**USING NEURAL NETWORK TO ANOMALY DETECTION**

Another way to perform forecasting on time series data is to use artificial neural networks (ANN). However, the ANN's types that we already studied do not support time-series because it does not have any temporal feedback process.

The backpropagation algorithm, used in the regular ANN, provides a different feedback channel. The backpropagation's idea is based on updating the weight values by calculating the error between predicted output and target output and then using an algorithm (gradient descent). This algorithm does not consider gradient descent results in the past moment, applying the calculation during the defined steps and presenting its results.

The time series models require a different perspective. The present prediction involves understanding the past; if you do not know the history, your error increases—an example of this type of problem you can see in the weather forecast. Suppose you know that it is running today. In that case, you can make a better prediction of what will happen tomorrow (remember when you studied the Bayesian rule in the statistic and probabilities course).

To solve this problem, you can use a different ANN type, the **Long Short-Term Memory (LSTM)** network. LSTM networks are a type of **recurrent neural network (RNN)** capable of learning order dependence in sequence prediction problems[[1]](#footnote-1). *Unlike standard feedforward neural networks, LSTM has feedback connections*.

An RNN has an internal state that can represent context information. They keep information about past inputs for an amount of time that is not fixed a priori but rather depends on its weights and the input data.

RRN contains cycles that feed the network activations from a previous time step to the network to influence predictions at the current time step. These activations are stored in the network's internal states, which can, in principle, hold long-***term temporal contextual information***. This mechanism allows RNNs to exploit a dynamically changing contextual window over the input sequence history[[2]](#footnote-2). Figure 1 presents how RNN works.

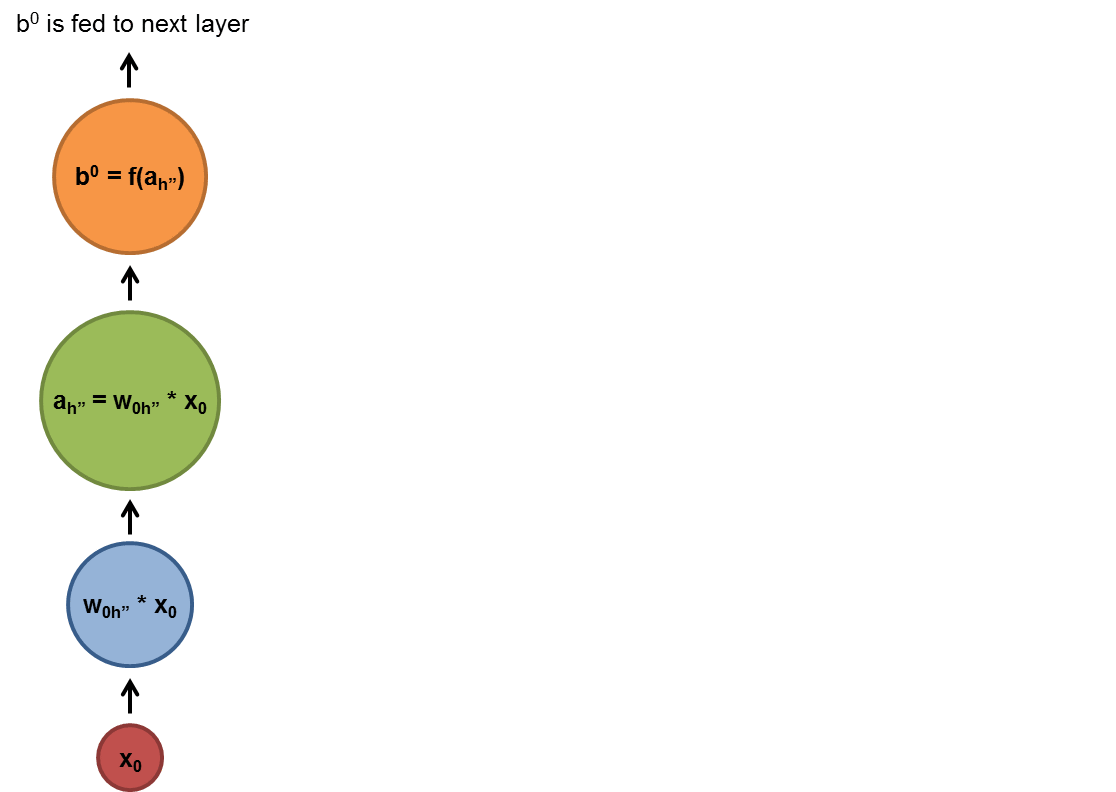


Figure 1 - How RNN works [source: https://i.pinimg.com/originals/fe/63/3c/fe633cdec14b8f32adf1c441e37f58dd.gif].

