Metadata of the chapter that will be visualized in SpringerLink

Book Title	Multimedia and Network Information Systems	
Series Title		
Chapter Title	Weight Adaptation Stability of Linear and Higher-Order Neural Units for Prediction Applications	
Copyright Year	2019	
Copyright HolderName	Springer Nature Switzerland AG	
Corresponding Author	Family Name	**Rodriguez-Jorge**
	Particle	
	Given Name	**Ricardo**
	Prefix	
	Suffix	
	Role	
	Division	Institute of Engineering and Technology
	Organization	Universidad Autónoma de Ciudad Juárez
	Address	Partido Romero, 32310, Ciudad Juárez, Mexico
	Email	ricardo.jorge@uacj.mx
Author	Family Name	**Bila**
	Particle	
	Given Name	**Jiri**
	Prefix	
	Suffix	
	Role	
	Division	
	Organization	Czech Technical University in Prague
	Address	Technická 4, 166 07 Praha 6, Prague, Czech Republic
	Email	
Author	Family Name	**Mizera-Pietraszko**
	Particle	
	Given Name	**Jolanta**
	Prefix	
	Suffix	
	Role	
	Division	Institute of Mathematics and Computer Science
	Organization	Opole University
	Address	Opole, Poland
	Email	
Author	Family Name	**Martínez-Garcia**
	Particle	
	Given Name	**Edgar A.**
	Prefix	
	Suffix	

Role	
Division	Institute of Engineering and Technology
Organization	Universidad Autónoma de Ciudad Juárez
Address	Partido Romero, 32310, Ciudad Juárez, Mexico
Email	

Abstract	This paper is focused on weight adaptation stability analysis of static and dynamic neural units for prediction applications. The aim of this paper is to provide verifiable conditions in which the weight system is stable during sample-by-sample adaptation. The paper presents a novel approach toward stability of linear and higher-order neural units. A study of utilization of linear and higher-order neural units with the foundations on stability of the gradient descent algorithm for static and dynamic models is addressed.
Keywords (separated by '-')	Linear neural units - Higher-order neural units - Stability analysis

Weight Adaptation Stability of Linear and Higher-Order Neural Units for Prediction Applications

Ricardo Rodriguez-Jorge[1](✉), Jiri Bila[2], Jolanta Mizera-Pietraszko[3], and Edgar A. Martínez-Garcia[1]

[1] Institute of Engineering and Technology, Universidad Autónoma de Ciudad Juárez, Partido Romero, 32310 Ciudad Juárez, Mexico
ricardo.jorge@uacj.mx

[2] Czech Technical University in Prague, Technická 4, 166 07 Praha 6, Prague, Czech Republic

[3] Institute of Mathematics and Computer Science, Opole University, Opole, Poland

Abstract. This paper is focused on weight adaptation stability analysis of static and dynamic neural units for prediction applications. The aim of this paper is to provide verifiable conditions in which the weight system is stable during sample-by-sample adaptation. The paper presents a novel approach toward stability of linear and higher-order neural units. A study of utilization of linear and higher-order neural units with the foundations on stability of the gradient descent algorithm for static and dynamic models is addressed. AQ1 AQ2

Keywords: Linear neural units · Higher-order neural units · Stability analysis

1 Introduction

Artificial neural networks were developed in the 1940's. Dealing with nonlinearities and uncertainties has been of major interest to artificial neural networks (ANNs). Due to their inherent approximation capabilities, the ANN has been extensible used in modeling of high complex nonlinear dynamic systems. These architectures can approximate spatio-temporal data by providing to the input of the network past inputs and past outputs. However, ANNs have been naturally studied with some problems about local minima, overfitting issue, and generalization capability [1–4].

One of the main applications of the ANNs is in the prediction field. Prediction has been considered as great interest to know what is unknown in the future. These ANNs have been applied to predict time series (a set of values of an attribute sensed in regular periods). Some applications of prediction by neural networks are: predicting the weather in a period, behavior of the merchandise sale level or maintaining the stock, financial reasons prediction, prediction catastrophic events into electrocardiograms, prediction of breathing patterns, and so on [5, 6].

