Anomaly detection for Cybersecurity: time series forecasting and deep learning

Giordano Colò

December 17, 2019

Abstract

Finding anomalies when dealing with a great amount of data creates issues related to the heterogeneity of different values and to the difficulty of modelling trend data during time. In this paper we combine the classical methods of time series analysis with deep learning techniques, with the aim to improve the forecast when facing time series with long-term dependencies. Starting with forecasting methods and comparing the expected values with the observed ones, we will find anomalies in time series. We apply this model to a bank cybersecurity case to find anomalous behavior related to branches applications usage.

1 Time series forecasting

1.1 Introduction

Parameters and properties describing the data are intrinsecally linked to the time at which they are collected. In order to modelling relations underlying them, an analysis of how they change is vital. Mathematical statistics point of view is to define a time series and deal with it in order to find recurrent patterns and abrupt or slowly changes.

1.2 Time series

A time series is a set of observations x_t each one recorded at a different time t. During the rest of the paper we will deal with *discrete time series*, in which the set T_0 of times of the observations is a discrete set.
The principal point of interest of time series is the possibility to draw inferences from their underlying laws.
The analysis of the series is performed by setting a probability model to represent the data (or a family of probability models). After the model is chosen, the estimation of parameters and the goodness of the model to fit data can help understanding the mechanism that generates the series.
Using this model, one can give a compact description of the data, by identifying for example seasonal components and remove them, in order not to confuse these components with long-term trends. Let's define what a time series model is.

A *time series* for observed data x_t is a specification of the joint distributions of a sequence of random variables X_t of which x_t is a realization.

A complete probabilistic time series model for the sequence of random variables $X_1, X_2, ...$ would specify all the joint distributions of the random vectors $X_1, .., X_n$, or equivalently all the probabilities

$$P[X_1 \leq x_1, X_n \leq x_n],\ -\infty < x_1, ..., x_n < +\infty,\ n = 1, 2, ... \tag{1.1}$$

Anyway, such a specification is rare in time series analysis, because in general there are too many parameters to be estimated from the data. Usually all that we do is find the *first-* and *second-order moments* of the joint distributions: the expected values EX_t and the expected products $E(X_{t+h}, X_t)$, $t = 1, 2, ...$, $h = 0, 1, 2, ...$, focusing on the sequence X_t that depends only on these ones.

The general approach adopted to analyze the time series is described by the following points.

1. Plot the series and check for :
   (a) trend,
   (b) seasonal component,
   (c) any apparent sharp changes in behavior,

(d) any outlying observation.

2. Remove the trend and seasonal components to get *stationary* residuals (we will define stationary below). There are different ways to evaluate trend and seasonal components, such as estimating them and subtracting from the data or *differencing* the data, i.e. replacing the original series X_t by $Y_t = X_t - X_{t-d}$ for some positive integer d. In any case, the aim is to produce a stationary series, the values of which we refer to as residuals.

3. Choose a model to fit the residuals, making use for example of autocorrelation function.

4. Forecasting will be done by forecasting the residuals and then inverting the transformation described above to arrive at forecasts of original series X_t.

5. Another approach is to express the series in terms of Fourier components, which are sinusoidal waves of different frequencies.

The most important objective of the analysis is to find stationary residuals. In order to give a general idea, a time series $\{X_t, t = 0, 1, ..., n\}$ is stationary if it has statistical properties similar to the "shifted" series $\{X_{t+h}, t = 0, \pm 1, ...\}$. Remember that we have to restrict the focus only on first- and second-order moments of $\{X_t\}$, so we give a more detailed definition:

Definition 1.1. Let $\{X_t\}$ be a time series with $E(X_t^2) < \infty$. The mean function of $\{X_t\}$ is

$$\mu_X(t) = E(X_t). \tag{1.2}$$

The covariance function of $\{X_t\}$ is

$$\gamma_X(r, s) = Cov(X_r, X_s) = E[(X_r - \mu_X(r))(X_s - \mu_X(s))] \tag{1.3}$$

for all integers r and s.

Definition 1.2. We say that $\{X_t\}$ is *weakly stationary* if:

1. $\mu_X(t)$ is independent of t,

2. $\gamma_X(t + h, t)$ is independent of t for each h.

If we add the condition that $(X_1, ..., X_n)$ and $(X_{1+h}, ..., X_{n+h})$ have the same joint distribution, the series is strictly stationary.

