LSTM RNNs For Long-Term Forecasting

- Methods for time series predictions -

Course: "Machine Learning con applicazioni"

Student: Luca Zilli, 983688

Professor: Carlo Barbieri

Università degli studi di Milano
Facoltà di scienze e tecnologie
Corso di laurea magistrale in Fisica
Year: 2020/2021

Project Description

The scope of this project is to improve the machine learning algorithm built for the thesis of my bachelor degree. My thesis is about time series predictions, for which I used a simple RNN cell with recurrent connection between hidden states and, for this project, I aim to improve the predictive power by using a more advanced Long-Short-Term-Memory (LSTM) cell. In the following pages, firstly I will review briefly the scope and results of my thesis, secondly I will explain the basic functioning of the LSTM and than I will show the results of this research project.

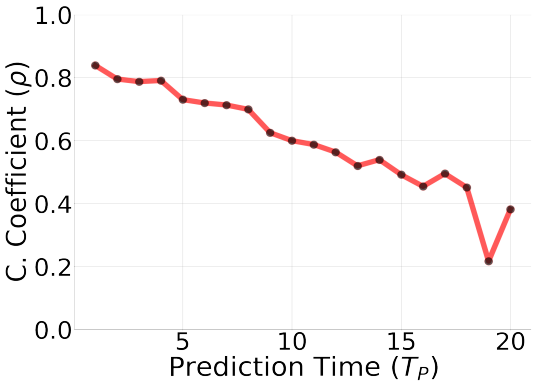
My thesis: methods for electrical power forecasting

The scope of the work was to design an algorithm to predict the energy's unbalances of the Italian power grid controlled by Terna S.p.A. To achieve this aim we followed three different paths and one of those was building a basic RNN. The results given by the Neural network displayed a better accuracy than the other two methods, however the predictive power of the model decreases rapidly as the prediction time interval grows up and this is the reason which motivated me to begin this project. Since I want to forecast a 1-dimensional time series the network needs to have one neuron in the input layer an one neuron in the output layer. To maximize the predictive power I used 1000 neurons in the hidden layers that provided more than 10^6 parameters to optimize. To avoid overfitting I needed a large dataset and, for this reason, in phase of model selection, I used 10^4 points: the 5% of them for the validation set and the remaining for the training set. Since it is a regression problem, I used the mean squared error as loss function and I trained the network using the Adam optimizer. In phase of model assessment I used a test set of 3000 points and as measure of the predictions' accuracy I used the correlation coefficient between the original and predicted time series. Let the energy's unbalances be the time series $\{x_{t'}\}_{t'=0}^{t'=t}$ and let $\{x_{pred,t'}\}_{t'=0}^{t'=t}$ be the time series predicted by the RNN, than the correlation coefficient is calculated by:

$$\rho = \frac{\sum_{i=1}^{n}(x_{pred,i} - \bar{x}_{pred})(x_i - \bar{x})}{\sqrt{\sum_{i=1}^{n}(x_{pred,i} - \bar{x}_{pred})^2}\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}}.$$

Calling T_P the prediction-time interval of my forecasting, for the comparison of the accuracy of the methods used, I plotted the correlation coefficient as function of T_P. Figure 1 shows the most accurate predictions I made for the thesis. In the last chapter of this report I will create the same function $\rho(T_P)$ using the predictions made with the LSTM and I will compare the two results. To read more about my thesis, download it from [1].

Figure 1: The correlation coefficient as function of T_P calculated between the actual and predicted time series of the test set. The predictions are calculated using a basing RNN with hidden size=1000.