Measurements of real-world systems usually have influences of several external processes, which can generate long-range correlations and non-stationarity.

K. Choroś et al. (Eds.): MISSI 2018, AISC 833, pp. 1–9, 2019.
https://doi.org/10.1007/978-3-319-98678-4_51

A non-stationary time series is usually characterized by time varying variance, mean, or even both. Thus, the learning process should then capture the current data behavior (dynamics) to the predictive model for improved prediction accuracy [7, 8].

Quadratic neural unit (QNU) and cubic neural unit (CNU) are known as type of higher-order neural unit (HONU), and comparing with linear neural unit (LNU), QNU and CNU provide good quality of nonlinear approximation. Basic concepts of learning and adaptation have been reviewed in [9].

This paper presents the sample-by-sample adaptation of linear, and higher-order neural units in their static and dynamic architectures, with error back-propagation and explores the gradient descent method. The implementation of linear and higher-order neural units is performed using a novel weight adaptation stability of the predictive models.

This paper is organized as follows: in Sect. 2, the static neural models are described, as well as the gradient descent method for linear and higher-order static neural models in sample-by-sample adaptation. Section 3 discusses the dynamic neural models for prediction. Section 4 discusses the weight system stability of the linear and higher order neural models. Section 5 discusses the results obtained after applying the weight update stability of the models. Finally, conclusion remarks and future work are described in Sect. 6.

2 Static Neural Models for Prediction

2.1 Linear Neural Unit

Linear neural units are simplest and computationally efficient, especially when the input data contain relatively higher number of inputs. LNUs yields the neural output as shown in Eq. (1).

$$\tilde{y}(k+h) = \sum_{i=0}^{n} w_i \cdot x_i = \mathbf{w} \cdot \mathbf{x}(k) \tag{1}$$

where, $\mathbf{w}$ denotes the row vector of all neural weights and $\mathbf{x}(k)$ represents the vector of inputs. The weight updates are directly calculated for a linear predictive model as presented next.

$$\Delta w_i(k) = -\frac{1}{2} \cdot \mu \cdot \frac{\partial e^2(k+h)}{\partial w_i} \tag{2}$$

where μ is the learning rate, $e(k+h)$ is the prediction error, and $\Delta w_i(k)$ is an adaptive weight increment of $i - th$ weight. The updates of all weights can be in their simplest form, i.e., without momentum or regularization term, as shown in Eq. (3).

$$w_i(k+1) = w_i(k) + \Delta w_i(k) \tag{3}$$

2.2 Quadratic Neural Unit

QNU is known as a type of HONU [2], and comparing with the traditional neural networks, QNU provide good quality of nonlinear approximation [6]. In addition, the mathematical structure is relatively comprehensible due to the minimum number of neural parameters. Basic concepts of learning and adaptation has been reviewed in [8]. The static quadratic neural unit used for prediction is shown in Fig. 1.

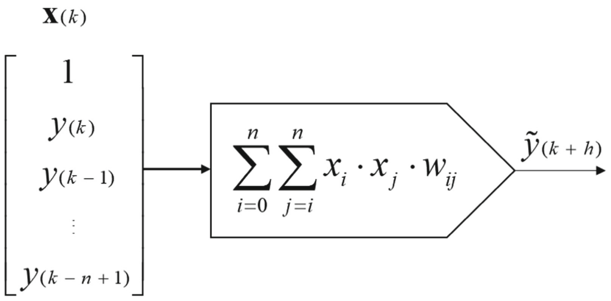

Fig. 1. Static quadratic neural unit for prediction.