Unfortunately, the range of contextual information that standard RNNs can access is in practice quite limited, *fail to learn in the presence of time lags more significant than 5 – 10 discrete time steps* between relevant input events and target signals. A recent model, "Long Short-Term Memory" (LSTM), is not affected by this problem. LSTM can learn to bridge minimal time lags over 1000 discrete time steps by enforcing constant error flow-through "constant error carrousels" (CECs) within special units, called cells.

The LSTM architecture was motivated by an analysis of error flow in existing RNNs which found that long-time lags were inaccessible to existing architectures. Backpropagated error either blows up or decays exponentially. An LSTM layer consists of a set of recurrently connected blocks, known as **memory blocks**. These blocks can be thought of as a differentiable version of the memory chips in a digital computer. Each one contains one or more recurrently connected memory cells and three multiplicative units – the **input, output, and forgets gates** – that provide continuous analogs of write, read and reset operations.

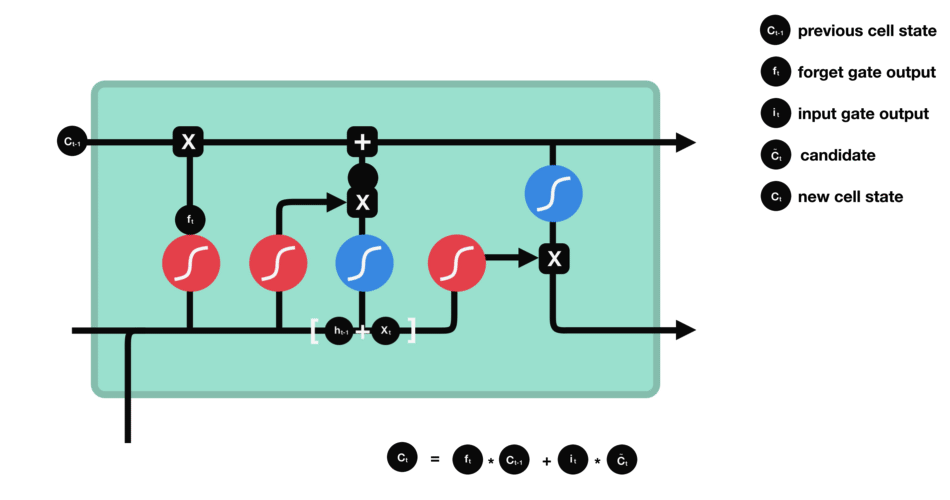


Figure 2 - LTSM [source: https://miro.medium.com/max/1900/1\*cmv5EOAd6iWMzWvHrZbl-w.gif].

Figure 2 presents how LTSM works. There you can see the information's update through the logical gates. When you compare both models is easiest to see the difference (check Figure 3). The RNN only has a single tanh layer in its structure, while the LSTM structure consists of 4 intersecting layers. This feature with the double connection paths reduces the long dependency of previous time information.

|  |  |
| --- | --- |
|  | RNN structure |
|  | LSTM structure |

Figure 3 - LSTM x RNN.

1. What is the main problem of the regular ANN when need to handle time-series data?
   1. There is no problem; the regular ANN is used to classification and forecast problems using time-series data.
   2. The main problem is related to the performance (time and processor). The regular ANN implementation consumes a massive amount of time and processors to solve time-series problems, like forecast and classification.
   3. The main problem is related to the performance (only time). The regular ANN implementation consumes a massive amount of time to solve time-series problems, like forecast and classification.
   4. **The regular ANN does not have any feedback channels based on previous time results, which is required to develop a machine learning model using time-series.**
2. Which does the limitation of RNN LSTM solve?
   1. The RNN does not provide contextual information.
   2. **The RNN does not work well with sizeable discrete time steps between relevant input events and target signals (bigger than 20).**
   3. LSTM improves the number of cells that you can use to solve machine learning problems.
   4. The RNN only has a single tanh layer in its structure, while the LSTM structure consists of 4 intersecting layers.