A fundamental tool used to analyze the series is the autocovariance function:

Definition 1.3. Let $\{X_t\}$ be a stationary time series. The *autocovariance function* (ACVF) of $\{X_t\}$ at lag h is:

$$\gamma_X(h) = Cov(X_{t+h}, X_t). \tag{1.4}$$

In the same way we define the *autocorrelation function* (ACF) of $\{X_t\}$ at lag h as:

$$\rho_X(h) = \frac{\gamma_X(h)}{\gamma_X(0)} = Cor(X_{t+h}, X_t) \tag{1.5}$$

Roughly speaking, these are measures of covariance and correlation "internal" to the series at different lags. This is very useful when one have to forecast future values of the series based on series analysis. In view of these definitions, we expect that the ACF falls towards zero as the points become more separated. This is logically due to the fact that it's harder to forecast data in the future.

When analyzing a time series, the first step is to plot the data and breaking it into homogeneous segments. As for the outliers, before discarding them, they have to be studied carefully to check what is the meaning of these values. By inspecting the graph it's possible to represent the data as a realization of a process and use the classical decomposition model:

$$X_t = m_t + s_t + Y_t \tag{1.6}$$

where m is a slowly changing function known as *trend*, s_t is a function with a period d referred to as *seasonal component*, and Y_t is a *random noise component* that is stationary. Our aim is to estimate and extract the deterministic components m_t and s_t in the hope that the residual component Y_t will be a stationary time series. Then, by using the theory of such processes, we will search for a satisfactory probabilistic model for the process Y_t, to analyze its properties and to use it, in conjunction with m_t and s_t for purpose of prediction of $\{X_t\}$.

1.3 Trend and Seasonality estimation

In order to estimate the trend component of the time series we can use two general approach: the first is to fit a polynomial trend (by least squares) and subtract the fitted trend from the data to find an appropriate stationary model; the second is to eliminate the trend by differencing (as we will see in a few).

Smoothing with a finite moving average filter. Let q be a nonnegative integer and consider the moving average:

$$W_t = (2q+1)^{-1} \sum_{j=-q}^{q} X_{t-j} \tag{1.7}$$

of the process $\{X_t\}$. Then, for $q+1 \leq t \leq n-q$,

$$W_t = (2q+1)^{-1} \sum_{j=-q}^{q} m_{t-j} + (2q+1)^{-1} \sum_{j=-q}^{q} Y_{t-j} \approx m_t \tag{1.8}$$

assuming that m_t is approximately linear in the interval $[t-q, t+q]$ and that the average of the error terms over this interval is close to zero. So the moving average provides us with the estimates:

$$\hat{m} = (2q+1)^{-1} \sum_{j=-q}^{q} X_{t-j}, \quad q+1 \leq t \leq n-q \tag{1.9}$$

In figure 1 we show an example of how the moving average filter is applied.

Exponential smoothing. For any fixed $\alpha \in [0,1]$, the process of applying the one-sided moving averages $\hat{m}_t$, defined by the recursions

$$\hat{m} = \alpha X_t + (1-\alpha)\hat{m}_{t-1}, \quad t = 2, ..., n, \tag{1.10}$$

and

$$\hat{m_1} = X_1 \tag{1.11}$$

is referred to as exponential smoothing, since the recursions imply that, for $T \geq 2, \hat{m}_t = \sum_{j=0}^{t-2} \alpha(1-\alpha)^j X_{t-j} + (1-\alpha)^{t-1} X_1$, a weighted moving average of $X_t, X_{t-1}, ...$ has exponentially decreasing weights (except for the last one).

Smoothing by elimination of high-frequency components. This is a technique that consists in eliminating the high-frequency components of the Fourier

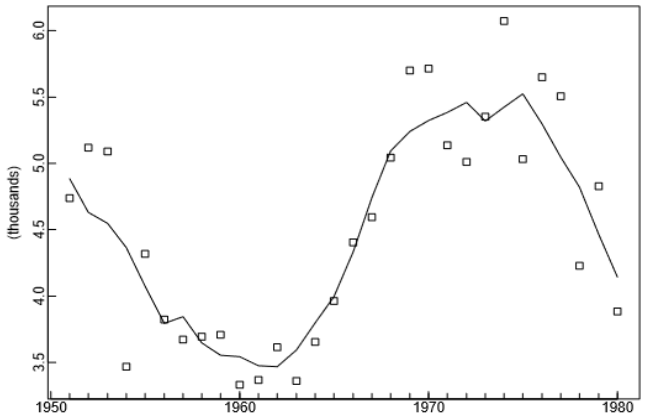

Figure 1: 5-term moving average $\hat{m}$

series expansion of the time series.
Polynomial fitting. The method of least squares here it's used to fit polynomial trends, and correlation between the residuals is taken into account when generalizing least squares estimation