In the case of the QNU static architecture, a quadratic operation using an $(1+n) \times (1+n)$ upper triangular weight matrix **W** is performed. This operation is presented in Eq. (4).

$$\tilde{y}(k+h) = \sum_{i=0}^{n}\sum_{j=i}^{n} x_i \cdot x_j \cdot w_{ij} = \mathbf{x}^{\mathbf{T}} \cdot \mathbf{W} \cdot \mathbf{x} \tag{4}$$

where the vector **x** denotes the $(1+n) \times 1$ vector with bias $x_0 = 1$, x_i denotes individual inputs (scalars), and h stands for the prediction horizon.

The static QNU is adapted every $k-th$ sample by means of sample-by-sample adaptation as presented in Eq. (5).

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}(k) \tag{5}$$

where the weight update is performed by gradient descent as shown in Eq. (6).

$$\Delta w_{ij}(k) = -\frac{1}{2} \cdot \mu \cdot \frac{\partial e^2(k+h)}{\partial w_{ij}} \tag{6}$$

where μ is the learning rate, and $e(k+h)$ stands for the error, which is defined as in Eq. (7).

$$e(k+h) = y(k+h) - \tilde{y}(k+h) \tag{7}$$

According to the sample-by-sample adaptation $\tilde{y}(k+h)$ stands for the neural output, $y(k+h)$ is the real value.

2.3 Cubic Neural Unit

A CNU is a type of HONU, the input-output relation of a static CNU is given as in Eq. (8).

$$\tilde{y}(k+h) = \sum_{i=0}^{n}\sum_{j=i}^{n}\sum_{l=j}^{n} x_i \cdot x_j \cdot x_l \cdot w_{i,j,l} \tag{8}$$

where $\mathbf{x}$ and $\tilde{y}(k+h)$ are the input vector and the neuron output, respectively; $\mathbf{W} = \{w_{i,j,l} : i,j,l \in n\}$ are the weights of the CNU. The input $\mathbf{x}(k)$ is defined by Eq. (9).

$$\mathbf{x}(k) = [\,1 \quad y(k) \quad y(k-1) \quad \cdots \quad y(k-n+1)\,]^T \tag{9}$$

where n is the total number of samples of the real signal $\mathbf{y}(k)$ and ^{T} stands for transposition. The CNU calculates the output neuron $\tilde{y}(k+h)$ when the input vector $\mathbf{x}$ is provided. The CNU learns by applying the back-propagation technique as a learning rule to adjust the weight matrix $\mathbf{W}$. The structure of the CNU for prediction is shown in Fig. 2.

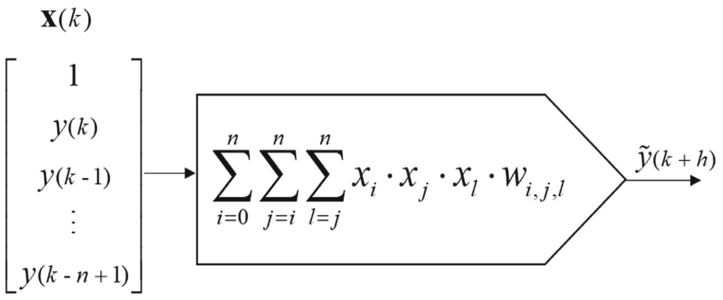
$\mathbf{x}(k)$
1
$y(k)$
$y(k-1)$
$\vdots$
$y(k-n+1)$
$\sum_{i=0}^{n}\sum_{j=i}^{n}\sum_{l=j}^{n} x_i \cdot x_j \cdot x_l \cdot w_{i,j,l}$
$\tilde{y}(k+h)$

Fig. 2. Cubic neural unit architecture for prediction.

The back-propagation technique as a learning rule is applied in a sample-by-sample adaptation. It follows the convergence criterion of the training performance, which is given as the sum of squared error in each iteration time, for N training samples as shown in Eq. (10).

$$Q(epoch) = \sum_{k=1}^{N} e(k+h) \tag{10}$$

where $e(k+h) = y(k+h) - \tilde{y}(k+h)$.