**HANDLING DATA IN THE ANN – USING NUMPY LIBRARY**

NumPy[[3]](#footnote-3) is a library for the Python programming language, adding support for large, multidimensional arrays and matrices, along with an extensive collection of high-level mathematical functions to operate on these arrays. Together with Pandas[[4]](#footnote-4), which is a base of data manipulation in the ML models.

NumPy's main object is the homogeneous multidimensional array. It is a table of elements (usually numbers), all of the same type, indexed by a tuple of non-negative integers. In NumPy, dimensions are called axes.

For example, the coordinates of a point in 3D space [1, 2, 1] have one axis. That axis has three elements in it, so we say it has a length of 3. In the example pictured below, the array has two axes. The first axis has a size of 2, and the second axis has a length of 3.

[[ 1., 0., 0.], [ 0., 1., 2.]]

An excellent reference to understanding how to works with NumPy can be found in the <https://numpy.org/doc/stable/user/absolute_beginners.html>. Use the above material to answer the following questions.

1. What is the output of the following code?

|  |
| --- |
| def question1():  import numpy as np  a = np.array([1, 2, 3])  print(type(a))  print(a.shape)  question1() |

<class 'numpy.ndarray'>

(3,)

1. What is the output of the following code?

|  |
| --- |
| def question2():  import numpy as np  a = np.array([1, 2, 3])  print(a.shape)  question2() |

(3,)

1. Given the following code, answer the questions?

|  |
| --- |
| a = np.array([[1,2,3],[4,5,6]]) |

1. What is the shape of array' a'?

(2, 3)

1. What is the dimension of array 'a'?

2

1. Given the following code, answer the questions?

|  |
| --- |
| a = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]]) |

1. What is the shape of array 'a'?

(2, 2, 3)

1. What is the dimension of array 'a'?

3

1. Given the following code, answer the questions?

|  |
| --- |
| c = np.array([[1, 2, 3], [4, 5, 6]]) |

1. What is the output of this call: *c.max()?*

6

1. What is the output of this call: *c.max(axis=0)?*

[4 5 6]

1. What is the output of this call: *c.max(axis=1)?*

[3 6]

1. Given the following code, answer the questions?

|  |
| --- |
| c = np.array([[[11,2,15],[4,5,6]],[[7,8,9],[10,3,1]]]) |

1. What is the output of this call: *c.max()?*

15

1. What is the output of this call: *c.max(axis=0)?*

[[11 8 15]

[10 5 6]]

1. What is the output of this call: *c.max(axis=1)?*

[[11 5 15]

[10 8 9]]

1. Given the following code, answer the questions?

|  |
| --- |
| c = np.array([[[11,2,15],[4,5,6]],[[7,8,9],[10,3,1]]]) |

1. What is the output of this call: *print(c[0])?*

[[11 2 15]

[ 4 5 6]]

1. What is the output of this call: *print(c[0, 1])?*

[4 5 6]

1. What is the output of this call: *print(c[1, 0, 1])?*

8

1. What is the output of this call: *print(c[1, -1])?*

[10 3 1]

1. What is the output of the following code?

|  |
| --- |
| def question9():  import numpy as np  c = np.array([[[11,2,15],[4,5,6]],[[7,8,9],[10,3,1]]])  e = c.reshape(3, 4)  print(e) question9() |

[[11 2 15 4]

[ 5 6 7 8]

[ 9 10 3 1]]

1. In the matrix ‘c’ (presented in Question 8), is it possible to reshape the matrix to this shape [5,3]? Why is it not possible?

No, it is not possible because the new shape needs to be multiple of the array elements.

**IMPLEMENTING A LSTM NETWORK**

During the first part of this tutorial, you learned about the LSTM networks and their main characteristics. In the second part, we understand the basic features of a vital support library, enabling data manipulation in unique arrays.

The current part shows how to implement an LSTM network using two robust frameworks: Keras[[5]](#footnote-5) and TensorFlow[[6]](#footnote-6). Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs. It minimizes the number of user actions required for common use cases and provides clear & actionable error messages.

