incoming weight has to be used for both storing certain inputs *and* ignoring others, w_{ji} will often receive conflicting weight update signals during this time (recall that j is linear): these signals will attempt to make w_{ji} participate in (1) storing the input (by switching on j) *and* (2) protecting the input (by preventing j from being switched off by irrelevant later inputs). This conflict makes learning difficult, and calls for a more context-sensitive mechanism for controlling "write operations" through input weights.

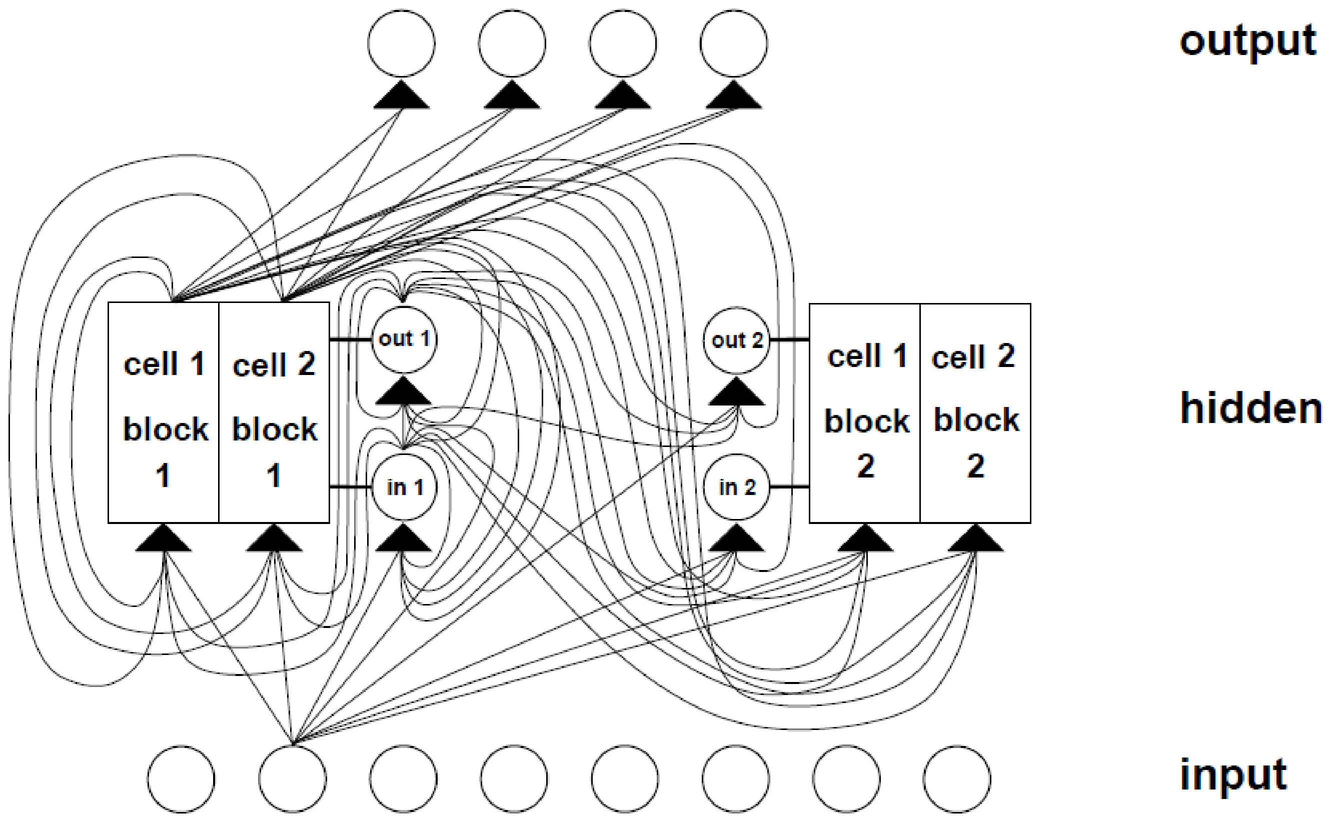
2. Output weight conflict: assume j is switched on and currently stores some previous input. For simplicity, let us focus on a single additional outgoing weight w_{kj}. The same w_{kj} has to be used for both retrieving j's content at certain times *and* preventing j from disturbing k at other times. As long as unit j is non-zero, w_{kj} will attract conflicting weight update signals generated during sequence processing: these signals will attempt to make w_{kj} participate in (1) accessing the information stored in j *and* — at different times — (2) protecting unit k from being perturbed by j. For instance, with many tasks there are certain "short time lag errors" that can be reduced in early training stages. However, at later training stages j may suddenly start to cause avoidable errors in situations that already seemed under control by attempting to participate in reducing more difficult "long time lag errors". Again, this conflict makes learning difficult, and calls for a more context-sensitive mechanism for controlling "read operations" through output weights.