The second method is the *Trend elimination by differencing.*
In this case, instead of removing the noise by smoothing as we have done in the previous steps, we eliminate the trend by differencing. Let's define the 1-lag difference operator ∇ by:

$$\nabla X_t = X_t - X_{t-1} = (1 - B)X_t, \tag{1.12}$$

where B is the backward shift operator ,

$$BX_t = X_{t-1}. \tag{1.13}$$

Powers of the operators B and ∇ are defined in an obvious way, $B^j(X_t) = X_{t-j}$ and $\nabla^j(X_t) = \nabla(\nabla^{j-1}(X_t)), j \geq 1$, with $\nabla^0(X_t) = X_t$. In this way, we can manipulate polynomials in B and ∇ in the same way as polynomial functions of real variables. If the operator ∇ is applied to a linear trend function $m_t = c_0 + c_1 t$, then we obtain the constant function $\nabla m_t = m_t - m_{t-1} = c_0 + c_1 t - (c_0 + c_1(t-1)) = c_1$.In the same way, every trend of degree k can be reduced to constant by application of the operator ∇^k. For example, if $X_t = m_t + Y_t$, where $m_t = \sum_{j=0}^{k} c_j t^j$ and Y_t is stationary with mean zero,

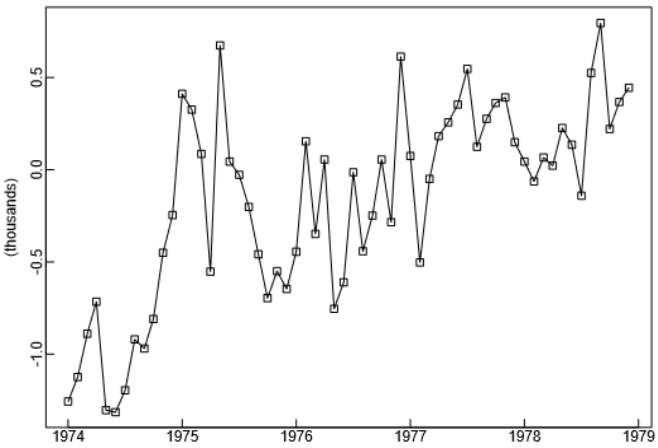

Figure 2: Example of a differenced series

application of ∇^k gives

$$\nabla^k X_t = k! c_k + \nabla^k Y_t, \tag{1.14}$$

a stationary process with mean $k!c_k$. This gives the possibility, given any sequence $\{x_t\}$ of data, of applying the operator ∇ repeatedly until we find a sequence $\{\nabla^k x_t\}$ that can be modeled as a realization of a stationary process.

1.4 Prediction

We want to stress the concept of how ACVF and ACF provide a measure of the degree of dependence among the values of a time series at different times. For this reason they play an important role when we consider the prediction of future values of the series in terms of the past and present values.
Here with a simple example we'd like to present the role of autocorrelation function. Suppose that $\{X_t\}$ is a stationary Gaussian time series and that we have observed X_n. We would like to find the function of X_n that gives us the best predictor of X_{n+h}, the value of the series after another h time units have been elapsed. By saying "best" predictor we mean the function of X_n with minimum mean squared error, for example. The conditional distribution of X_{n+h} given $X_n = x_n$ is

$$N(\mu + \rho(h)(x_n - \mu), \sigma^2(1 - \rho(h)^2)), \tag{1.15}$$

where μ and σ^2 are the mean and the variance of $\{X_t\}$. The value of the constant c that minimizes $E(X_{n+h} - c)^2$ is the conditional mean

$$m(X_n) = E(X_{n+h}|X_n) = \mu + \rho(h)(X_n - \mu). \tag{1.16}$$

The corresponding mean squared error is:

$$E(X_{n+h} - m(X_n))^2 = \sigma^2(1 - \rho(h)^2). \tag{1.17}$$

This calculation shows that, for stationary Gaussian time series, prediction of X_{n+h} in terms of X_n is more accurate as $|\rho(h)|$ become closer to 1 and as $\rho \to \pm 1$ the best predictor approaches $\mu \pm (X_n - \mu)$ and the corresponding mean squared error approaches 0. Let's notice that the assumption of joint normality of X_{n+h} and X_n played a crucial role The fact that the best linear predictor depends only on the mean and ACF of the series $\{X_t\}$ means that the predictor can be calculated without more detailed knowledge of the joint distributions. This is important because in practice it's difficult to estimate the joint distributions or, even if the distributions are known, it's difficult to compute the conditional expectations.

1.5 From ARMA to ARIMA models

The prominent models used to predict values of a time series are the ARIMA models. The class of linear time series models includes autoregressive moving-average (ARMA) models, that are a powerful tool to study stationary process.

Definition 1.4. The time series $\{X_t\}$ is a *linear process* if it has the representation

$$X_t = \sum_{j=-\infty}^{\infty} \psi_j Z_{t-j}, \tag{1.18}$$

for all t, where $\{Z_t\} \sim \text{WN}\ (0, \sigma^2)$[1] and $\{\psi_j\}$ is a sequence of constants with $\sum_{j=-\infty}^{\infty} |\psi_j| < \infty$. In terms of backwards shift operator B introduced above, the equation 1.18 can be written simply as

$$X_t = \psi(B)Z_t, \tag{1.19}$$

[1] $\text{WN}(0, \sigma^2)$ is the white noise with zero mean and σ^2 variance

where $\psi(B) = \sum_{j=-\infty}^{\infty} \psi_j B^j$. A linear process is called a *moving average* or MA(∞) if $\psi_j = 0$ for all $j < 0$,i.e, if

$$X_t = \sum_{j=0}^{\infty} \psi_j Z_{t-j}. \tag{1.20}$$

We can think to $\psi(B)$ as a linear filter which, applied to the white noise input series $\{Z_t\}$ produces the output $\{X_t\}$. It's out of the scope of this discussion but there is a theorem that establishes that a linear filter, when applied to a stationary input series, produces a stationary output series.
Let us assume that $\{X_t\}$ is a stationary series satisfying the equations:

$$X_t = \phi X_{t-1} + Z_t, \quad t = 0, \pm 1, ..., \tag{1.21}$$

where $\{Z_t\} \sim \text{WN}(0, \sigma^2)$, $\phi < 1$ and Z_t is incorrelated with X_s for each $s < t$. A first-order autoregression or AR(1) is a stationary solution $\{X_t\}$ of the equations:

$$X_t - \phi X_{t-1} = Z_t. \tag{1.22}$$

To show that such a solution exists and is the unique stationary solution, we consider the linear process defined by

$$X_t = \sum_{j=0}^{\infty} \phi^j Z_{t-j}. \tag{1.23}$$

(The coefficients ϕ^j for $j \geq 0$ are absolutely summable, since $|\phi| < 1$). It is easy to verify that the process 1.23 is a solution of 1.22 and is also stationary with mean 0 and ACVF

$$\gamma_X(h) = \sum_{j=0}^{\infty} \phi^j \phi^{j+h} \sigma^2 = \frac{\sigma^2 \phi^h}{1 - \phi^2} \tag{1.24}$$

for $h \geq 0$. In addition, it can be easily showed that 1.23 is the only stationary solution to 1.22.

Definition 1.5. $\{X_t\}$ is an *ARMA*(p, q) process if $\{X_t\}$ is stationary and for every t,

$$X_t - \phi_1 X_{t-1} - ... - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + ... + \theta_q Z_{t-q}, \tag{1.25}$$

where $\{Z_t\} \sim WN(0, \sigma^2)$ and the polynomials $(1 - \phi_1 z - ... - \phi_p z^p)$ and $(1 + \theta_1 z + ... + \theta_q z^q)$ have no common factors.

So the ARMA model is the combination of an autoregressive process of order p and a moving-average model of order q. The requirement that $\{X_t\}$ is stationary allows us to introduce that a stationary solution $\{X_t\}$ of the equation 1.25 exists and is unique if and only if

$$\phi(z) = 1 - \phi_1 z - ... - \phi_p z^p \neq 0 \text{ for all } |z| = 1. \tag{1.26}$$

In other words, a stationary solution exists if $\phi(z) \neq 0$ for all the complex z on the unit circle.