The LSTM cell permits to use larger prediction time interval

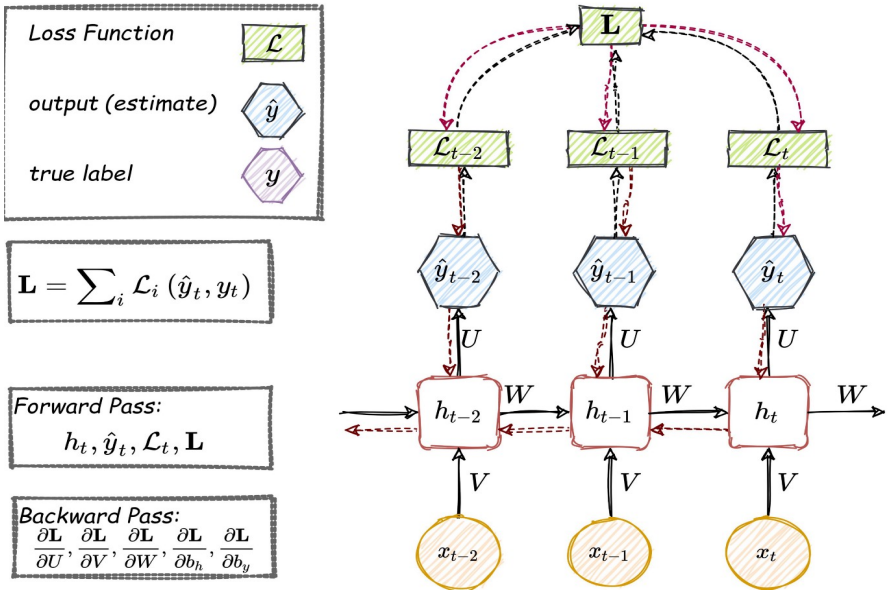
The most effective sequence models used in practical applications are called gated RNNs. On of them is called long short-term memory (LSTM) and it will be used in this project. The main problem regarding basic RNN is related to the training operation over many steps, because is very likely that the gradient will vanish or will explode. To overcome the problem, Gated RNNs create connections among the weights that change at each time step. In practise we need a mechanism which forget the old state by setting it to zero. However, instead of manually deciding when to clear the state, we want the neural network to learn to decide when to do it. This is what gated RNNs do. Supposing that the decreasing trend of $\rho(T_P)$ shown in figure 1 is caused by a vanishing gradient, the LSTM model should fix the problem and increase the predictions' accuracy for large values of T_P.

The vanishing or exploding gradient is caused by recurrent operations

The basic problem of learning long-term dependencies is that gradients propagated over many time steps tend to either vanish (most of the time) or explode (rarely, but with much damage to the optimization). Even if we assume that the parameters are such that the recurrent network is stable (with gradients not exploding), the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions (involving the multiplication of many Jacobians, as we will see in this section) compared to short-term ones. Training an RNN is done by defining a loss function (L) that estimates the error between the true label and the output, and minimizes it by using forward pass and backward pass. The figure 2 summarizes the entire backpropagation through time (BPTT) process. For a single time step, the following procedure is done: first, the input arrives, then it processes trough a hidden layer/state, and the estimated label is

calculated. In this phase, the loss function is computed to evaluate the difference between the true label and the estimated label. The total loss function, L, is computed, and by that, the forward pass is finished. The second part is the backward pass, where the various derivatives are calculated.

Figure 2: Diagram of the information's flow as BPTT operates in a RNN found at [2].



The training of the RNN begin as we "backpropagate" gradients through layers and also through time. Hence, in each time step we have to sum up all the previous contributions until the current one, as given in the equation:

$$\frac{\delta \boldsymbol{L}}{\delta \theta} = \sum_{i=0}^{T} \frac{\delta L_i}{\delta \theta} \propto \sum_{i=0}^{T} \left(\prod_{i=k+1}^{T} \frac{\delta h_i}{\delta h_{i-1}} \right) \frac{\delta h_k}{\delta \theta}$$

However two common problems usually occur during the backpropagation through time:

- Vanishing gradient: $\left\| \frac{\delta h_i}{\delta h_{i-1}} \right\|_2 < 1$
- Exploding gradient: $\left\| \frac{\delta h_i}{\delta h_{i-1}} \right\|_2 > 1$