原始LSTM

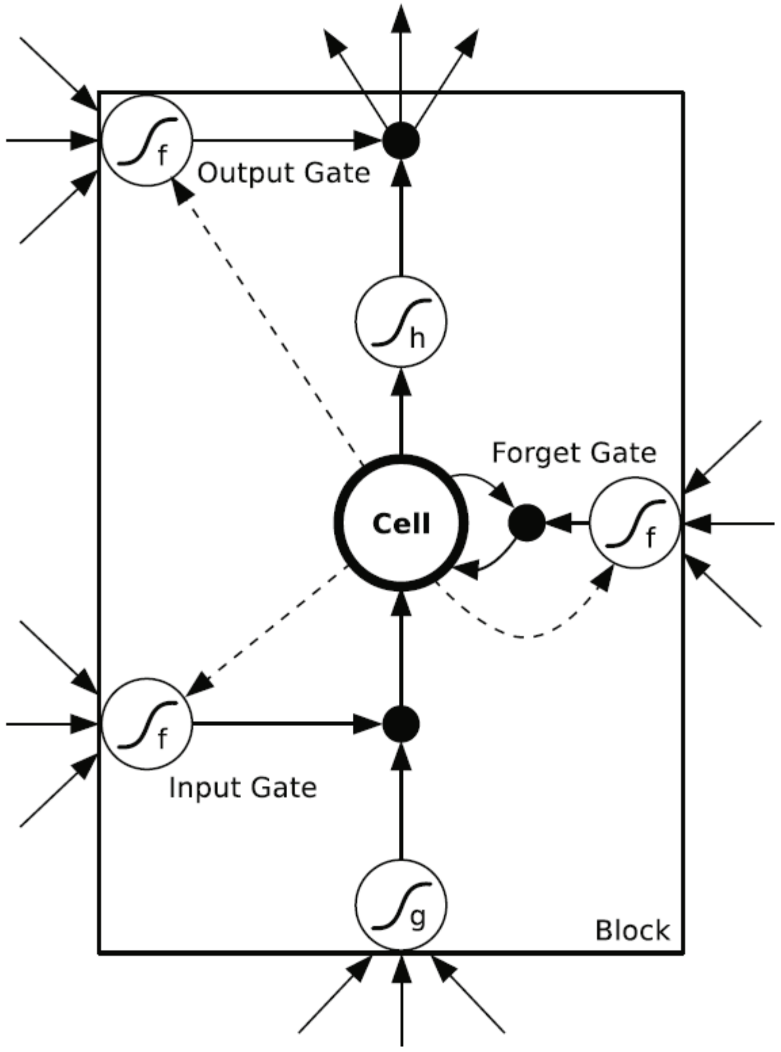


net_{c_j}
$s_{c_j} = s_{c_j} + g\, y^{in_j}$
y^{c_j}
g
$g\, y^{in_j}$
1.0
h
$h\, y^{out_j}$
$w_{c_j i}$
y^{in_j}
y^{out_j}
$w_{i c_j}$
net_{in_j}
net_{out_j}
$w_{in_j i}$
$w_{out_j i}$