The neural weights adaptation $w_{i,j,l}(k+1)$ in the new adaptation time of the model is presented in Eq. (11).

$$w_{i,j,l}(k+1) = w_{i,j,l}(k) + \Delta w_{i,j,l}(k) \tag{11}$$

The increment of each individual weight with the gradient descent rule is described in Eq. (12).

$$\Delta w_{i,j,l}(k) = \mu \cdot e(k+h) \cdot \frac{\partial \tilde{y}(k+h)}{\partial w_{i,j,l}} \tag{12}$$

3 Dynamic Neural Models for Prediction

Towards the goal of predicting data from a system with the biologic neural characteristics, a mathematical model named dynamic linear neural unit (D-LNU), dynamic quadratic neural unit (D-QNU), and dynamic cubic neural unit (D-CNU) have been developed [1, 2]. The D-QNU and D-CNU are considered a type of dynamic higher-order neural unit (D-HONU). D-HONU have been capable of processing systems that present linearities and non-linearities on spaces of continuous or discrete time. The QNU model is considered as a special class of polynomial neural networks. The works of [1, 2, 6] have presented QNUs indicating the characteristics of the neural unit of providing a faster response comparing to a controller by linear states feedbacks as well as with controllers applied to nonlinear systems, unstable systems, and unknown non-linear dynamic systems.

3.1 Dynamic Quadratic Neural Unit

The D-QNU use a RTRL learning method and can be implemented in discrete or continuous real-time. In Eq. (13) the notation of D-QNU with RTRL is shown, where $\tilde{y}(k+h)$ is the predicted neural output and ^{T} represents the transpose of the vector $\mathbf{x}$, which contains the feedback neural output values and the signal from the system.

$$\tilde{y}(k+h) = \left[\sum_{i=0}^{nx-1} \sum_{j=0}^{nx-1} x_j \cdot w_{i,j} \cdot x_i \right] = \left(\mathbf{x}^T \cdot \mathbf{W} \cdot \mathbf{x} \right) \tag{13}$$

The upper triangular matrix of weights $\mathbf{W}$ with neural bias $w_{0,0}$ is defined as

$$\mathbf{W} = \begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,nx-1} \\ 0 & w_{1,1} & \cdots & w_{1,nx-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & w_{nx-1,nx-1} \end{bmatrix} \tag{14}$$

Equation (15) is the column of the augmented vector $\mathbf{x}$ of the neural input, $y(k)$ is the real value, n is the number of real values that feeds the neural input, k is the variable that describes the discrete time and h is the prediction horizon.

$$\mathbf{x} = \begin{bmatrix} 1 \\ \tilde{y}(k+h-1) \\ \tilde{y}(k+h-2) \\ \vdots \\ \tilde{y}(k+1) \\ y(k) \\ y(k-1) \\ \vdots \\ y(k-n+1) \end{bmatrix} \tag{15}$$

In the architecture of the D-QNU, $1/z$ represents the values of feedback, which are fed to the augmented vector $\mathbf{x}$.

An error function is defined in such a way that, in each learning step indicates how close the solution is to minimize the error function (mean squared error function) of the D-QNU model; $\frac{1}{N}\sum_{k=1}^{N} e(k+h)^2$ in each prediction time $k+h$. In Eq. (16), $w_{i,j}(k+1)$ stands for neural weights adaptation in the new adaptation time of the model, which are the sum of the neural weights increment $\Delta w_{i,j}(k)$ from the adaptation of the model, with individual weights $w_{i,j}$ of each k discretized value from the system.

$$w_{i,j}(k+1) = w_{i,j}(k) + \Delta w_{i,j}(k) \tag{16}$$

The increment of the neural weights with RTRL is described in (17), where μ is the learning rate that determines in which proportion the neural weights are updated and the velocity of the learning process, $e(k+h)$ is the error in each discrete time $k+h$ that multiplies the partial derivative of the neural output with respect to the weights $w_{i,j}$ described in Eq. (18).

