Mathematics Bachelor

Bachelor Thesis

Recurrent Neural Networks for time series forecasting in complex dynamical systems and their applications

Arturo Fredes Cáceres

supervised by

Dr. Sergio Gutiérrez Rodrigo

César Arbiol Herrera

2022-2023

Contents

1 Introduction

Artificial intelligence and machine learning techniques are currently a very hot topic, and recent advances in the field have raised hopes, doubts and fear in society. These techniques are useful for tackling several types of problems like classification, regression, generation of content or forecasting. The focus of this work was set on forecasting time series, which plays an important role in different areas such as business, weather or economics providing insights into future trends or outcomes.

2 Objectives and methodology

The objective was to a study how recurrent neural networks perform when forecasting future steps of time series, by working with different sets of data of increasing complexity. A proof of concept was made working with one-dimensional data of the sum of two waves and some noise. Different types of neurons and architectures were used to compare performance in different scenarios. Next, the study was continued using data from a 3D chaotic system (the Lorenz Attractor), and finally it was attempted to apply these techniques to real business data.

The code of this work was written in Python and the Keras library was used to build the models. Keras is an open-source deep learning library that serves as an interface to build and train neural networks in a user-friendly way. Google Colab notebooks were used to run the code because of the free GPU runtime provided, which made compilation faster. When longer runtimes and more RAM where needed a laptop with 8 GB of RAM and an intel Core i5 CPU was used. The code, trained models and generated data can be found in the GitHub repository [3].

3 Neural Networks

Neural networks constitute one of the most popular branches of AI due to their capacity of finding trends and patterns in data and generalizing them to new data. These networks consist in webs of artificial neurons distributed in different layers through which information flows to process an output.

3.1 Artificial Neurons

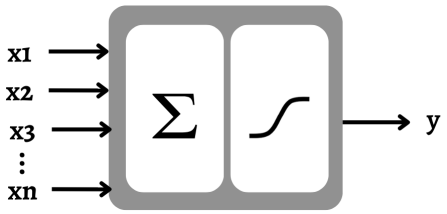


Figure 1: *Scheme of an artificial neuron. The product of an input vector x and a weight vector is evaluated by an activation function to produce an output y*

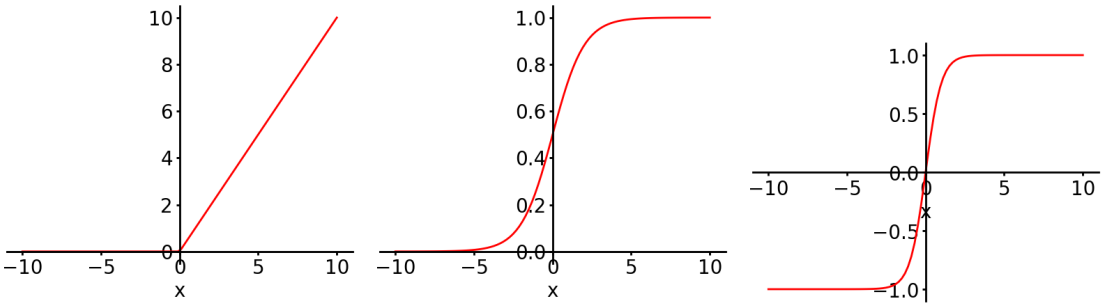
The first step will be introducing the inner workings of a simple artificial neuron . These neurons are inspired by biological neurons, and they learn through weighted connections. Usually, a neuron receives an input vector, which is multiplied by a weight vector. Weights are trainable parameters, which give a sense of importance of the input for our output.

Neurons are activated if the input signal is more intense than a certain threshold. This is translated to the artificial neuron with the use of an activation function. The simplest approach is the use of a step function, which dictates if the neuron is 'on' or 'off', but other activation functions can be used to determine the 'intensity' of the output. The resulting output of the neuron will be something like Equation 1, where x is the input vector, w is the weight vector, b is the bias and Φ is the activation function.

$$y = \Phi(\sum_{i=1}^{n} w_i \cdot x_i \ + \ b) \tag{1}$$

Some of the most common activation functions are the Rectified Linear Unit (ReLU), the Logistic sigmoid and the Hyperbolic Tangent, which can be seen in Equations 2.

$$ReLU(x) = max(x, 0) \quad ; \quad \sigma(x) = \frac{1}{1 \ + \ e^{-x}} \quad ; \quad tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2}$$



(a) (b) (c)

Figure 2: *Some of the most common activation functions: (a) ReLU (b) Logistic sigmoid (c) Hyperbolic tangent*

After explaining how the network generates output, it must be explained how it 'learns' to generate the desired output. A training data set is used to 'teach' the network how to compute the outputs, or to be more precise, to adjust its weights and biases. The training data set must consist of K input data elements x^k and the desired outputs for these elements, y^k. The neuron has to learn from its hits and misses, by reinforcing connections when the output is similar to the desired one and changing them when it is not. For each element, the neuron's output $\hat{y}^k$ (the prediction) and the desired output y^k are compared by a function $C^k(y^k, \hat{y}^k)$. The loss function is defined as the sum of these, and since the training data set is fixed, it will be a function of the weights and biases:

$$C(w, b) = \sum_{i=1}^{K} C_k(y^k, \hat{y}^k) = \sum_{i=1}^{K} C^k(\Phi(x \cdot w + b), \hat{y}^k) \tag{3}$$

By minimizing this loss function, the optimum weights and biases will be obtained. Some examples of loss functions that will be used along this work are the Mean Squared Error (MSE) and the Mean Absolute Error (MAE) seen in equation 4.

$$MSE(y,\hat{y}) = \frac{1}{K}\sum_{i=1}^{K}(\hat{y}^k - y^k)^2 \quad ; \quad MAE(y,\hat{y}) = \frac{1}{K}\sum_{i=1}^{K}|\hat{y}^k - y^k| \tag{4}$$

To minimize the loss function, the gradient descent method is used. The gradient of the loss function is computed, and it indicates the direction of the steepest increase. The negative gradient will point in the direction of the steepest decrease. Therefore, weights and biases will be recalculated as in Equation 5.

$$w_i = w_i - \alpha\frac{\partial C}{\partial w_i} \quad ; \quad b_i = b_i - \alpha\frac{\partial C}{\partial b_i} \tag{5}$$

The solution will be approached by carrying out this process iteratively. The hyper parameter α is called learning rate, and it must be chosen carefully for the training to converge.

3.2 Multilayer Neural Networks

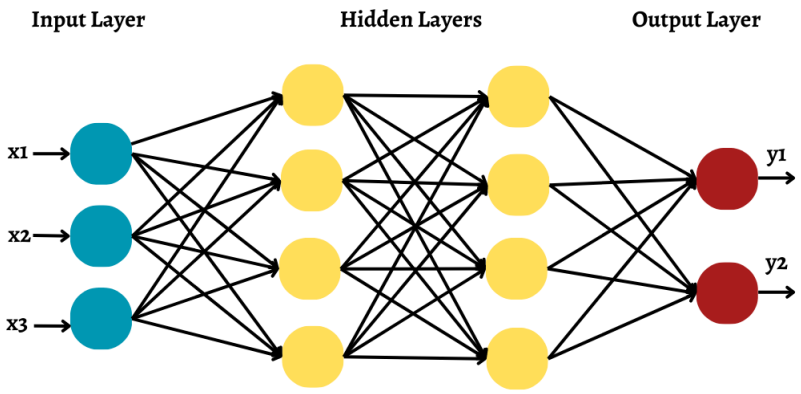


Figure 3: *Scheme of a Multilayer Neural Network*

Single neurons are not able to learn complex patterns, and it has been proven that they can only be used to classify separable problems. Nevertheless, when organized and stacked into multilayer networks, neurons become really powerful. This webs are called deep neural networks and they can represent any function if enough parameters are used. Deep neural networks have an input and an output layer which are responsible for taking data into the network and carrying it out. The intermediate layers are called hidden layers. The input of the neurons of one layer will be the output of the neurons of the previous one, so information will 'flow from left to right'. The number of neurons in each layer will be another important hyper parameter affecting the performance of the model.

When using deep neural networks optimization of the loss function becomes very costly due to the large number of parameters. The flow of information being only in one direction enables the calculation of derivatives using the chain rule. This 'trick' reduces computation time drastically when finding the optimum weights and is called backpropagation. Another

technique used to reduce computational cost is the stochastic gradient descent. It consist in taking a random sample of the training data set of big enough size so it will probably point in a direction similar to the one obtained when using the whole set. The Adaptative Moment Estimation (Adam) optimizer [2] is a variation of the stochastic gradient descent, and it will be the one used throughout this work.

4 Working with Time Series

As it was mentioned earlier, the task in hand is working with time series, and using neural networks to forecast future values of the series. Time series are sequences of data which take up different values through time. The structure of these data sets are usually tensors with dimensions (number of steps, number of features). This means that for each series we will make observations of some variables (features), for example, temperature, humidity and atmospheric pressure. These observations are taken periodically a specific number of times (number of steps), for example once a day during ten days. For the previous example, data would have the following structure: (10,3).

4.1 Recurrent Neural Networks

Feedforward neural networks can be used to work with time series but they are not the best candidate due to their lack of 'memory'. When a time series is fed to these type of networks, they will take the whole series as a vector and then give a forecast. This way, it does not necessarily 'see' the relation between a time step and the previous one.