In the first case, the term goes to zero exponentially fast, which makes it difficult to learn some long period dependencies. This problem is called the vanishing gradient. In the second case, the term goes to infinity exponentially fast, and their value becomes a NaN due to the unstable process. This problem is called the

exploding gradient. Both cases are particular to recurrent networks as argued [2]. Since wee can think of the following recurrence relation

$$\boldsymbol{h_t} = \boldsymbol{W^T h_{t-1}}$$

as a very simple recurrent neural network lacking a nonlinear activation function and lacking inputs $\boldsymbol{x}$, the derivatives $\frac{\delta h_i}{\delta h_{i-1}}$ are equal to each other. This causes the gradient be proportional to a product of many identical factors. In the scalar case, imagine multiplying a weight w by itself many times. The product w^t will either vanish or explode depending on the magnitude of w. If we make a non recurrent network that has a different weight w_t at each time step, the situation is different. If the initial state is given by 1, then the state at time t is given by $\prod_t w_t$. Very deep feed forward networks with carefully chosen scaling can thus avoid the vanishing and exploding gradient problem.

Explanation of the functioning of the LSTM model

As underline Ian Goodfellow et al. ([3]), the clever idea of introducing self-loops to produce paths where the gradient can flow for long a duration is a core contribution of the initialling short-term memory(LSTM) cell. By making the weight of this self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically. In this case, we mean that even for an LSTM with fixed parameters, the time scale of integration can change based on the input sequence, because the time constants are output by the model itself.
The corresponding forward propagation equations are given below, for a shallow recurrent network architecture. Instead of a unit that simply applies an element-wise nonlinear map to the affine transformation of inputs and recurrent units, LSTM recurrent networks have "LSTM cells" that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN. Each cell has the same inputs and outputs as an ordinary recurrent network, but also has more parameters and a system of gating units that controls the flow of information.
The most important component is the state unit $s_i^{(t)}$ whose value is defined by:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \tanh\left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\right),$$

where $\boldsymbol{b}$, $\boldsymbol{U}$ and $\boldsymbol{W}$ respectively denote the biases, input weights, and recurrent weights into the LSTM cell. $f^{(t)}$ and $g^{(t)}$ are called respectively *forget gate* and *input gate*. They are computed similarly to each other:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right),$$

$$g_i^{(t)} = \sigma\left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\right),$$

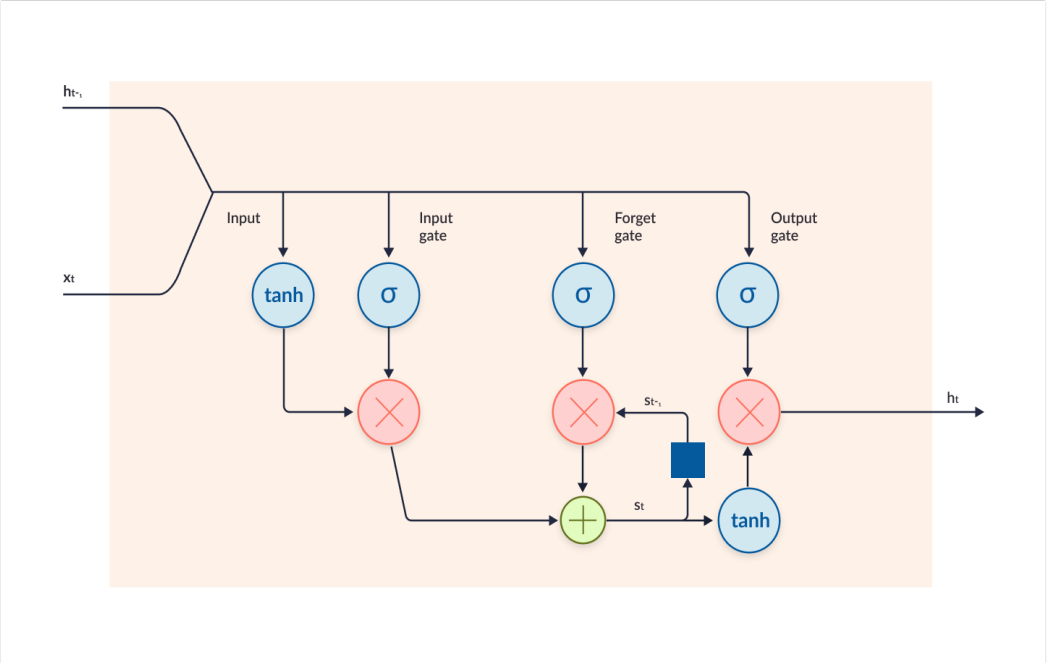
the sigmoid map "gates" the value between 0 and 1 and, thanks to this mechanism, the LSTM cell manages to learn when to forget the state $s^{(t-1)}$ or the ouput of the tanh map, avoiding the exponential increase of the state.
Moreover the output $h_i^{(t)}$ of the LSTM cell can also be shut off, via the *output gate* $q_i^{(t)}$, which also uses a sigmoid unit for gating:

$$h_i^{(t)} = \tanh\left(s^{(t)}\right) q^{(t)},$$

$$q_i^{(t)} = \sigma\left(b_i^q + \sum_j U_{i,j}^q x_j^{(t)} + \sum_j W_{i,j}^q h_j^{(t-1)}\right),$$

which has parameters $\boldsymbol{b^o}$, $\boldsymbol{U^o}$ and $\boldsymbol{W^o}$ for its biases, input weights and recurrent weights, respectively. Even if LSTM networks have been shown to learn long-term dependencies more easily than the simple recurrent architectures, they are not the unique solution for the vanishing gradient. For an overview of the problem I recommend the book [3].

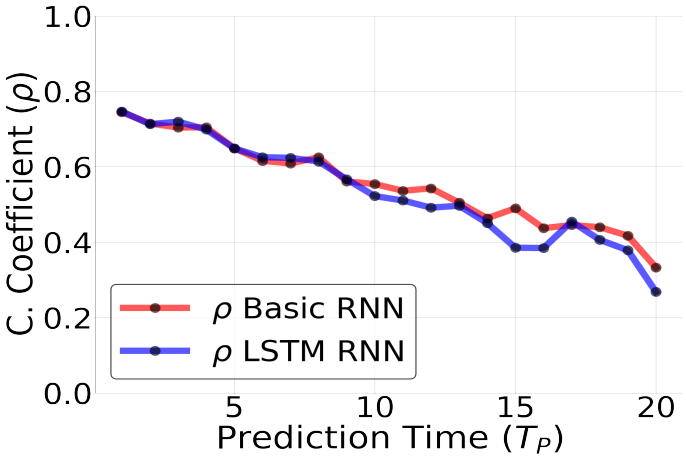
Figure 3: Diagram of the LSTM Cell operations interpolating the sequence given the input x(t). Image found at [4].



The predictions of the LSTM are not more accurate

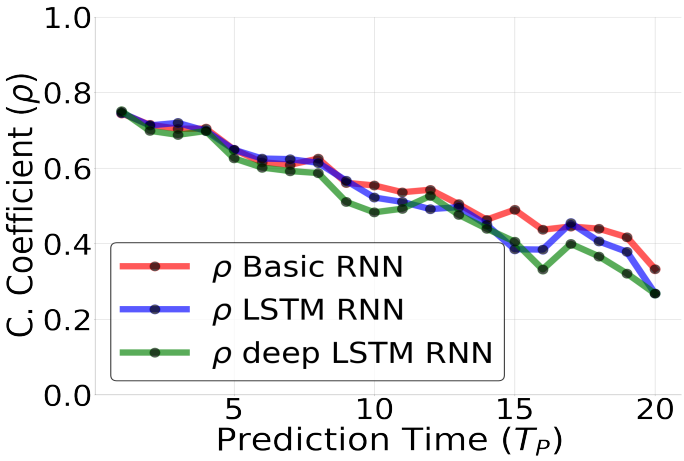
As recommended in many books I start to forecast using a simple LSTM model and than I started increasing the hidden size. As happened with the Basic RNN, the predictive power of the LSTM cell increases with the size of the hidden layer, however this leads to the first problem which I encountered. Since a LSTM cell has got 4 times the number parameters of the basic one , its phase of training is much slower than the basic cell's one at equal hidden size. For this reason I compared the 2 types of cell using a size smaller than the one used for my thesis. Precisely I used an hidden size equal to 250 fro both cells. As explained in the first chapter, the layer in the first and last output needs to have one neuron because we want to forecast a 1-dimensional time series. In phase of model selections and assessment I used the same datasets used for the analysis in my thesis, the MSE as loss function and ADAM optimizer.