$$\Delta w_{i,j}(k) = \mu \cdot e(k+h) \cdot \frac{\partial \tilde{y}(k+h)}{\partial w_{i,j}} \tag{17}$$

$$\begin{aligned} &\frac{\partial \tilde{y}(k+h)}{\partial w_{i,j}} = \\ &\frac{\partial(\mathbf{x}^T \cdot \mathbf{W} \cdot \mathbf{x})}{\partial w_{i,j}} = \left(\frac{\partial \mathbf{x}^T}{\partial w_{i,j}} \cdot \mathbf{W} \cdot \mathbf{x} + \mathbf{x}^T \cdot \frac{\partial \mathbf{W}}{\partial w_{i,j}} \cdot \mathbf{x} + \mathbf{x}^T \cdot \mathbf{W} \cdot \frac{\partial \mathbf{x}}{\partial w_{i,j}}\right) \end{aligned} \tag{18}$$

In Eq. (19), $\mathbf{jxw}_{i,j}$ is described as the partial derivative of the augmented vector $\mathbf{x}$ of the neural inputs with respect to the neural weights.

$$\frac{\partial \mathbf{x}}{\partial w_{i,j}} = \mathbf{jxw}_{i,j} = \\ \left[0 \; \frac{\partial \tilde{y}(k+h-1)}{\partial w_{i,j}} \; \frac{\partial \tilde{y}(k+h-2)}{\partial w_{i,j}} \; \cdots \; \frac{\partial \tilde{y}(k+1)}{\partial w_{i,j}} \; 0 \; \cdots \; 0 \right]^T \tag{19}$$

Meanwhile, in Eq. (20) the representation of a full matrix is shown, which is a convenient adaptive representation of a D-QNU, where $\mathbf{r}\Delta\mathbf{W}(k)$ is a vector with all increased weights and $\mathbf{jxw}_{i,j}$ is the Jacobian matrix evolving recurrently [9].

$$\mathbf{r}\Delta\mathbf{W}(k) = \mu \cdot e(k+h) \cdot \\ \left(\left(\mathbf{jxw}_{i,j}(k)^T \cdot \mathbf{W} \cdot \mathbf{x}\right)^T + \mathbf{r}\{\mathbf{x}^T \cdot \mathbf{x}\} + \mathbf{x}^T \cdot \mathbf{W} \cdot \mathbf{jxw}_{i,j}(k) \right) \tag{20}$$

where $\mathbf{r}\{\mathbf{x}^T \cdot \mathbf{x}\}$ represents all the combination of the input $\mathbf{x}$. The Jacobian matrix is updated on a recurrent basis for the adaptation of the model.

4 Weight Adaptation Stability

The weight system to be influenced during the weight adaptation stability can be expressed in matrix form for the static neural model, even linear, quadratic or cubic neural unit, and the learning rates will be used individually for each weight as shown in Eq. (21).

$$\mathbf{colW}(k+1) = \mathbf{colW}(k) + \mathbf{M} \cdot e(k+h) \cdot \mathbf{colx} \tag{21}$$

where $\mathbf{M}$ is $(nw \times nw)$ main diagonal matrix of individual learning rates. Accordingly, the individual learning rates corresponds to the main diagonal of the matrix $\mathbf{M}$ while the other values remain in zero.

Let us express the weight adaptation system using the introduced notation by substituting the definition of the error, as shown in Eq. (22).

$$\mathbf{colW}(k+1) = \mathbf{colW}(k) + \mathbf{M} \cdot (y(k+h) - \tilde{y}(k+h)) \cdot \mathbf{colx} \tag{22}$$

After several hand-work step-by-step, and considering the weight state vector $\mathbf{colW}$, the representation of the weight dynamics system can be rewritten as shown in Eq. (23).

$$\mathbf{colW}(k+1) = \mathbf{colW}(k)(\mathbf{1} - \mathbf{M} \cdot \mathbf{colx} \cdot \mathbf{rowx}) + \mathbf{M} \cdot y(k+h) \cdot \mathbf{colx} \tag{23}$$

where **1** denotes $(nw \times nw)$ identity matrix and $\mathbf{A} = (\mathbf{1} - \mathbf{M} \cdot \mathbf{colx} \cdot \mathbf{rowx})$. Next, the stability criterion is applied using Eq. (24).