Recurrent neural networks not only have connections pointing forward, but also going backwards. For each time step of the data sequence, the network will take a vector of the sequence and the output of the previous iteration to give a new output. As the output is a result of all the previous time steps that have gone through the network, we can understand it as some kind of memory, although very limited. The outputs are calculated as we can see in Equation 6.

$$y(t) = \Phi(W_x^\intercal \cdot x(t) \ + \ W_y^\intercal \cdot y(t-1) \ + \ b) \tag{6}$$

In Figure 4 shows the diagram of a basic recurrent cell and how information flows through time.

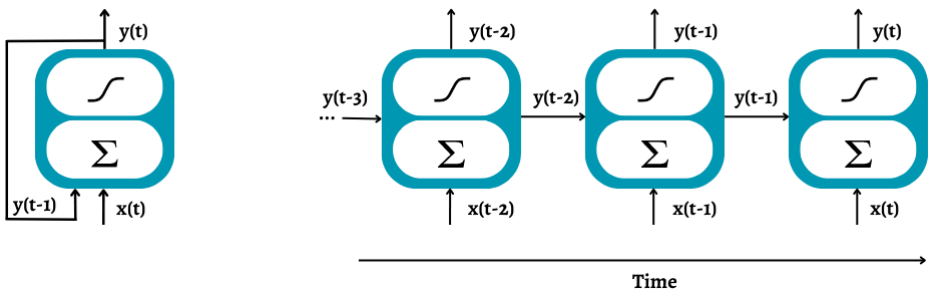


Figure 4: *Left: Scheme of a Recurrent cell. Right: Scheme of a recurrent cell unrolled through time.*

When the input of the cell is a sequence, two types of outputs can be given out. If we take into account the output of every time step like in the left of Figure 5, the resulting output is also a sequence. There are other occasions when only the last output is of interest, for example, when forecasting the next time step. In these cases, the rest of the outputs are ignored and the final result is a vector, like on the right of Figure 5. Each of the approaches gives different results when training the network as it will be observed in the following sections.

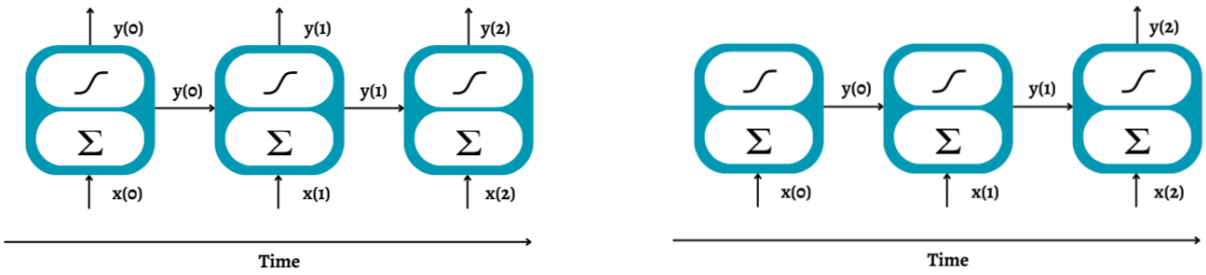


Figure 5: *On the left a sequence to sequence recurrent cell, and on the right a sequence to vector cell*

4.2 Long Short Term Memory Cells

Basic recurrent cells are better at keeping information about previous time steps than feedforward networks, but they struggle with detecting long term patterns in the data. Long Short Term Memory cells (LSTM) [5] were designed to tackle this problem and have an additional state vector c(t) which is a long-term state. Their training also converges faster than one of basic recurrent cells.

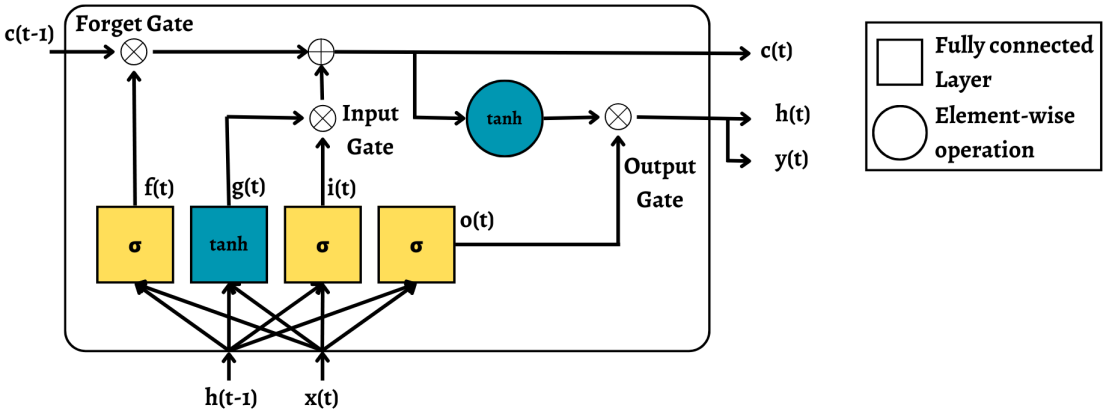


Figure 6: *Scheme of a LSTM cell*

In first place, the current input vector x(t) and the previous short-term state h(t–1) are fed to four different fully connected layers. Each of the layers plays a different role.

The main layer is the one that outputs g(t) seen in Equation 7. It analyses the current inputs x(t) and the previous short-term state h(t–1) using the hyperbolic tangent activation function. This layer works just like the basic cell, but now y(t) and h(t) are not directly outputted, and additional operations are made to take into account the long-term state c(t), which will also be modified with information of the new input.

$$g(t) = tanh(W_{xg}^{\intercal} \cdot x(t) \ + \ W_{hg}^{\intercal} \cdot h(t-1) \ + \ b_g) \tag{7}$$

The other three layers use the logistic activation function, and they work as gates. Since their output varies from 0 to 1 and goes to element-wise multiplications, it 'opens' (1) or 'closes' (0) the gate.

The output of the first layer, f(t) seen in Equation 8, controls the 'forget' gate. In this gate some parts of the long-term sate c(t-1) are dropped and others continue to flow through the cell.

$$f(t) = \sigma(W_{xf}^{\intercal} \cdot x(t) \ + \ W_{hf}^{\intercal} \cdot h(t-1) \ + \ b_f) \tag{8}$$

The second of the layers outputs i(t) seen in Equation 9, which controls the input gate. This gate identifies which part of the input is important and updates the long-term sate by adding this new information to it.

$$i(t) = \sigma(W_{xi}^{\intercal} \cdot x(t) \ + \ W_{hi}^{\intercal} \cdot h(t-1) \ + \ b_i) \tag{9}$$

After this, the cell outputs the long-term state c(t) as in Equation 10, which by now has erased and added new memories from the process.

$$c(t) = f(t) \otimes c(t-1) \ + \ i(t) \otimes g(t) \tag{10}$$

A copy of the long-term sate is made, and the hyperbolic tangent is applied to it. Finally, the output gate which is controlled by o(t) (Equation 11) decides which parts of the information of this copy is outputted as h(t) and y(t), Equation 12.

$$o(t) = \sigma(W_{xo}^{\intercal} \cdot x(t) \ + \ W_{ho}^{\intercal} \cdot h(t-1) \ + \ b_o) \tag{11}$$

$$y(t) = h(t) = o(t) \otimes tanh(c(t)) \tag{12}$$

In a nutshell, LSTM cells are able to store important bits of data in its long term memory and use it when it is needed or delete it if it is not.

4.3 Gated Recurrent Unit (GRU)

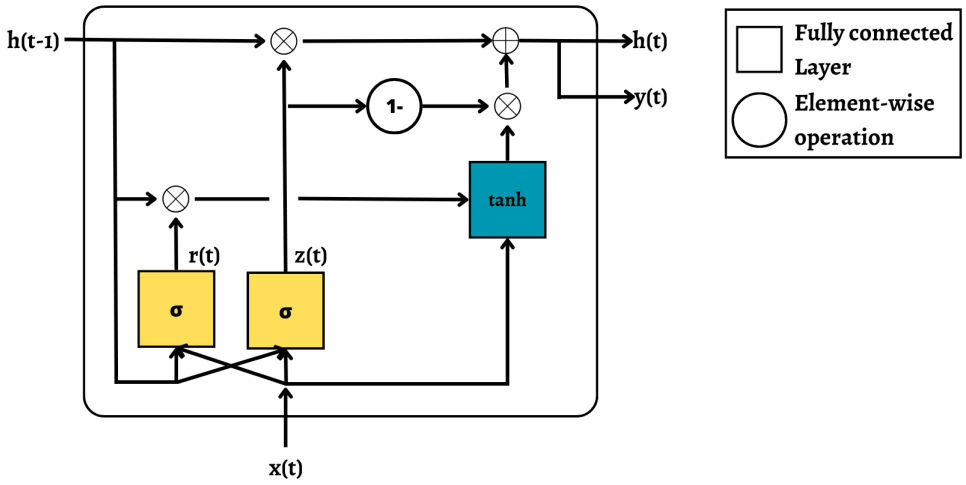


Figure 7: *Scheme of a GRU cell*

The Gated Recurrent Unit (GRU) [1] is a simplified version of the LSTM cell, and it has a similar performance. Like in the basic cell, we have a single sate-vector h(t), and now a single layer controls both the input and the forget gate using function z(t) seen in Equation 13.

$$z(t) = \sigma(W_{xz}^{\intercal} \cdot x(t) \ + \ W_{hz}^{\intercal} \cdot h(t-1) \ + \ b_z) \tag{13}$$

This is done by feeding the output z(t) to the forget gate and 1-z(t) to the input gate. Consequently, when one is closed, the other is open, forcing the cell to forget memories to be able to make new ones or ignore the input data to maintain the current ones.