LSTM网络的一个例子



“现代的”LSTM



前向传送公式

Input Gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{4.2}$$

$$b_\iota^t = f(a_\iota^t) \tag{4.3}$$

Forget Gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} \tag{4.4}$$

$$b_\phi^t = f(a_\phi^t) \tag{4.5}$$

Cells

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{4.6}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{4.7}$$

前向传送公式

Output Gates

$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{4.8}$$

$$b_\omega^t = f(a_\omega^t) \tag{4.9}$$

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t) \tag{4.10}$$

反向传递

$$\epsilon_c^t \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial b_c^t} \qquad \epsilon_s^t \overset{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{h=1}^{H} w_{ch}\delta_h^{t+1} \tag{4.11}$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t)\sum_{c=1}^{C} h(s_c^t)\epsilon_c^t \tag{4.12}$$

反向传递

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t \tag{4.13}$$

Cells

$$\delta_c^t = b_\iota^t g'(a_c^t)\epsilon_s^t \tag{4.14}$$

Forget Gates

$$\delta_\phi^t = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t \tag{4.15}$$

Input Gates

$$\delta_\iota^t = f'(a_\iota^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t \tag{4.16}$$

LSTM怎样解决梯度消失问题？

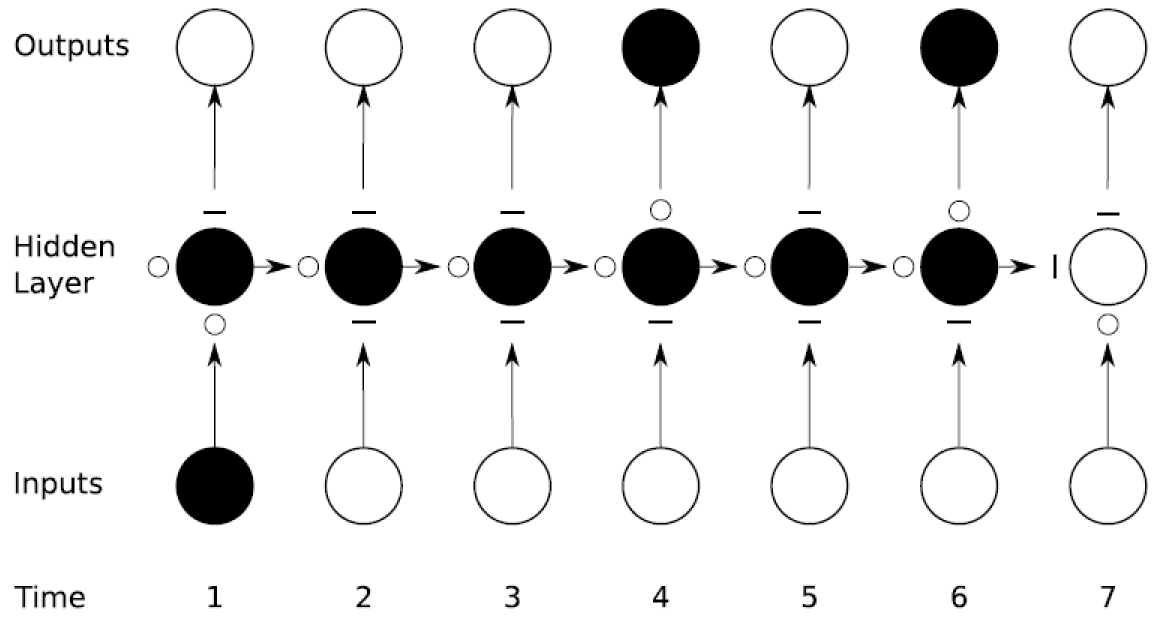


Fig. 4.4 Preservation of gradient information by LSTM. As in Figure 4.1 the shading of the nodes indicates their sensitivity to the inputs at time one; in this case the black nodes are maximally sensitive and the white nodes are entirely insensitive. The state of the input, forget, and output gates are displayed below, to the left and above the hidden layer respectively. For simplicity, all gates are either entirely open (‘O’) or closed (‘—’). The memory cell ‘remembers’ the first input as long as the forget gate is open and the input gate is closed. The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