$$Max|eig(\mathbf{A})| \leq 1 \tag{24}$$

Then, the weight system is stable (contractive) at time *epoch* if Eq. (24) is true. However, when the stability condition (non-expansiveness) of weight system is not satisfied, the learning rates are recalculated by random vector $\boldsymbol{\delta}$ of $(1 \times nw)$ dimension and the vector of learning rates are slightly modified as shown in Eq. (25).

$$\boldsymbol{\mu} = \boldsymbol{\mu} \cdot \left(1 - \frac{\boldsymbol{\delta}}{5}\right) \tag{25}$$

This is performed to maintain the condition (24). The division of the random vector rate $\boldsymbol{\delta}$ by, for example: the number 5, allows maintaining the learning rate very close to the original values. Next, after recalculating the new learning rates, the maximum absolute eigenvalues are evaluated again according to the condition Eq. (24). In case that the new learning rate matrix influence the stability condition, i.e. Eq. (24) is satisfied, the neural unit is updated. Otherwise, the learning rate matrix **M** is changed again until at least non-expanding autonomous part of weight dynamic update system is obtained.

For the dynamic neural model (D-LNU, D-QNU or even D-CNU) the weight update stability can be given using the previous expressions by fundamental gradient descent in RTRL.

5 Discussion

According to the adaptation of each individual weight as shown in the linear and higher-order models, all weights might appear as linear, therefore the adaptation weights can be transformed into a state-space form. Therefore, the vector **colW** has been introduced to perform the weight adaptation stability; which corresponds to the transformation of the weight matrix **W**, for higher-order neural units, or the vector **w** for linear neural units, i.e., **colW** contains all the weights ordered from every row of the upper-triangular weight matrix, for the higher-order neural units. In the case of **colx**, stands for a column vector of input multiplications, for the case of HONUs, and **rowx** stands for $\mathbf{colx}^T$.

6 Conclusions

This paper presented the linear and higher order neural units mathematical notations and their architecture as well. In addition, their stability evaluation approach has been introduced to ensure the stability of the weight update system of static and dynamic neural models applied in prediction applications. The approach of the stability analysis presented is based on the eigenvalues of state-space representation of the weight update dynamic system for gradient descent adaptation rule.

Acknowledgements. The project is supported by a research grant No. DSA/103.5/16/10473 awarded by PRODEP and by Autonomous University of Ciudad Juarez. Title - Detection of Cardiac Arrhythmia Patterns through Adaptive Analysis.

References

1. Rodríguez Jorge, R.: Lung tumor motion prediction by neural networks. Ph.D. thesis. Czech Technical University in Prague, Czech Republic (2012)
2. Gupta, M.M., Jin, L., Homma, N.: Static and Dynamic Neural Networks. Wiley, U.S.A (2003)
3. Bach, F.: Breaking the curse of dimensionality with convex neural networks. J. Mach. Learn. Res. **18**, 1–53 (2017)
4. Rodriguez, R., Vergara Villegas, O.O., Cruz Sanchez, V.G., Bila, J., Mexicano, A.: Arrhythmia disease classification using a higher-order neural unit. In: 2015 Fourth International Conference on Future Generation Communication Technology (FGCT), pp. 1–6. IEEE, July 2015
5. Rodríguez, R., Mexicano, A., Bila, J., Ponce, R., Cervantes, S., Martinez, A.: Hilbert transform and neural networks for identification and modeling of ECG complex. In: 2013 Third International Conference on Innovative Computing Technology (INTECH), pp. 327–332. IEEE, August 2013
6. Cancino, E., Rodriguez Jorge, R., Vergara Villegas, O.O., Cruz Sánchez, V.G., Bila, J., Nandayapa, M., Israel, P., Soto, A., Abad, A.: Monitoring of cardiac arrhythmia patterns by adaptive analysis. In: The 11th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC-2016) (2016)
7. Mironovova, M., Bíla, J.: Fast fourier transform for feature extraction and neural network for classification of electrocardiogram signals. In: 2015 Fourth International Conference on Future Generation Communication Technology (FGCT), pp. 1–6. IEEE, July 2015
8. Rodriguez, R., Bukovsky, I., Bila, J.: Weight Adaptation Stability of Static Quadratic Neural Unit. Ústav Přístrojové a řídicí techniky, pp. 83–88. Czech Technical University in Prague (2011)
9. Rodríguez Jorge, R., Martínez García, E., Mizera-Pietraszko, J., Bila, J., Torres Córdoba, R.: Prediction of highly non-stationary time series using higher-order neural units. In: Xhafa, F., Caballé, S., Barolli, L. (eds.) Advances on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2017. Lecture Notes on Data Engineering and Communications Technologies, vol. 13. Springer, Cham (2017)