There is an additional gate, which is controlled by r(t) seen in Equation 14, and it decides which part of the previous state h(t-1) can be seen by the main layer. Once again, the main layer is the one working with the tanh activation function, and it outputs g(t), Equation 15.

$$r(t) = \sigma(W_{xr}^{\intercal} \cdot x(t) \ + \ W_{hr}^{\intercal} \cdot h(t-1) \ + \ b_r) \tag{14}$$

$$g(t) = tanh(W_{xg}^{\intercal} \cdot x(t) \ + \ W_{hg}^{\intercal} \cdot (r(t) \otimes h(t-1)) \ + \ b_g) \tag{15}$$

Finally, y(t) and h(t) are the result of adding what comes out from the input and forget gate as seen in Equation 16

$$y(t) = h(t) = z(t) \otimes h(t-1) \ + \ (1 - z(t)) \otimes g(t) \tag{16}$$

5 Toy Model

5.1 Data

Firstly, we will work with a simple one-dimensional data to illustrate how the different techniques can improve our predictions as seen in [2]. The data used will be the sum of two waves with different amplitude, frequency and phase ,and some noise. For each sequence, four random numbers between 0 and 1: $w_1,\ w_2,\ \phi_1\ and\ \phi_2$ were generated, and for each timesteps some noise was added, which was a random value in [-0.05,0.05]. The time step selected was $\Delta t = \frac{1}{n_{steps}}$.

$$y(t) = \overbrace{0.5 \cdot sin[(t-\phi_1) \cdot (10\omega_1 + 10)]}^{\text{Wave 1}} + \overbrace{0.2 \cdot sin[(t-\phi_2) \cdot (20\omega_2 + 20)]}^{\text{Wave 2}} + noise \tag{17}$$

10.000 sequences of 50 steps were generated, and later divided into 7000 sequences for training, 2000 for validation and 1000 for testing.

5.2 Models

To begin, some baselines to beat should be defined. The data is made up by the sum of continuous functions and some noise which is an order of magnitude smaller. Therefore, it would be a good guess to think that in the next step, the series will take a value close to the last one. The first 'naive' baseline defined is to assume that the next value will be the same as the last one.

Next, each one of the different cells mentioned in sections 1 and 2 were trained through 20 epochs, with the training data set of 7000 sequencers. The MSE loss function, and the Adam optimizer were chosen for training. The results of the predictions of each cell on the test set with respect to their targets, in terms of MSE, are collected in Table 1

	Naive	Feedforward	Basic Recursive	LSTM	GRU
Parameters	-	51	3	12	12
Test MSE	0.0199	0.0041	0.0109	0.0111	0.0109
Improvement (resp. naive)	0%	79.4%	45.2%	44.2%	45.2%

Table 1: *MSE of predictions of the different types of cells (feedforward, simple recurrent, LSTM and GRU with respect to their targets). Training was carried out though 20 epochs with MSE loss function and Adam optimizer. The 'naive' model consists in predicting the previous step, and the improvement with respect to this baseline is also shown.*

It can be observed that the best performing model is the feedforward neuron with flattened input, although it loses the temporal component of the data. In this case, the recursive cells have a reduced number of trainable parameters, and therefore limited learning capacity, whilst the feedforward neuron has significantly more, making it perform better. It can also be seen that the three recursive cell give similar results, this could be because the data consists of short series, and long-term memory does not play an important role.

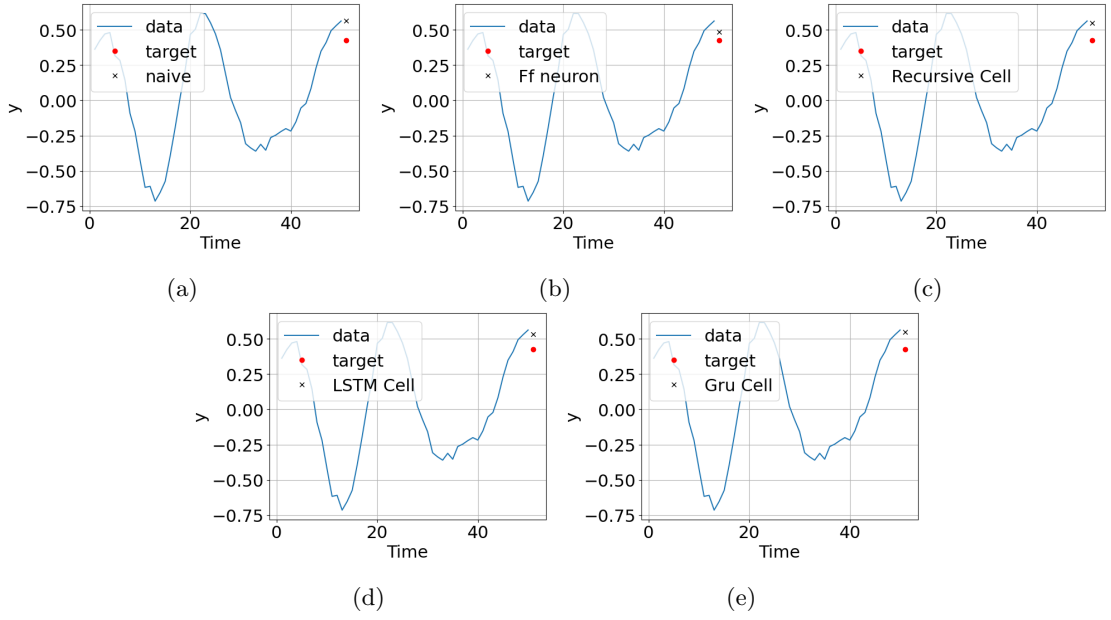


Figure 8

Figure 9: *Example sequence, target, and prediction made by different type of cells. Each cell was trained with a dataset of 7000 sequences of 50 steps through 20 epochs, using MSE loss function and Adam optimizer. (a) Naive baseline, forecasting the previous step (b) feedforward neuron (c) Simple recurrent cell (d) LSTM cell (e) GRU cell*

To see if the recursive neurons are of some use, deep neural networks with the same architecture but using the different types of cells were trained. The networks had the following architectures:

1. A layer of 20 neurons
2. A second layer of 20 neurons
3. A dense output layer of 1 neuron and ReLU activation function

All networks were trained for 20 epochs using the Adam optimizer. The results can be seen in Table 2.

	Naive	Feedforward	Basic Recursive	LSTM	GRU
Parameters	-	1461	1281	5061	3921
Test MSE	0.0199	0.0033	0.0027	0.0026	0.0030
Improvement (resp. naive)	0%	83.4%	86.4%	86.9%	84.9%

Table 2: *MSE of predictions of neural networks with the same architecture but different types of cells (feedforward, simple recurrent, LSTM and GRU) with respect to their targets. Training was carried out though 20 epochs with MSE loss function and Adam optimizer. The 'naive' model consists in predicting the previous step, and the improvement with respect to this baseline is also shown.*

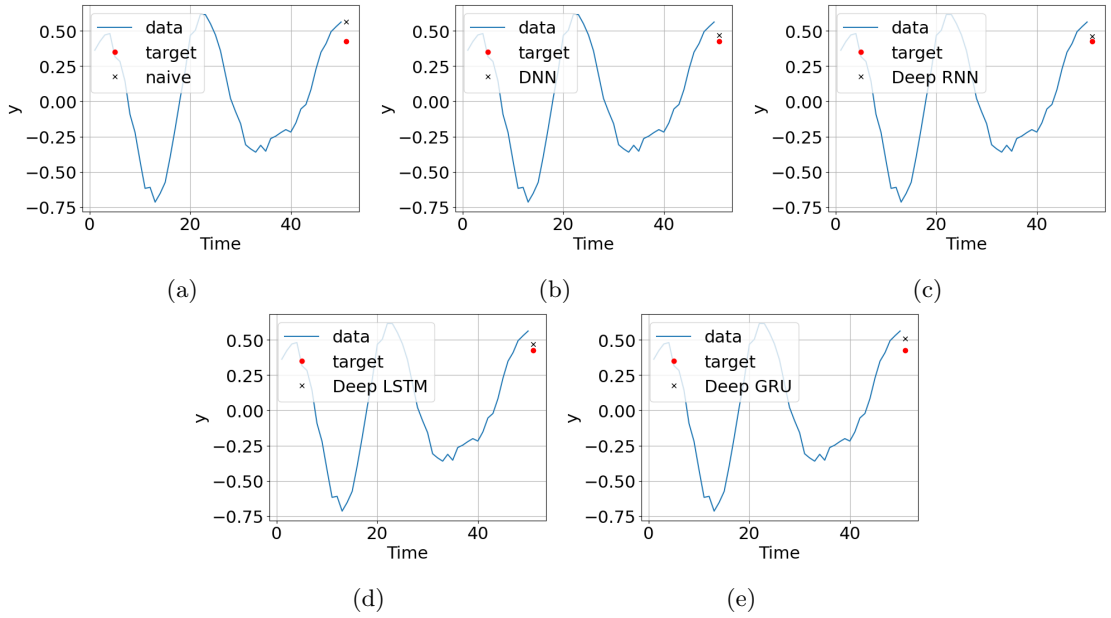


Figure 10: *Example sequence, target, and prediction made by networks with the same architecture, two layers of 20 cells and an output layer, using different type of cells. Each network was trained with a dataset of 7000 sequences of 50 steps through 20 epochs, using MSE loss function and Adam optimizer. (a) Naive baseline, forecasting the previous step (b) feedforward neurons (c) Simple recurrent cells (d) LSTM cells (e) GRU cells*

5.3 Predicting Several Time Steps

The next challenge is to predict several timesteps into the future. A study on the effectiveness of three different techniques will be made using the LSTM network, which was the best performing one.