样例

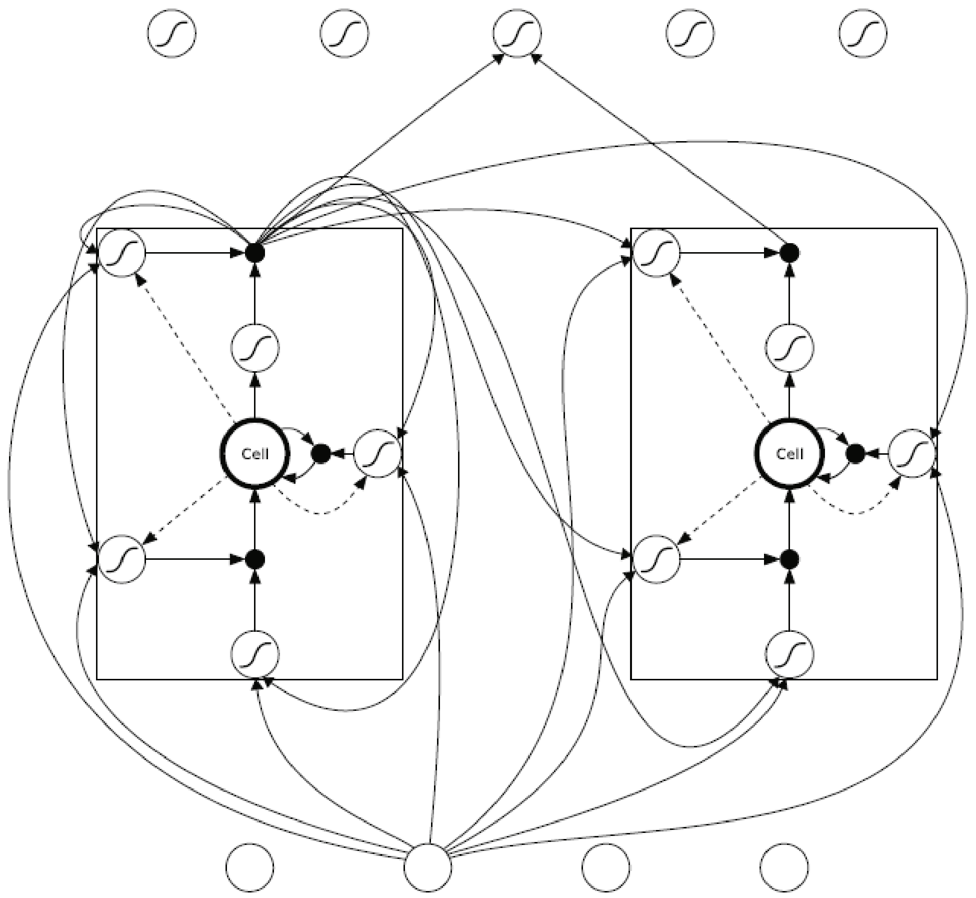


Fig. 4.3 An LSTM network. The network consists of four input units, a hidden layer of two single-cell LSTM memory blocks and five output units. Not all connections are shown. Note that each block has four inputs but only one output.

参考文章

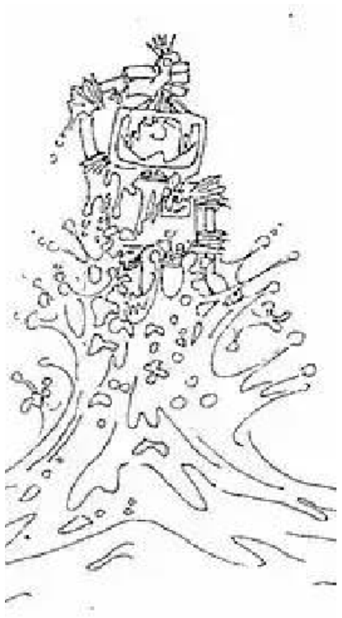
- http://blog.csdn.net/shincling/article/details/49362161
- http://www.tuicool.com/articles/6ZF7JzE
- http://blog.csdn.net/rtygbwwwerr/article/details/50422296
- http://blog.csdn.net/Dark_Scope/article/details/47056361
- http://www.csdn.net/article/2015-06-05/2824880