Author Query Form

Book ID :	**466725_1_En**
Chapter No :	**51**

Springer
the language of science

Please ensure you fill out your response to the queries raised below and return this form along with your corrections.

Dear Author,
During the process of typesetting your chapter, the following queries have arisen. Please check your typeset proof carefully against the queries listed below and mark the necessary changes either directly on the proof/online grid or in the 'Author's response' area provided below

Query Refs.	Details Required	Author's Response
AQ1	This is to inform you that corresponding author has been identified as per the information available in the Copyright form.	
AQ2	Per Springer style, both city and country names must be present in the affiliations. Accordingly, we have inserted the city names "Ciudad Juárez and Prague" in affiliations "1 and 2". Please check and confirm if the inserted city names "Ciudad Juárez and Prague" are correct. If not, please provide us with the correct city names.	

MARKED PROOF

Please correct and return this set

Please use the proof correction marks shown below for all alterations and corrections. If you wish to return your proof by fax you should ensure that all amendments are written clearly in dark ink and are made well within the page margins.

Instruction to printer	*Textual mark*	*Marginal mark*
Leave unchanged	• • • under matter to remain	ⓥ
Insert in text the matter indicated in the margin	⋏	New matter followed by ⋏ or ⋏ⓧ2
Delete	/ through single character, rule or underline or ⊢—⊣ through all characters to be deleted	₰ or ₰ⓧ2
Substitute character or substitute part of one or more word(s)	/ through letter or ⊢—⊣ through characters	new character / or new characters /
Change to italics	— under matter to be changed	⌴
Change to capitals	≡ under matter to be changed	≡
Change to small capitals	= under matter to be changed	=
Change to bold type	～ under matter to be changed	～
Change to bold italic	$\overline{～}$ under matter to be changed	$\overline{～}$
Change to lower case	Encircle matter to be changed	⧺
Change italic to upright type	(As above)	⊥
Change bold to non-bold type	(As above)	⊥
Insert 'superior' character	/ through character or ⋏ where required	Y or ⋎ under character e.g. $\overset{2}{Y}$ or $\overset{2}{⋎}$
Insert 'inferior' character	(As above)	⋏ over character e.g. $\underset{2}{⋏}$
Insert full stop	(As above)	⊙
Insert comma	(As above)	,
Insert single quotation marks	(As above)	$\overset{‘}{Y}$ or $\overset{‘}{⋎}$ and/or $\overset{’}{Y}$ or $\overset{’}{⋎}$
Insert double quotation marks	(As above)	$\overset{“}{Y}$ or $\overset{“}{⋎}$ and/or $\overset{”}{Y}$ or $\overset{”}{⋎}$
Insert hyphen	(As above)	⊢⊣
Start new paragraph	⎍	⎍
No new paragraph	↩	↩
Transpose	⊔⊓	⊔⊓
Close up	linking ⁐ characters	⁐
Insert or substitute space between characters or words	/ through character or ⋏ where required	Y
Reduce space between characters or words	| between characters or words affected	↑