We point out above that the two measures that are crucial for studying the series are the ACVF and ACF functions. Before calculating them for the ARMA models, we introduce the concept of *casuality*:

Definition 1.6. An ARMA (p, q) process $\{X_t\}$ is casual if there exist constants $\{\psi_j\}$ such that $\sum_{j=0}^{\infty} |\psi_j| < \infty$ and

$$X_t = \sum_{j=0}^{\infty} \psi_j Z_{t-j} \text{ for all } t. \tag{1.27}$$

Causality is equivalent to the condition

$$\phi(z) = 1 - \phi_1 z - ... - \phi_p z^p \neq 0 \text{ for all } z \leq 1 \tag{1.28}$$

The equivalence of the casuality and 1.6 follows from elementary properties of power series. Let's now talk about ACF and ACVF of an ARMA process. There is more than one method to calculate the ACVF. Here we choose to multiply each side of the equations

$$X_t - \phi_1 X_{t-1} - ... - \phi_p X_{t-p} = Z_t + \theta_1 Z_{t-1} + ... + \theta_q Z_{t-q}, \tag{1.29}$$

by $X_{t-k}, k = 0, 1, 2...$, and, taking expectations on each side, we find that

$$\gamma(k) - \phi_1 \gamma(k-1) - ... - \phi_k \gamma(k-p) = \sigma^2 \sum_{j=0}^{\infty} \theta_{k+j} \psi_j, \;\; 0 < k < m, \tag{1.30}$$

and

$$\gamma(k) - \phi_1 \gamma(k-1) - ... - \phi_p \gamma(k-p) = 0 \;\; k \geq m, \tag{1.31}$$

where $m = \max(p, q+1), \psi_j := 0$ for $j < 0, \theta_0 := 1$, and $\theta_j := 0$ for $j \notin \{0, ..., q\}$. In calculating the right-hand side of 1.30 we made use of

the expansion 1.27. Equations 1.31 are a set of homogeneus linear difference equations with constant coefficients, for which the solutions are well known and are of the form:

$$\gamma(h) = \alpha_1 \chi_1^{-h} + \alpha_2 \chi_2^{-h} + ... + \alpha_p \chi_p^{-h} \quad h \geq m - p, \tag{1.32}$$

where $\chi_1, ..., \chi_p$ are the distinct roots of the equation $\phi(z) = 0$ and $\alpha_1, ..., \alpha_p$ are arbitrary constants. If we substitute the solution of 1.32 into 1.30 we obtain a set of m linear equations that uniquely determines the costants $\alpha_1, ..., \alpha_p$ and the $m - p$ autocovariances $\gamma(h),\ 0 \leq h \leq m - p$. Recall that the ACF of an ARMA process $\{X_t\}$ is the function $\rho(\cdot)$ found immediately from the ACVF $\gamma(\cdot)$ as

$$\rho(h) = \frac{\gamma(h)}{\gamma(0)} \tag{1.33}$$

Another function can be calculated while analyzing ARMA processes, the partial autocorrelation function (PACF) that is the function defined by the equations:

$$\alpha(0) = q \tag{1.34}$$

and

$$\alpha(h) = \phi_{hh}, \quad h \geq 1, \tag{1.35}$$

where ϕ_{hh} is the last component of

$$\phi(h) = \Gamma_h^{-1} \gamma_h, \tag{1.36}$$

$\Gamma_h = [\gamma(i - j)]_{i,j=1}^h$, and $\gamma(h) = [\gamma(1), \gamma(2), ..., \gamma(h)]'$.
Let's now close the discussion on time series with the ARIMA processes. A problem which frequently arose in practice and in applications is that a series of observations $\{x_1, x_2, ..., x_n\}$ is not stationary. As we have already seen above, in case of stationarity we attempt to fit an ARMA model to the data. In case of non-stationary series we have to look for a transformation of the data that generates a new series that has no deviations from stationarity and a rapidly decreasing autocovariance function. This is achieved by considering ARIMA (autoregressive integrated moving average) models, that are processes that reduces to ARMA if differenced finitely many times.