Figure 4: The correlation coefficient as function of T_P calculated between the actual and predicted time series of the test set. In blue the predictions are made using a LSTM cell, in red using a basic cell.



The results suggest that the predictions are similar in accuracy, even for larger prediction-time interval when the Basic RNN is supposed to have a vanishing or exploding gradient. For this reason I conclude that using a LSTM cell did not improve the predictive power of the model.
To have a better understanding and complete the discussion I tried to forecast the data using a more complex model that consists in a deeper network made of two hidden layers of LSTM cells. The first of size 250, the second of size 1. However, the results did not improved. As figure 5 shows the accuracy is even worse than before.

Figure 5: The correlation coefficient as function of T_P calculated between the actual and predicted time series of the test set. In blue the predictions are made using a LSTM cell, in red using a basic cell, in green using a deep LSTM cell.



The data format can influence the predictions' accuracy

During the analysis of the models I noted that the shape of the dataset given to RNN both as input and target can qualitatively change the predictions' accuracy. As underlined in [5], the way the recurrent neural networks are trained differs from how machine learning algorithms are usually trained. Typically a machine learning algorithm is trained by learning the relationship between the x data and the y data. However recurrent neural networks are trained to recognize the relationship in a sequence of y values. For a standard machine learning algorithm, the training data has the form of (x,y) so the machine learning algorithm learns to associate a y value with a given x value. This is useful when the test data has x values within the same range as the training data. However, for this application, the x values of the test data are outside of the x values of the training data and the traditional method of training a machine learning algorithm does not work as well. For this reason, in the last chapter, the recurrent neural network was trained on sequences of y values of the form

$$\begin{aligned} \boldsymbol{D_1} = \{&((y_t, y_{t+1}, ..., y_{t+S}), (y_{t+T_P}, y_{t+1+T_P}, ..., y_{t+S+T_P})), \\ &((y_{t+1}, y_{t+2}, ..., y_{t+1+S}), (y_{t+1+T_P}, y_{t+1+T_P}, ..., y_{t+1+S+T_P})), \\ &...\} \end{aligned}$$

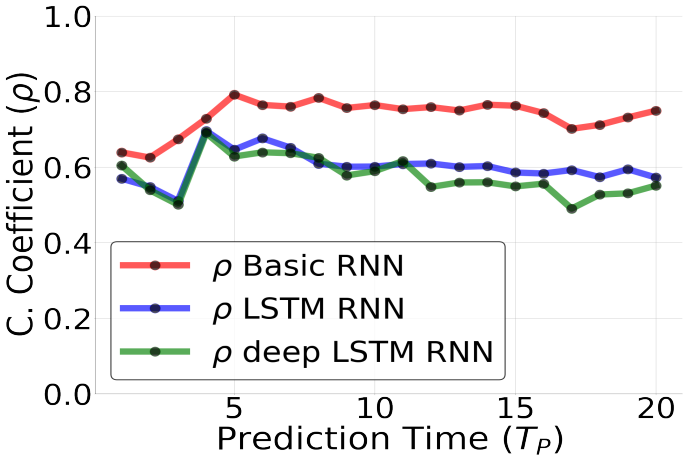
where s is fixed parameter that represent the length of the sequence of each batch of data. In this way the network is concerned with learning the pattern of the y data and not the relation between the x and y data.

However, while I was making tests, I noted that organizing data in tensors like

$$\begin{aligned} \boldsymbol{D_2} = \{ & ((y_t, y_{t+1}, ..., y_{t+T_P}), y_{t+2T_P}), \\ & ((y_{t+1}, y_{t+2}, ..., y_{t+1+T_P}), y_{t+1+2T_P}), \\ & ...\} \end{aligned}$$

let the network improve the long-term performance at the expense of the short-term ones solving the problem that motivated me to begin this project. Moreover the improvement is such that, to reach the same value of accuracy obtained in my thesis, only 32 neurons in the hidden layer and 700 data for the training set are sufficient. This makes the whole process even fast.