5.3.1 Iteration

The first method consisted in predicting a single timestep with the previous models, adding the prediction to the timeseries and making new predictions with the new timeseries that started from step 2 and ended in step 51. After 10 iterations we obtained the 10 next steps. With this approach the results seen in Table 3 were obtained.

Average MSE	Last time step MSE
0.0183	0.0363

Table 3: *MSE of predictions of the next 10 time steps and of the 60th step over the test data set. Predictions were done predicting each one of the following steps iteratively using the LSTM model mentioned in the previous section.*

5.3.2 Sequence to Vector

As it will be seen, the results from the previuos method are not the best, since predictions accumulate error through the iterations and after each one, the target is further away. The next method consisted in changing the targets to be sequences of the desired length of the predictions. After this, the network was trained to predict the following steps all at once, finding relationships between the input and output sequences. To do so we had to change the last layer of our model:

~~3. A dense output layer of 1 neuron and ReLU activation function~~

3. A dense output layer of 10 neurons and ReLU activation function Results can be seen in Table 4.

Average MSE	Last time step MSE
0.0102	0.0193

Table 4: *MSE of predictions of the next 10 time steps and of the 60th step over the test data set. Predictions were done using a sequence-to-vector LSTM model, with the same layers as mentioned in the previous section and changing the output layer.*

5.3.3 Sequence to sequence

Finally, the previous approach was taken a step further and a sequence-to-sequence network was constructed. Now, the output after each timestep will be saved using a TimeDistributed dense neuron in the output layer:

~~3. A dense output layer of 1 neuron and ReLU activation function~~

3. A TimeDistributed dense output layer of 10 neurons and ReLU activation function

When training, the network will give an output of the m following steps at each step and compare it with the real m following steps. Therefore, the targets will now be collections of sequences of the form (number of steps, m). This way, the network learns to replicate the sequence at every step. This could appear not to be a great practice because there is a big overlap between data and targets. On the other hand, if we focus on each time step, there is no overlap since the network only 'sees' the previous steps and the targets contains the following

m steps. Results obtained are shown in Table 5

Last Series Average MSE	Last time step MSE
0.0028	0.0042

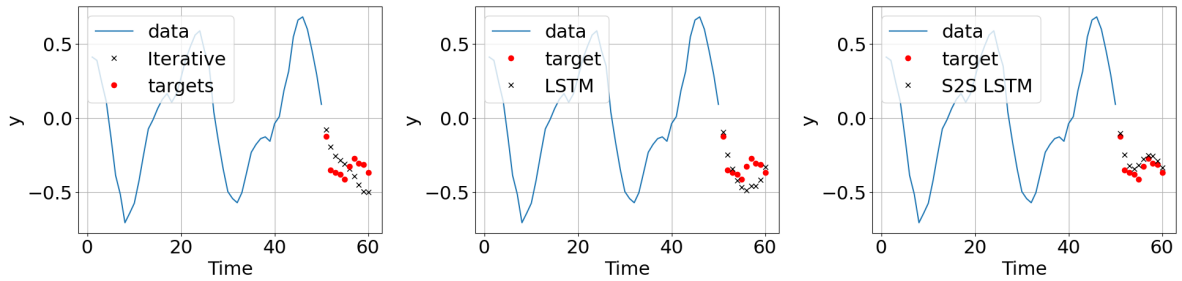
Table 5: *MSE of predictions of the next 10 time steps and of the 60th step over the test data set. Predictions were done using a sequence-to-sequence LSTM model, with the same layers as mentioned in the previous section but changing the output layer.*

In Table 6 it can be observed that predictions improve by a 44.4% compared to the iterative method when using targets of longer length. The predictions made by the sequence-to-sequence model improve even more, a 84.7%. Although the computational cost is higher due to the size of the targets, using these techniques brings significant improvements in performance.

		Sequence Average MSE	Last step MSE
Iteration	Test	0.0183	0.0363
(5061 params.)	Improvement	-	-
S-V	Test	0.0102	0,0193
(5250 params.)	Improvement	44.4%	46.8%
S-S	Test	0,0028	0,0042
(5250 params.)	Improvement	84.7%	88.4%

Table 6: *Comparison of accuracy of predictions of the next 10 time steps and the 60th step of a 50 step sequence using three different methods and a model of LSTM cells. The model consisted in two layers of 20 LSTM cells and the output layer changed to match the desired output. The first method was an iterative method of predicting the next time step. The second method consisted in using a sequence-to-vector network to output the next ten steps at once. Finally, a sequence-to-sequence model was used and the next 10 steps for every point of the series were calculated.*

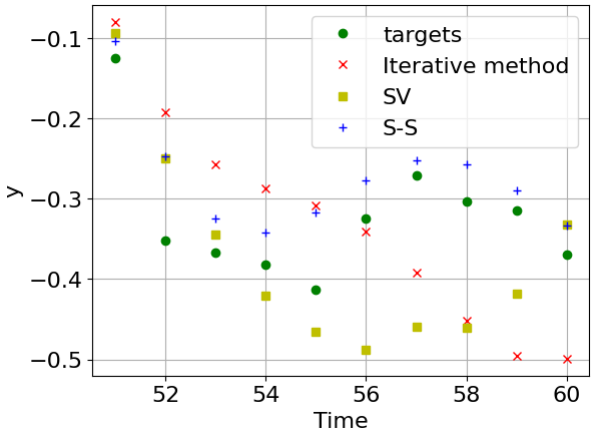
In figure 11 the graphs of the data and predictions of each of the three models and how they compare are shown. In this example it appears clear that there is a gradual improvement with each of the new techniques introduced.



(a) *Iteration*
MSE = 0.0146

(b) *Sequence to Vector*
MSE = 0.0114

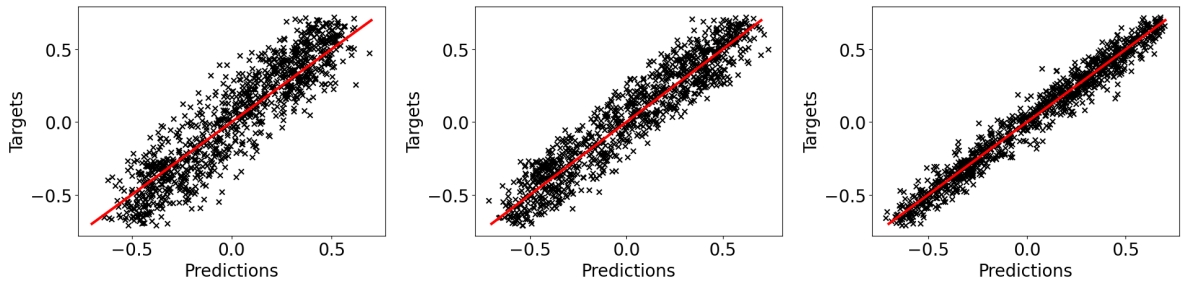
(c) *Sequence to Sequence*
MSE = 0.0030



(d) *Comparison of predictions and targets*

Figure 11: *Example sequence, target, and prediction made by networks with the same two layers of 20 LSTM cells but different output layers and targets. Each network was trained with a dataset of 7000 sequences of 50 steps through 20 epochs, using MSE loss function and Adam optimizer. (a) Iterative method: the networks outputs the next step iteratively adding steps to the sequence (b) Sequence to vector network: now the output consists of the next 10 steps. (c) Sequence to sequence network: for each timestep of the sequence we forecast the next 10 steps (d) Comparison of the 3 methods with the real sequence*

A visual way to see how predictions improve is plotting targets against predictions. If predictions were perfectly accurate, they would overlap with the straight line of slope 1 that passes through the origin of coordinates. In Figure 12 the last step predicted was plotted along with its target value, and it can be clearly observed how the sequence-to-sequence model's results are the closest to this line, followed by the sequence-to-vector model, while the iterative method is the one were points are the most disperse.



(a) (b) (c)

Figure 12: *Values of the last time step predicted plotted along with its target values. Predictions were made over the test data set using three different methods to predict the next 10 steps. (a)Iterative method (b)Sequnce-to-vector (c)Sequence-to-sequence*

Finally, the same study was made with the other two types of recursive cells to see if the improvement with these techniques were also noticeable. Looking at the results of Table 7 it can be concluded that they do.