J. Schmidhuber

- http://people.idsia.ch/~juergen/
- https://en.wikipedia.org/wiki/J%C3%BCrgen_Schmidhuber
- 递归神经网络（RNN）之父
- 瑞士人工智能实验室 IDSIA 的科学事务主管，同时任教于卢加诺大学和瑞士南部应用科学与艺术学院。
- 1987年和1991年在慕尼黑工业大学先后获得计算机科学的学士和博士学位。自1987年以来，一直引领着自我改进式（self-improving）通用问题求解程序（problem-solver）的研究。从1991年开始，他成为深度学习神经网络领域的开拓者。

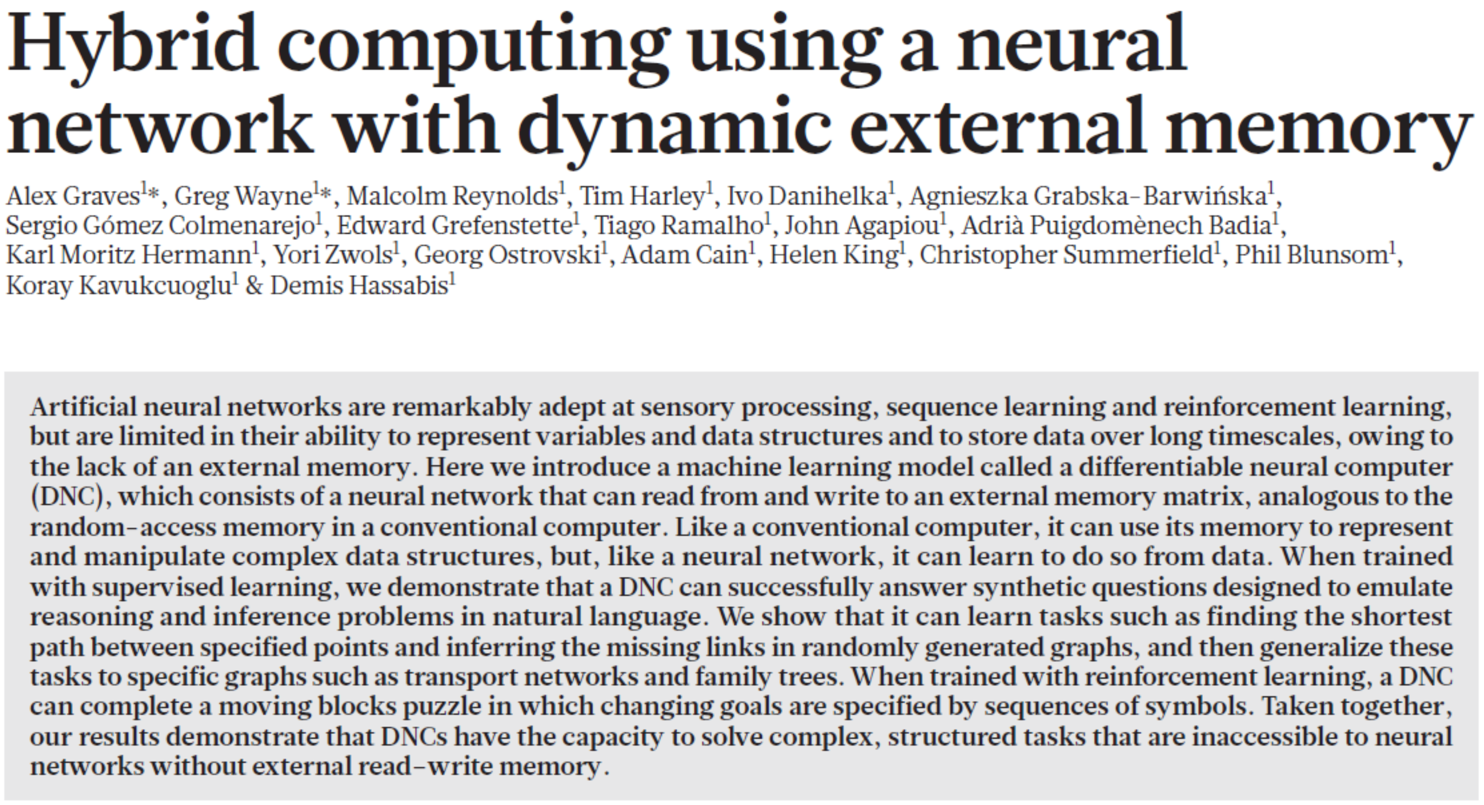
J. Schmidhuber

- 他在 IDSIA 和慕尼黑工业大学的研究团队开发了一种递归神经网络LSTM，并率先在正式的国际性比赛中获胜。这些技术革新了手写体识别、语音识别、机器翻译和图片注释技术，现在被谷歌、微软、IBM、百度和其他很多公司应用。谷歌 DeepMind 的创始人中，有两位都是他以前的学生。
- 2009年，Schmidhuber 当选欧洲科学与艺术学院的院士。他获得过很多奖项，包括 2013 年国际神经网络协会的亥姆霍兹奖，以及 2016 年电气与电子工程师协会的神经网络先锋奖。2014 年，他参与创办了人工智能公司 NNAISENSE，旨在打造第一个有实用价值的通用人工智能。
- http://www.92to.com/xinwen/2016/10-08/11308296.html



Alex Graves

- https://en.wikipedia.org/wiki/Alex_Graves_(computer_scientist)
- http://www.cs.toronto.edu/~graves/



Hybrid computing using a neural network with dynamic external memory

Alex Graves[1]*, Greg Wayne[1]*, Malcolm Reynolds[1], Tim Harley[1], Ivo Danihelka[1], Agnieszka Grabska-Barwińska[1], Sergio Gómez Colmenarejo[1], Edward Grefenstette[1], Tiago Ramalho[1], John Agapiou[1], Adrià Puigdomènech Badia[1], Karl Moritz Hermann[1], Yori Zwols[1], Georg Ostrovski[1], Adam Cain[1], Helen King[1], Christopher Summerfield[1], Phil Blunsom[1], Koray Kavukcuoglu[1] & Demis Hassabis[1]

Artificial neural networks are remarkably adept at sensory processing, sequence learning and reinforcement learning, but are limited in their ability to represent variables and data structures and to store data over long timescales, owing