Figure 6: The correlation coefficient as function of T_P calculated between the actual and predicted time series of the test set. In blue the predictions are made using a LSTM cell, in red using a basic cell, in green using a deep LSTM cell. In contrast to what has been done in figure 5, data have been reshaped in the second type of the tensor $\boldsymbol{D_2}$.

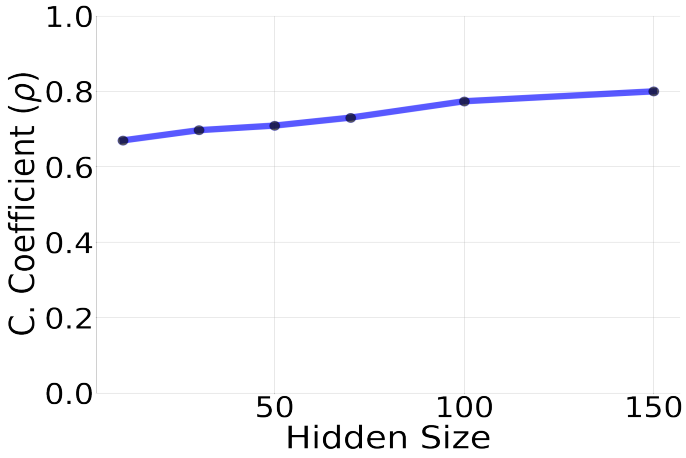


As figure 6 shows, using this data format, the accuracy of long-term predictions improve, even if, against expectations, the LSTMs seem to generate the less effective models.

The performance of the LSTM models are not caused by overfitting

Someone might argue that LSTM models overfit data during training, since their cell have (at fixed hidden size) 4 times the basic cell's number of parameters. However the accuracy of the test set improve at the increasing hidden layer's size. As instance I plotted in figure 7 the correlation coefficient as function of the cell's hidden size at fixed prediction time interval $T_P = 4$. In the example has been used a LSTM cell trained using data format $\boldsymbol{D_1}$, however I noted the same trend for all the models and data format used for this report.

Figure 7: The correlation coefficient between the actual and predicted time series of the test set with fixed $T_P = 4$. For this image has been used just an hidden layer of a LSTM model, whose hidden size has been changed starting from 30 to 150. Since for figure 6 I used the same network with hidden size=32, I conclude that the model did not overfit data.



Conclusions

- Using data format $\boldsymbol{D_1}$ permits to make more accurate short-term predictions, while using data format $\boldsymbol{D_2}$ permits to make more accurate long-term predictions.
- LSTM have made less accurate predictions than a basic RNN cell.

I suppose that in general the network more easily predict T_P time-steps in the future training on $\boldsymbol{D_2}$, because the network is trained on batches of sequence T_P time-steps long. However, for small T_P, the network is trained on short-length sequences and does not have enough information to learn. This problem is indeed solved training the network using $\boldsymbol{D_1}$ due to sequence-length be fixed by the parameter S.
Regarding the second point I don't have a complete answer. I suppose I haven't found the right hyperparameters during the training operation due to the algorithm being stuck in a local minimum, in addition to the fact that the problem seems to be easy for a basic RNN.

Bibliography

[1] Luca Zilli. Short-time forecasting using chaos theory: The case of the italian energy market. B.s. thesis, Università degli studi di Milano, december 2020. https://github.com/LucaZilli/THESIS.

[2] Barak Or. The exploding and vanishing gradients problem in time series. https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Deep learning long short-term memory (lstm) networks: What you should remember. https://missinglink.ai/guides/neural-network-concepts/deep-learning-long-short-term-memory-lstm-networks-remember/.

[5] Carlo Barbieri. Programmi in python per il corso di "machine learning con applicazioni". https://gitlab.com/craolus/MachineLearning-con-applicazioni/-/blob/master/README.md.