GRU		Last sequence Average MSE	Last step MSE
Iteration	Test	0.0463	0.0917
(3921 params.)	Improvement	-	-
S-V	Test	0.0072	0.0149
(4110 params.)	Improvement	84.4%	83.7%
S-S	Test	0.0029	0.0045
(4110 params.)	Improvement	93.6%	95.1%
Recurrent Neurons		Last sequence Average MSE	Last step MSE
Iteration	Test	0,0221	0,0406
(1281 params.)	Improvement	-	-
S-V	Test	0.0099	0.0198
(1470 params.)	Improvement	55.0%	51.3%
S-S	Test	0.0081	0.0132
(1470 params.)	Improvement	63.3%	67.5%

Table 7: *Comparison of accuracy of predictions of the next 10 time steps and the 60th step of a 50 step sequence using three different methods and models of GRU cells and simple recurrent neurons. The models consisted in two layers of 20 cells and the output layer changed to match the desired output. The first method was an iterative method of predicting the next time step. The second method consisted in using a sequence-to-vector network to output the next ten steps at once. Finally, a sequence-to-sequence model was used and the next 10 steps for every point of the series were calculated.*

6 Working with chaotic systems

With the last example, how recursive networks can improve forecasting when working with timeseries respect from feedforward networks was seen. However, these networks have not yet exploited their full potential. Now, data of a chaotic system[6], the Lorenz System, will be generated and used to explore the capabilities of the technique for its use in mathematical

modelling.

6.1 The Lorenz System

In 1963 Edward Lorenz presented a mathematical model describing atmospheric convection[4] . The model consisted in a three dimensional system of ordinary differential equations as seen in Equation 18

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases} \tag{18}$$

In the equations, x represents the fluid flow rate, y represents the horizontal temperature gradient and z the vertical temperature gradient. Describing the dynamics of a two-dimensional fluid layer uniformly cooled from above and warmed from below.

What makes this system interesting is that it presents chaotic solutions. These solutions receive the name of Lorenz Attractor, and they describe aperiodic, bounded, non-linear trajectories with a very high sensitivity to changes in initial conditions. Unlike other classical systems where trajectories end up in orbits, fixed points or diverge, the solutions of the Lorenz system stay bounded without repeating the same path, tracing a shape that resembles the wings of a butterfly as we can see in Figures 13 and 15. Furthermore, high sensitivity to initial conditions means that very small differences in initial conditions can lead to completely different outcomes after some time, making the system both deterministic and highly unpredictable.

This phenomenon is popularly called the 'butterfly effect' and it makes forecasting a difficult challenge in these types of systems. That is why it is of great interest to test the predictive models with data of the Lorenz Attractor.

6.2 Data

5000 3D sequences of 1000 steps were generated. The first 900 steps of each sequence where used as input data and the next 100 steps were used as targets. The parameters took values $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$ as chosen by Lorenz in his paper, this way we obtained chaotic solutions. Initial conditions x_0, y_0, and z_0 were random numbers chosen from the [-1,1] interval. Data was generated using the fourth order Runge-Kutta method with $\Delta t = 0.01$. 3500 elements were used for training, 1000 for validation and 500 for testing.

6.3 Models

As done in the previous section, networks with the same architecture but different types of neurons were trained. The chosen architecture was:

1. Layer of 64 neurons
2. A second layer of 64 neurons

3. A dense layer of 3 Neurons and ReLU activation function

All the networks were trained for 20 epochs using MSE as loss functions and the Adam optimizer. Results of training are shown in Table 8

	Naive	Feedforward	Basic Recursive	LSTM	GRU
Parameters	-	177219	12812	50627	38403
Test MSE	1.5986	1.6067	0.0136	0.0005	0.0008
Improvement (resp. naive)	-	-0.51%	99.15%	99.97%	99.95%

Table 8: *MSE of neural networks with the same architecture but different types of cells after training for 20 epochs with MSE loss function and Adam optimizer.*

The improvement of MSE in the test data is very significant now with respect to the feedforward network, which does not beat the baseline. Another important observation is that the gated cells have a MSE two orders of magnitude smaller than the basic recursive cell. This is probably due to the effect of having a 'long-term memory' and working with longer sequences.

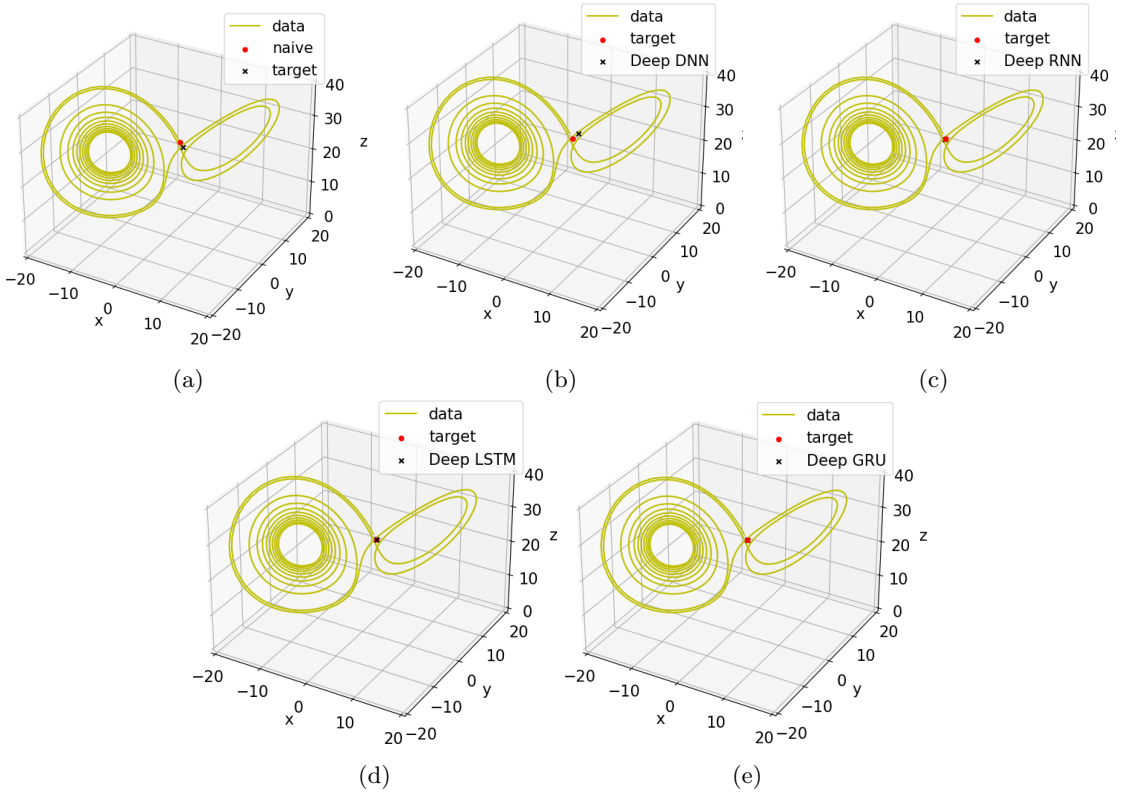


Figure 13: *Example sequence, target, and prediction made by networks with the same architecture, two layers of 64 cells and an output layer, using different type of cells. Each network was trained with a dataset of 3500 sequences of 900 steps through 20 epochs, using MSE loss function and Adam optimizer. (a) Naive baseline, forecasting the previous step (b) feedforward neurons (c) Simple recurrent cells (d) LSTM cells (e) GRU cells*

6.4 Predicting Several Time Steps

Next, the study of the forecast of several timesteps into the future using different methods was repeated with the new data set. This time the first 900 timesteps were used as data and the next 100 steps were forecasted. Again, the output layer had to be replaced when using the sequence to vector and sequence to sequence approaches:

S-V: A dense output layer of 300 neurons and ReLU activation function

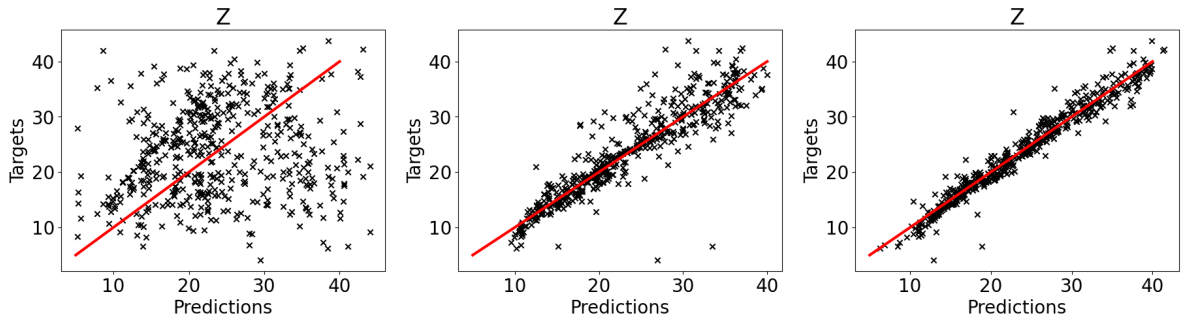
S-S: A TimeDistributed dense output layer of 300 neurons and ReLU activation function

The results can be seen in Table 9.