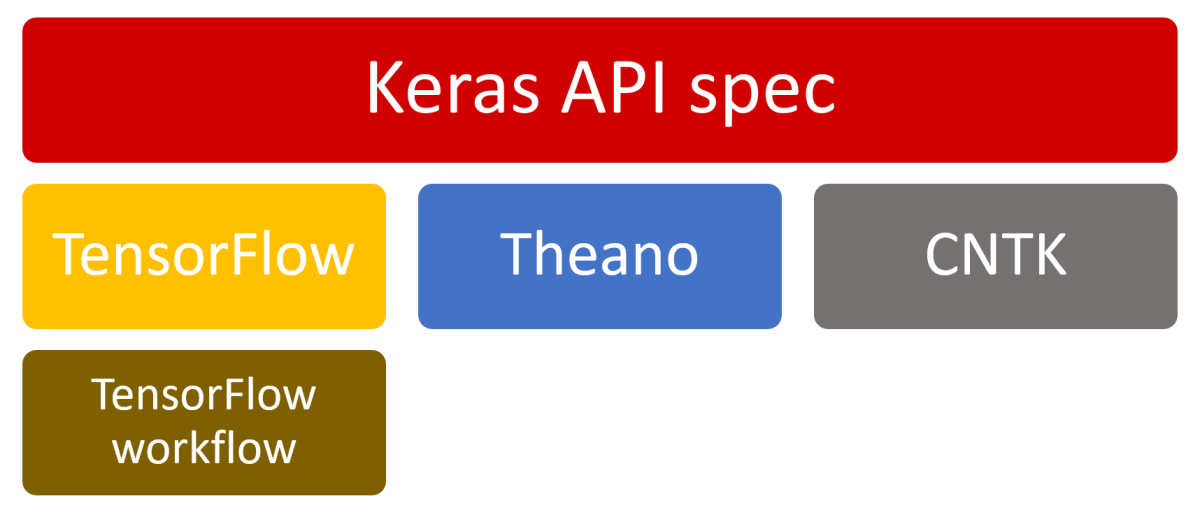


Figure 4 - Keras API.

Keras provides a high-level API stack designed for human beings, not machines, which reduces the complexity involves in the math operations required to implement a deep learning model. In Figure 4, it is possible to see that Keras can implement different low-level deep-learning frameworks, like TensorFlow, Theano, and CNTK.

Besides the options, the most famous and stable framework is TensorFlow. TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push state-of-the-art ML. Developers can quickly build and deploy ML-powered applications.

The installation of tensor-flow and Keras following commands:

*# Installing Tensorflow  
pip install tensorflow  
# Installing Keras  
pip install --upgrade keras*

The used scenario to show the LTSM use for anomaly detection is the same in the previous tutorial (CPU load). The main difference of the current read\_csv() approach[[7]](#footnote-7) is that you define that Pandas try to infer the data fields as a timestamp. You do not return a panda dataset in the method but a Numpy representation of the DataFrame (*property values*).

Equation 1 - Load Data from CSV file.