to the lack of an external memory. Here we introduce a machine learning model called a differentiable neural computer (DNC), which consists of a neural network that can read from and write to an external memory matrix, analogous to the random-access memory in a conventional computer. Like a conventional computer, it can use its memory to represent and manipulate complex data structures, but, like a neural network, it can learn to do so from data. When trained with supervised learning, we demonstrate that a DNC can successfully answer synthetic questions designed to emulate reasoning and inference problems in natural language. We show that it can learn tasks such as finding the shortest path between specified points and inferring the missing links in randomly generated graphs, and then generalize these tasks to specific graphs such as transport networks and family trees. When trained with reinforcement learning, a DNC can complete a moving blocks puzzle in which changing goals are specified by sequences of symbols. Taken together, our results demonstrate that DNCs have the capacity to solve complex, structured tasks that are inaccessible to neural networks without external read–write memory.

炼数成金逆向收费式网络课程

- Dataguru（炼数成金）是专业数据分析网站，提供教育，媒体，内容，社区，出版，数据分析业务等服务。我们的课程采用新兴的互联网教育形式，独创地发展了逆向收费式网络培训课程模式。既继承传统教育重学习氛围，重竞争压力的特点，同时又发挥互联网的威力打破时空限制，把天南地北志同道合的朋友组织在一起交流学习，使到原先孤立的学习个体组合成有组织的探索力量。并且把原先动辄成千上万的学习成本，直线下降至百元范围，造福大众。我们的目标是：低成本传播高价值知识，构架中国第一的网上知识流转阵地。
- 关于逆向收费式网络的详情，请看我们的培训网站 http://edu.dataguru.cn

Thanks

FAQ时间