		Sequence MSE	Last step MSE
Iteration	Test	87.3004	111.3869
(50627 params.)	Improvement	-	-
S-V	Test	3.4737	8.1686
(69737 params.)	Improvement	96.02%	92.67%
S-S	Test	1.1842	2.2665
(69737 params.)	Improvement	98.64%	97.97%

Table 9: *Comparison Lorenz*

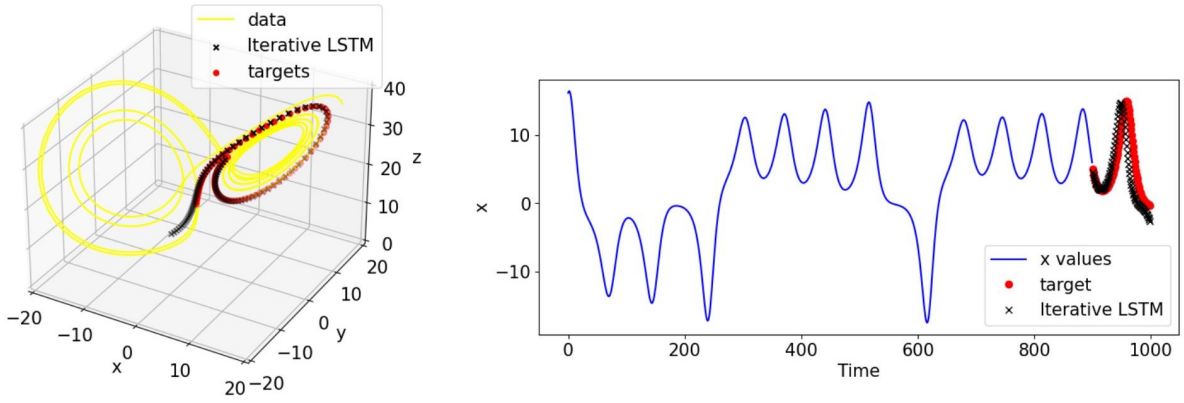
It can be observed that the iterative method is considerably worse than the other two. This may be an effect of working with a system which is very sensitive to small changes in initial conditions, so the accumulated error leads to bigger differences in trajectories. It must also be noted that the sequence-to-sequence model still outperforms the sequence to vector model. In Figure 14 target values of the Z-coordinate of the 100th step are plotted against the predicted values, the improvement in performance can be clearly appreciated in these graphs.



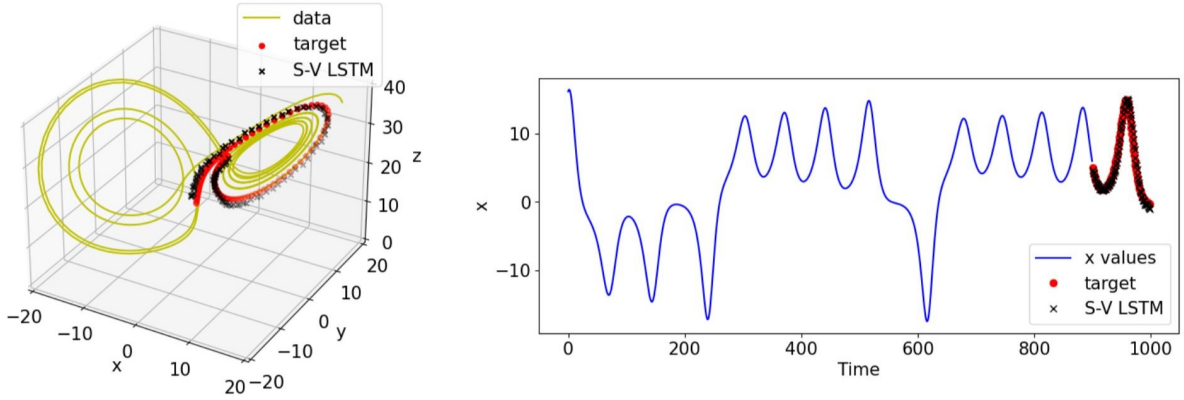
(a) (b) (c)

Figure 14: *Values of the Z-coordinate of the last time step predicted plotted along with its target values. Predictions were made over the test data set using three different methods to predict the next 100 steps. (a)Iterative method (b)Sequnce-to-vector (c)Sequence-to-sequence*

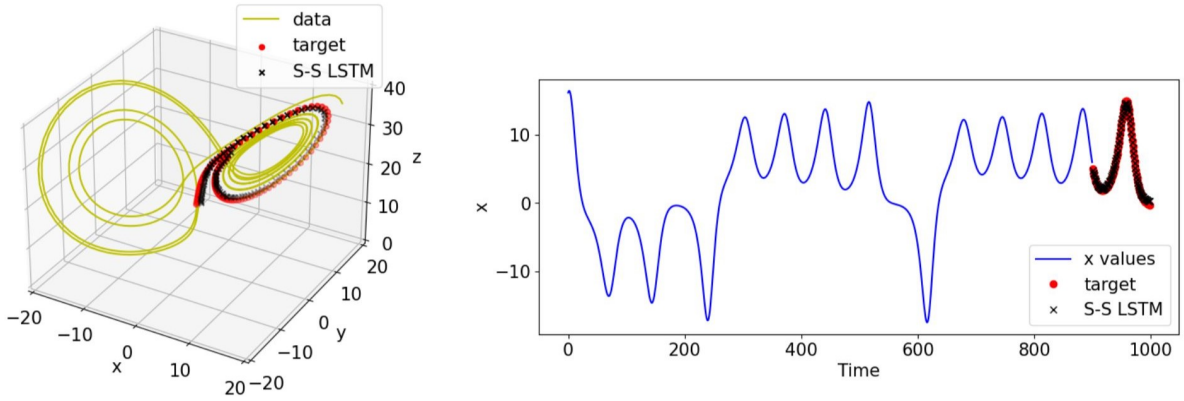
In Figure 15 an example of predictions using the three methods is illustrated.



(a) *Iteration ; MSE = 30.7558 ; Last step MSE = 18.2213*



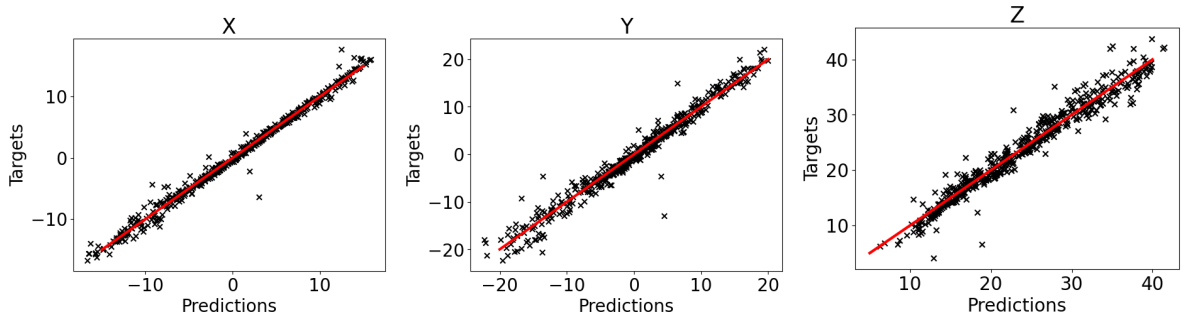
(b) *Sequence to Vector ; MSE = 1.4238 ; Last step MSE = 0.6661*



(c) *Sequence to Sequence ; MSE = 0.2845 ; Last step MSE = 0.2296*

Figure 15: *Example sequence , target, and prediction made by networks with the same two layers of 64 LSTM cells but different output layers and targets. Each network was trained with a dataset of 3500 sequences of 900 steps through 20 epochs, using MSE loss function and Adam optimizer. (a) Iterative method: the networks outputs the next step iteratively adding steps to the sequence (b) Sequence to vector network: now the output consists of the next 100 steps. (c) Sequence to sequence network: for each timestep of the sequence we forecast the next 100 steps*

To get a better idea of how good these predictions were, a further analysis of the results of our last model, which is the best performing, was made. The mean relative error gives a notion of how far of off the predictions were, and the results can be seen in Figure 16 were the target values of the last step were plotted against the predictions.



(a) *Rel. Error = 34%*
Rel. Error = 10%*

(b) *Rel. Error = 50%*
Rel. Error = 26%*

(c) *Rel. Error = 6%*
Rel. Error = 6%*

Figure 16: *Targets plotted against predictions for the three coordinates of the last time step of the predictions. Mean eelative errors of the predictions made over a test data set of 500 elements with a sequence to sequence LSTM neural network are included in the captions. *Target values in the interval [-0.1,0.1] excluded.*

It can be clearly seen that the points follow the tendency of the line with a slope of 1 that passes through the origin of coordinates. The last step misses by an average of 50% in the y coordinate, which is a considerable error. This is due to the target values in the denominator being close to zero and making the relative error bigger even if the miss was small in absolute values. To solve this, the relative error was calculated excluding elements with values in the interval [-0.1,0.1], leaving 98% of all the elements. It was observed that the relative error was reduced to 26%. This could be considered satisfactory or not depending on the accuracy requirements.

7 Forecasting demand with business data

In this section, focus will finally be shifted to real data, working in partnership with Editorial Edelvives. Edelvives is the oldest publishing house in Spain with more than 130 years of history. The company specializes in educational materials, as well as children's and young adult literature, offering a diverse range of educational resources, including textbooks, teaching materials and reading books. The latter category was the one of interest for this work, and forecasting the demand of literature books will be attempted.

The problem aimed to address is calculating the number of units of a book to be printed by forecasting the demand of the upcoming months. Currently, this task is carried out by a very experienced worker in the company who possesses extensive knowledge of the business aspect of this specific sector. The goal is to help her in this endeavor.