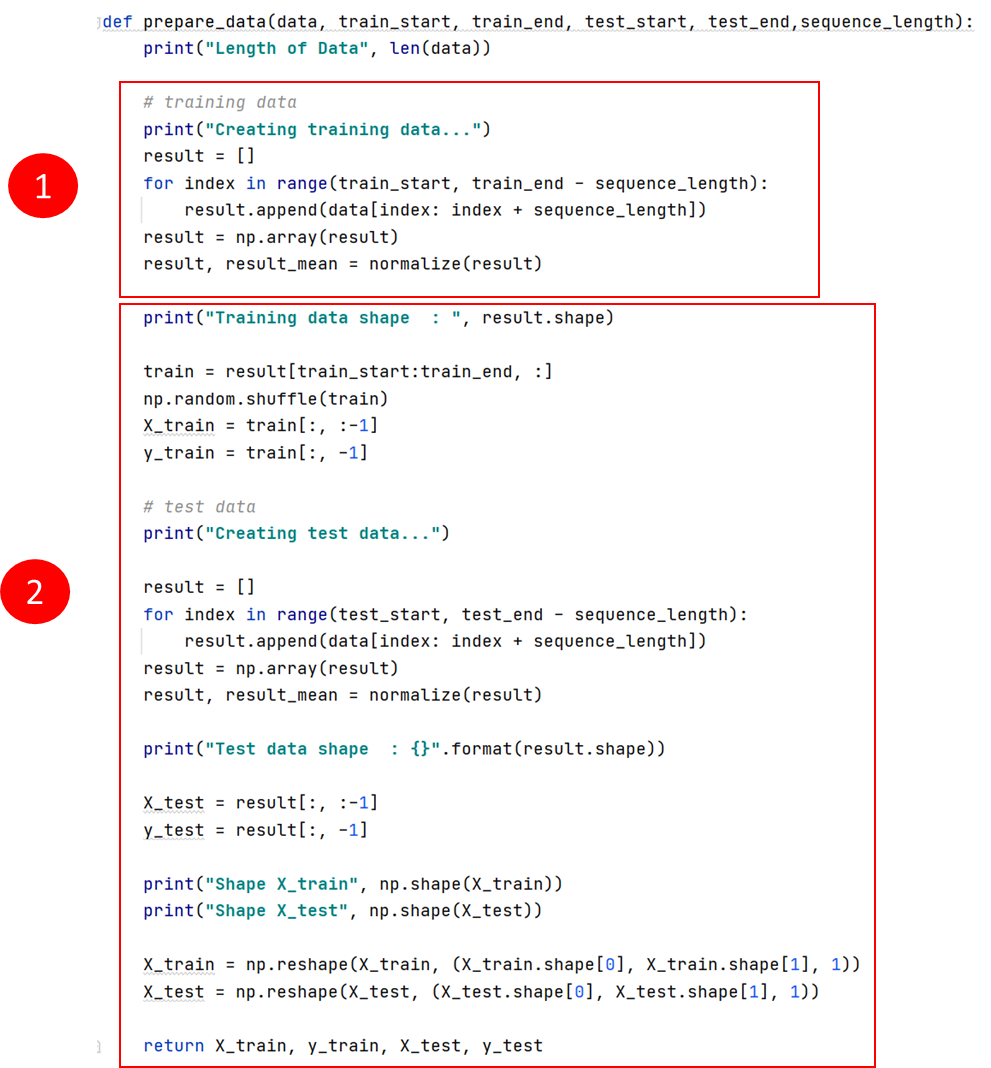
Interface gráfica do usuário, Texto

Descrição gerada automaticamente

The next step is preparing data (split the dataset into training and test data) to be consumed by the LSTM model. An essential question in the LTSM design is that lags (time between the observations) need to be fixed; it does not change over time. This reason makes that we defined this lag using the ***sequence\_lenght*** variable.

The first task is the creation of a ***temp array*** using the for a loop. This array will be used for the generation of test and training datasets. Following, you need to convert it into a Numpy array and normalize it, transforming the data to be in this interval (-1 to 1). The normalization method returns the array normalized as its mean. This method is presented in *Equation 3*.

Equation 2 - Preparing Data to LSTM model.



Equation 3 - Normalization Method.

Texto, Carta

Descrição gerada automaticamente

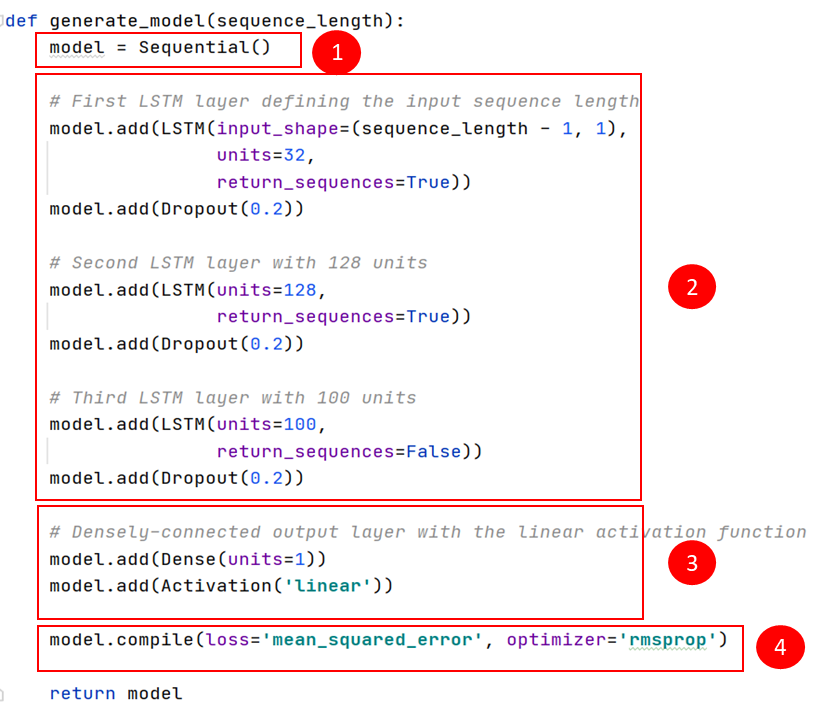
If the dataset in the correct format (Numpy) and normalized, we can start to create the test and the training dataset. A complementary and essential task is the X array's reshaping, required to fill the LSTM model. The first parameter is the array to be reshaped; the second is the new shape (3d array).