7.1 Data

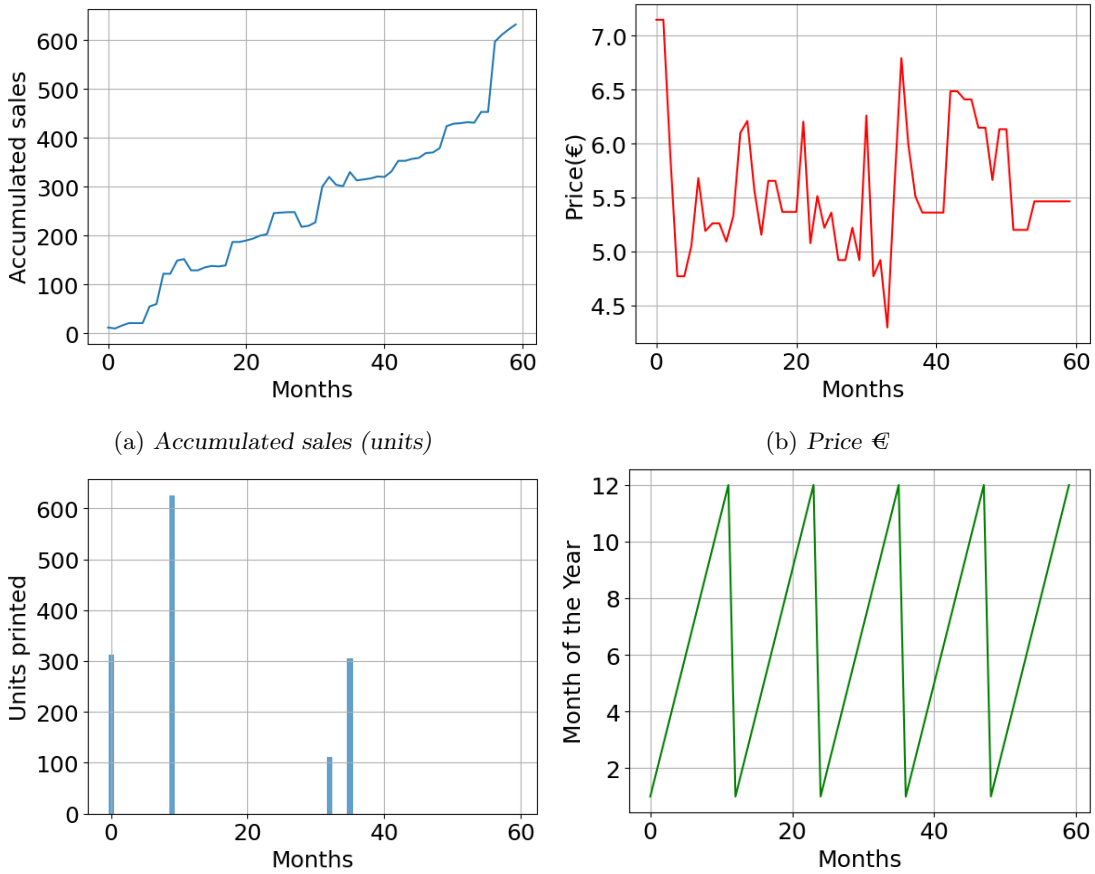
A monthly timescale for the time series was chosen, and the model will predict the demand of the following 6 months using information of the past year. Four features that fluctuate through time were selected:

- Accumulated sales (units): this is the feature to be forecasted. Accumulated sales were

chosen over sales because this way the function would have less jumps.

- Price (€): the price of an article fluctuates through time, and an increase or decrease in pricing can lead to more or less sales.
- Previous printings (units): Other units printed in the previous year were taken into account.
- Months of the year (1-12): The model will be indicated which months of the year the data presented belong to. The data has strong seasonality due to the Christmas and summer campaigns and the aim of this feature is to bare this in mind.

In Figure 17 An example of 5 years of each features of an article are shown.



(a) *Accumulated sales (units)*

(b) *Price €*

(c) *Units printed*

(d) *Month of the year*

Figure 17: *Representations of 5 years of each feature for a specific article.*

The data of 4081 articles through 5 years was transformed into all possible series of 12 months of data and 6 months of targets. The result was 171402 data elements which were divided into 97944 for training (3 years), 48972 for validation (1 year) and 4081 for testing (the last sequence of every article). In the end, sequences where the sales over the target 6 months were zero were discarded, leaving 56068 elements for training, 30311 for validation and 2827 for testing.

7.2 MC Dropout

A new technique was introduced for these predictions, the Montecarlo Dropout. Firstly, dropout layers were added after each of the recurrent layers. The effect of this, for every training step, is that each neuron has a probability p (dropout rate) of being turned 'off'. Therefore, the network has to work with only the neurons which are 'on' at each step, helping neurons to be more versatile and useful on their own. It also prevents the network from becoming too dependent on a few very specialized neurons and reduces overfitting.

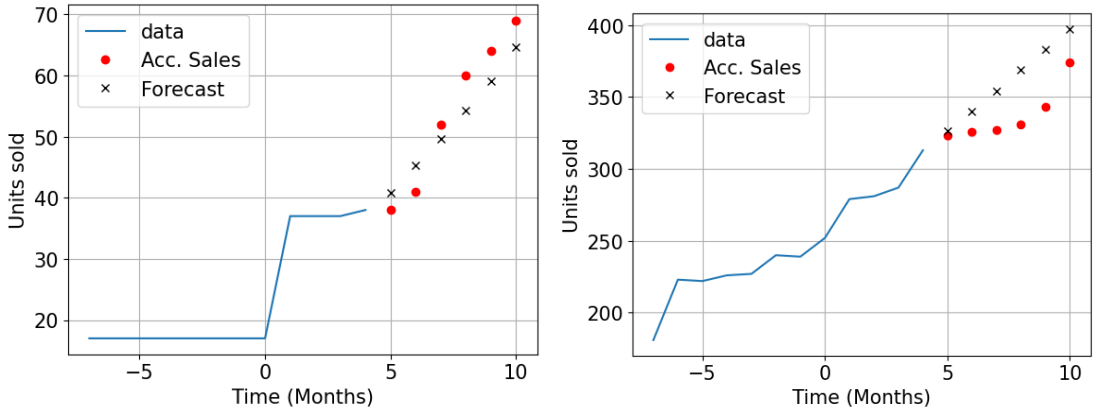
Another interpretation of this, is that if the network is composed of N neurons, at each step, a different network of the 2N possibilities is selected (each of the neurons can be on or off). This networks, although different, are not independent. We will take advantage of this interpretation to make our predictions. Usually, after training, the dropout layers are not activated, so outputs are always computed by the same neural network. The MC Dropout technique consists in leaving the dropout layers 'on' when doing predictions, and for each element of the data set, we will take the average of a large number of predictions.

7.3 Model

After trying several models trained with different amounts of neurons, loss functions and types of outputs, the following was selected:

1. A sequence to sequence LSTM layer of 32 neurons
2. A Dropout layer with a dropout rate of 0.2
3. A second sequence to sequence LSTM layer of 32 neurons
4. Another Dropout layer with a dropout rate of 0.2
5. A Timedistributed Dense layer of 6 neurons.

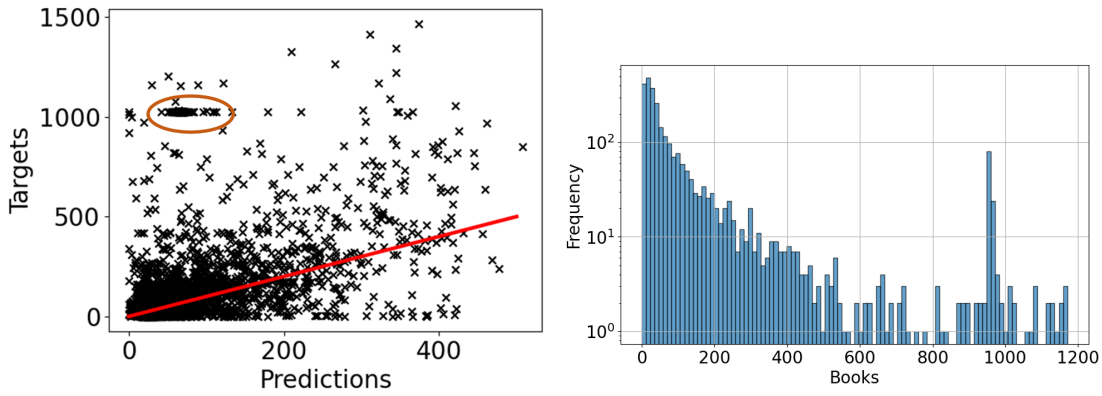
The model was trained for 50 epochs using Adam optimizer and the MSE loss function, and predictions on the test data set using the MC dropout technique. To have a sense of how good the predictions were, the absolute error was used, since now what must be known is how many books would have ended in a shelf of the warehouse or how short the stock would have been, resulting in a new reprint. The result obtained for the predictions of the test data set was a MAE of 160 books. Nevertheless, the model performs well on many elements, 2015 elements (71%) miss by less than 100 books, and 1597 (57%) by less than 50 books. Some examples are shown in Figure 18



(a) *Missed by 4 books* (b) *Missed by 23 books*

Figure 18: *Examples of sequences were our model made predictions correctly. Predictions were made using MC dropout and averaging over 100 predictions.*

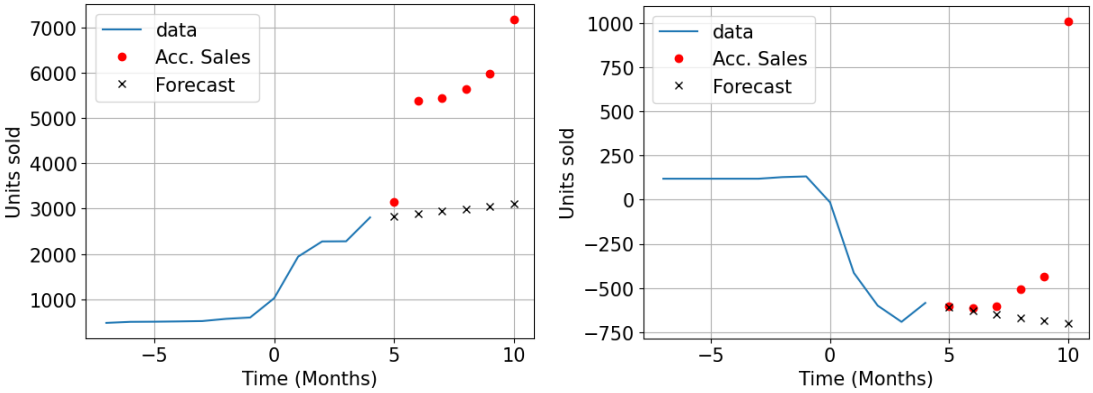
To further analyse the performance, predictions of total sales over the next 6 months were plotted against the actual sales over the next 6 months in Figure 19 (a). It can be observed that the linear tendency is not clear, although the cloud of points is denser near the line. A different zone with higher density of points was circled.



(a) (b)

Figure 19: *Predictions were obtained using MC dropout and averaging over 100 predictions. (a)predictions of total sales over the next 6 months plotted against the actual sales over the next 6 months (b)Distribution of absolute errors of the model over a test data set of 4081 articles. The distribution is presented in logarithmic scale, and we can clearly see two peaks, one near zero and one around 1000. The group of elements were predictions failed by around 1000 books had a common characteristic that we were not taking into account, they were part of special offers.*

It can be seen when the distribution of absolute errors is drawn (Figure 19b) that there is clearly a second peak near 1000 books of error. Under further examination, it was found out that this peak corresponded to articles which had a sudden peak in sales, as the examples in Figure 20. The model is capable of finding the trend in sales until the peak occurs. The reason behind this is that for some articles online promotions were made (gifting a book with the purchase of another one, 2x1 in a certain line of products...) which led to this steep increase in sales. To solve this problem, a fifth feature including the historical information on special offers is needed.



(a) *Missed by 4075 book* (b) *Missed by 1711 books*

Figure 20: *Example of data, predictions ,and actual sales of two articles. These articles belong to the group which had been part of special offers, and the predictions miss by more than 1000 books.*

If articles to the left of this peak are selected (error smaller than 900 books), 94% of the total articles, an average absolute error of 83 books is obtained, which is an improvement of 48 %. The next steps would be including more business knowledge into the model by interviewing the company workers.

8 Conclusions

Recurrent neural network appears to be a useful tool when forecasting time series. It was first studied how these techniques can beat the forecasts made by feedforward networks when working with time series using simple one-dimensional data as a proof of concept. It was also seen that, although having a higher computational cost, sequence-to-sequence and sequence-to-vector models bring a significant improvement in performance with respect to iterating one step predictions.

The study of applying these techniques to chaotic 3D sequences (Lorenz Attractor) was satisfactory, and it was possible to predict the 100th step into the future with the highest average error for a coordinate of 26%, which has been considered satisfactory. This time, the improvement in performance of recurrent neural networks with respect to their feedforward counterparts was more noticeable, the latter ones were not even able to beat the baseline. LSTM and GRU cells proved to be more useful working with longer series where the long-term memory played an important role. When working with predictions further into the future the difference in accuracy between the iterative method, sequence-to-vector and sequence-to-sequence models was also accentuated due to the chaotic nature of the sequences, making accumulated error bigger.

Finally, the study of its application to real data did not work as well due to a lack of business knowledge taken into account when constructing the model. The demand of literature books was predicted with an average absolute error of around 160 books, which was not considered good enough. A missing feature was discovered, and taking out books which belonged to special cases, reduced error to around 80 books. Overall, results were promising and improving this model will be left as future work.

Bibliography

[1] Kyunghyun Cho. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: (2014). DOI: 10.48550/arXiv.1406.1078.

[2] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, Second Edition*. O'Reilly Media, Inc., 2019. ISBN: 978-1-492-03264-9.

[3] *GitHub repository of the work*. URL: https://github.com/arturofredes/rnn-chaotic-timeseries-applications.git.

[4] Edward N. Lorenz. "Deterministic Nonperiodic Flow". In: *Journal of the Atmospheric Sciences, Volume 20* (1963). DOI: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[5] Jürgen Schmidhuber Sepp Hochreiter. "Long Short-Term Memory". In: *Neural Computation (1997) 9 (8): 1735–1780* (1997). DOI: 10.1162/neco.1997.9.8.1735.

[6] Elizabeth M. Cherry Shahrokh Shahi Flavio H. Fenton. "Prediction of chaotic time series using recurrent neural networks and reservoir computing techniques: A comparative study". In: *Machine Learning with Applications, Volume 8* (2022). DOI: 10.1016/j.mlwa.2022.100300.

9 Annex I: Resumen en español

El objetivo de este trabajo era realizar un estudio sobre cómo las redes neuronales recurrentes se desempeñan en la predicción de pasos futuros en series temporales, trabajando con diferentes conjuntos de datos de complejidad cada vez mayor. Se realizó una prueba de concepto trabajando con datos unidimensionales que consistían en la suma de dos ondas y algo de ruido. Diferentes tipos de neuronas y arquitecturas fueron utilizadas para comparar el rendimiento en diferentes escenarios. Posteriormente, se continuó el estudio utilizando datos de un sistema caótico tridimensional (el Atractor de Lorenz) y finalmente se intentó aplicar estas técnicas a datos de la empresa para realizar predicciones de demanda.

En la primera sección se introdujeron brevemente las redes neuronales y su funcionamiento en general. Posteriormente, se describe más en detalle el problema a estudiar, definiendo qué son la series temporales y explicando el funcionamiento de las redes neuronales recurrentes. Las series temporales son secuencias de datos que varían con el tiempo y se organizan generalmente como tensores con dimensiones (número de pasos, número de características). Las redes neuronales recurrentes, funcionan como las densas, pero a deferencia de ellas, en cada paso reciben nueva información y la salida de la red en la iteración anterior. La salida de la red es dependiente de todos los pasos anteriores, por lo que puede entenderse como una especie de memoria. Este tipo de redes puede producir dos tipos de salidas: una secuencia de salidas si se consideran todas las etapas temporales, o un vector final si solo interesa la última salida, por ejemplo, al predecir el próximo paso temporal. La elección del tipo de salida puede afectar al entrenamiento de la red y los resultados obtenidos, como se explica en secciones posteriores.

La memoria de las redes neuronales recurrentes simples es limitada, por lo que se introdujo una nueva unidad de procesamiento, la celda LSTM. Esta celda emplea un estado a largo plazo y es mejor a la hora de hacer predicciones con series largas. También se explica una versión simplificada de ella, las celdas GRU.

Una vez finalizada la introducción teórica, en primer lugar, se estudió cómo estas técnicas pueden superar las predicciones realizadas por las redes densas utilizando datos unidimensionales sencillos como prueba de concepto. Las series empleadas se regían por la Ecuación 17. Las neuronas densas se mostraron superiores funcionando solas debido al mayor número de parámetros entrenables, pero cuando se aumentaba el número de neuronas las redes recurrentes presentaban un mayor rendimiento. Una vez hecho esto, se pasó al estudio de diferentes técnicas para predecir varios pasos. Se observó que, aún teniendo un mayor costo computacional, los modelos de secuencia a secuencia y de secuencia a vector mejoraban significativamente el rendimiento en comparación con las predicciones iterativas de un solo paso.

Tras esta prueba de concepto, se utilizaron secuencias del atractor de Lorenz (Ecuaciones 18)generadas con el Runge-Kutta de orden 4 y parámetros $\sigma = 10$, $\rho = 28$ y $\beta = 8/3$ para obtener las soluciones caóticas. Estas soluciones no lineales, acotadas, aperiódicas y con alta sensibilidad a las condiciones iniciales suponían un reto considerable a la hora de hacer predicciones, permitiendo poner a prueba las técnicas. El estudio de la aplicación de estas técnicas a secuencias caóticas 3D fue satisfactorio, y fue posible predecir el paso futuro número 100 en el futuro con un error promedio más alto del 26% para una coordenada, lo cual se consideró satisfactorio. En esta ocasión, la mejora en el rendimiento de las redes neuronales recurrentes

en comparación con las densas fue más evidente; estas últimas ni siquiera pudieron superar el 'baseline'. Las celdas LSTM y GRU resultaron ser más útiles al trabajar con series más largas en las que la memoria a largo plazo desempeñaba un papel importante. Cuando se trabajó con predicciones más lejanas en el futuro, la diferencia en la precisión entre el método iterativo y los modelos de secuencia a vector y de secuencia a secuencia también se acentuó debido a la naturaleza caótica de las secuencias, lo que aumentó el error acumulado.

Finalmente, se emplearon datos de la Editorial Edelvives y se intentó predecir la demanda de libros de literatura teniendo en cuenta diferentes características de los artículos. El estudio de la aplicación a datos reales no funcionó tan bien debido a la falta de conocimiento empresarial implícito en la construcción del modelo. La demanda de libros de literatura se predijo con un error absoluto promedio de alrededor de 160 libros, lo que no se consideró lo suficientemente bueno. Se descubrió una característica que faltaba y, al eliminar los libros que pertenecían a casos especiales, se redujo el error a alrededor de 80 libros. En general, los resultados fueron prometedores y mejorar este modelo quedará como trabajo futuro.