Now, the dataset is ready to generate the LTSM model (see Equation 4). The first step is the instantiation of a Sequential model. A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

After the model instantiation, you need to create the LTSM layers[[8]](#footnote-8). During the layer, instantiation is required to define the output dimensional (units). Suppose the output is or not the complete sequence of data[[9]](#footnote-9) (perceives that this param is False only in the last LTSM layer). Also, the first layer is required to define the input shape, which is a bi-dimensional array.

Together with the LTSM layer is required a particular layer, which names are Dropout. The insertion of a Dropout layer aims to regularize the ANN. In general, regularization means to make things regular or acceptable. In machine learning, regularization is the process that regularizes or shrinks the coefficients towards zero. In simple words, regularization discourages learning a more complex or flexible model to prevent overfitting.

Equation 4 - Model Generation.



Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped out” randomly. This means that their contribution to downstream neurons' activation is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass. The effect is that the network becomes less sensitive to the specific weights of neurons. The consequence’s results are a network capable of generalization and less likely to overfit the training data.

The param passed during the Dropout layer's instantiation is the rate, fraction of the input units to drop.

The last layer in the model is Dense. A Dense layer implements the operation: ***output = activation(dot(input, kernel) + bias)***, where activation is the element-wise activation function passed as the activation argument, a kernel is a weights matrix created by the layer, and bias is a bias vector produced by the layer (only applicable if use\_bias is True). An activation function defines the output of that node given an input or set of inputs, in particular. Using a Dense Layer at the end of the model connects the previous layer to generate the desired output.

The last part of model generation is the compilation. To compile an artificial neural network is required to define the loss as the optimizer. The loss or cost functions represent the optimization objective of the model. The main idea is to select an optimization model that minimizes the error, making the model more accurate.

In the current implementation, we define that the error is measured using the *mean square error (MSE)*, and the optimization function is the *rmsprop*. The mean square is defined as the arithmetic mean of the squares of the prediction value (y) minus the test value () (see below Equation).

RMSprop is a gradient-based optimization technique used in training neural networks. Gradients of complex functions like neural networks tend to either vanish or explode as the data propagates through the function (refer to vanishing gradients problem). Rmsprop was developed as a stochastic technique for mini-batch learning.

RMSprop deals with the above issue by using a moving average of squared gradients to normalize the gradient. This normalization balances the step size  (momentum),  decreasing the step for large gradients to avoid exploding and increasing the degree for small gradients to avoid vanishing.

Simply put, RMSprop uses an adaptive learning rate instead of treating the learning rate as a hyperparameter. It means that the learning rate changes over time.

1. Given two sets, A (test set) and B (model’s predictions), calculate the mean square error (MSE) of the model (show the calculations):
   1. A = {0.3, 0.6, 0.8, 0.6}
   2. B={0.2,0.4,0.7, 0.5} is the model predictions;
2. What is the problem in the recursive neural network (RNN) that the RMSP algorithm solves?
   1. The RMSP algorithm solves the vanishing gradients problem, which tends to either vanish or explode as the data propagates through the gradient function.
   2. The RSMSP algorithm solves the rate learning fixed problem.
   3. It solves the elevation of error rate into the RNN networks have.
   4. It does not solve any problem in the RNN.
3. What is TRUE about the Dropout layer:

The Dropout layer increases the number of nodes, consequently, **INCREASE** the accuracy.

Dropout is a technique where randomly selected neurons are ignored during training; consequently, **INCREASE** the time and processors use.

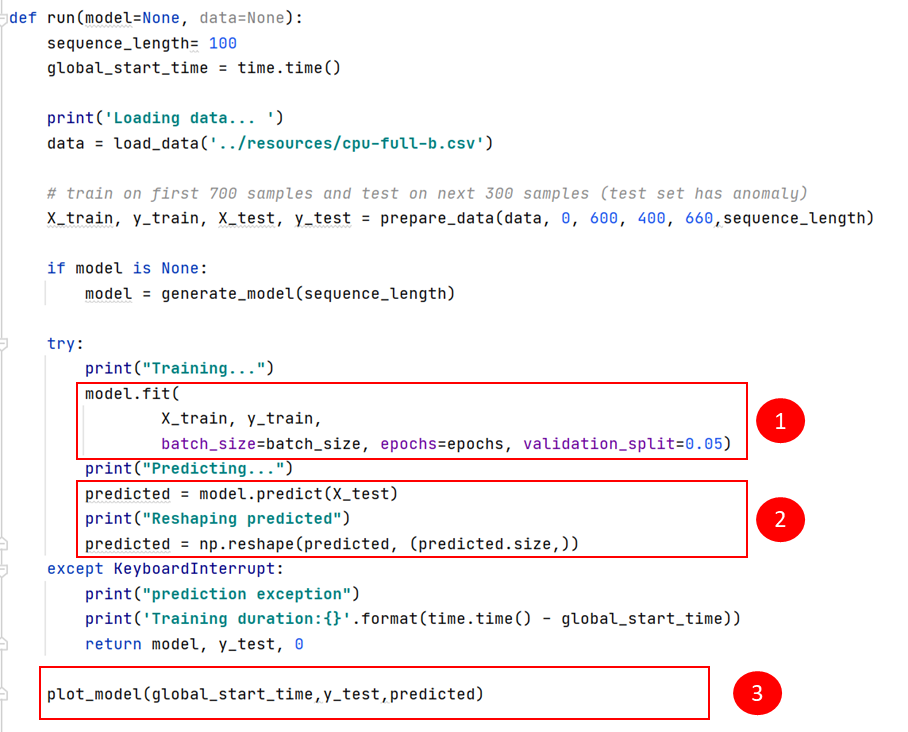
Dropout is a technique where randomly selected neurons are ignored during training; consequently, **DECREASE** the time and processors use.

* 1. Dropout is a technique where randomly selected neurons are ignored during training; consequently, the network results are capable of **BETTER** generalization and are less likely to overfit the training data.

**USING A LSTM NETWORK TO IDENTIFY ANOMALY BEHAVIORS**

After the generation of the model, the next step is its use to predict anomaly behaviors. The process is very similar to the previous ARIMA model. Following the X and Y training values, you can define the batch size (it is optional). The batch size is the number of samples per gradient update. If it is unspecified, its value will be 32. Next, you need to define the number of epochs applied to the model. Finally, you define the fraction of the training data to be used as validation data (validation split).

Equation 5 - Running the LTSM model.



An important feature is that you can check the MSE of each epoch during the ANN training (see Figure 5). Using the previous cited Figure, we can see that the three epochs' errors were: 0.4913, 0.2254, and 0.1986.

Texto

Descrição gerada automaticamente

Figure 5 - Training an ML model.

After the model is trained, it is ready to make a prediction. To be possible to have a better understanding of its accuracy, you can plot the result (check Figure 6).

Equation 6 - Plot the Predictions Performance.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Gráfico, Gráfico de linhas, Histograma

Descrição gerada automaticamente

Figure 6 - Model Performance.

1. Based on the time series shown in Figure 6, what period you as an Analyst can state with a small margin of error that an anomaly has occurred.
   1. In the interval 60 to 120.
   2. In the interval 0 to 40.
   3. In the interval 110 to 150.
   4. This time series does not contain an anomaly.

1. Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. PMID 9377276. S2CID 1915014. [↑](#footnote-ref-1)
2. Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." IEEE transactions on neural networks 5.2 (1994): 157-166. [↑](#footnote-ref-2)
3. <https://numpy.org/> [↑](#footnote-ref-3)
4. <https://pandas.pydata.org/> [↑](#footnote-ref-4)
5. <https://keras.io/> [↑](#footnote-ref-5)
6. <https://www.tensorflow.org/> [↑](#footnote-ref-6)
7. For more information: <https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html> [↑](#footnote-ref-7)
8. <https://keras.io/api/layers/recurrent_layers/lstm/> [↑](#footnote-ref-8)
9. The *return\_sequences* params default is False. [↑](#footnote-ref-9)