Definition 1.7. If d is a nonnegative integer, then $\{X - t\}$ is an ARIMA(p, d, q) process if $Y_t := (1 - B)^d X_t$ is a casual ARMA(p, q) process.

This definition means that $\{X_t\}$ satisfies a difference equation of the form

$$\phi^*(B)X_t \equiv \phi(B)(1-B)^d X_t = \theta(B)Z_t, \quad \{Z_t\} \sim \text{WN}(0, \sigma^2), \tag{1.37}$$

where $\phi(z)$ and $\theta(z)$ are polynomial of degrees p and q, respectively, and $\phi(z) \neq 0$ for $|z| \leq 1$. As already said, deviation from stationarity may be suggested by the graph of the series or by the sample autocorrelation function. Inspection of the graph can reveal a strong dependence of variability on the level of the series, in which case we have to transform or eliminate the dependence. If we are dealing with non stationary time series that has a slowly decaying sample ACF, we differencing it to transform the series in a process with a rapidly decreasing ACF. The degree of differencing of a time series $\{X_t\}$ was largely determined by applying the difference operator repeatedly until the sample ACF of $\{\nabla^d X_t\}$ decays quickly. The differenced time series can be modeled by a low-order ARMA(p, q) process, and hence the resulting ARIMA (p, d, q) model for the original data has an autoregressive polynomial $(1 - \phi_1 z - ... - \phi_p z^p)(1 - z)^d$ with d roots on the unit circle.

1.6 Forecasting ARIMA models

In order to find anomalies, which is our purpose, the first steps to take is to analyze the time series, forecast values and find the differences with observed values. At the moment we are able to do analysis of ARIMA processes. Let's briefly talk about forecasting values of ARIMA. If $d \geq 1$, the first and second moments EX_t and $E(X_{t+h}X_t)$ are not determined by the difference equations 1.37. We cannot determine best linear predictors for $\{X_t\}$ without further assumptions. For example, suppose that $\{Y_t\}$ is a casual ARMA(p, q) process and that X_0 is any random variable. Define

$$X_t = X_0 + \sum_{j=1}^{t} Y_j, \quad t = 1, 2... \tag{1.38}$$

So $\{X_t, t \geq 0\}$ is an ARIMA$(p, 1, q)$ process with mean $EX_t = EX_0$ and autocovariances $E(X_{t+h}X_t) - (EX_0)^2$. Our goal is to find the linear combination of $1, X_n, X_{n-1}, ..., X_1$, that forecasts X_{n+h} with minimum mean squared error. The best linear predictor in terms of $1, X_n, ..., X_1$ is denoted by P_n and has the form

$$P_n X_{n+h} = a_0 + a_1 X_n + ... + a_n X_1. \tag{1.39}$$

The values $a_0, ..., a_n$ are determined by finding the values that minimize

$$S(a_0, ..., a_n) = E(X_{n+h} - a_o - a_1 X_n - ... - a_n X_1)^2. \tag{1.40}$$

Since S is a quadratic function of $a_0, ..., a_n$ and is bounded below by zero, is it clear that there is at least one value of $(a_0, ..., a_n)$ that minimizes S and that the minimum $(a_0, ..., a_n)$ satisfies the equation

$$\frac{\partial S(a_0, ..., a_n)}{\partial a_j} = 0, \quad j = 0, ..., n. \tag{1.41}$$

Evaluation of the derivatives in the equation 1.41 gives the equivalent equations

$$E\Big[X_{n+h} - a_0 - \sum_{i=1}^{n} a_i X_{n+1-j}\Big] = 0 \tag{1.42}$$

$$E\Big[(X_{n+h} - a_0 - \sum_{i=1}^{n} a_i X_{n+1-i}) X_{n+1-j}\Big] = 0, \quad j = 1, ..., n. \tag{1.43}$$

If we use vector notation, the equations can be written

$$a_0 = \mu\Big(1 - \sum_{i=1}^{n} a_i\Big) \tag{1.44}$$

and

$$\Gamma_n \boldsymbol{a_n} = \gamma_n(h) \tag{1.45}$$

where

$$\boldsymbol{a_n} = (a_1, ..., a_n)', \quad \Gamma_n = [\gamma(i-j)]_{i,j=1}^{n}, \tag{1.46}$$

and

$$\gamma(h) = (\gamma(h), \gamma(h+1), ..., \gamma(h+n-1))'. \tag{1.47}$$

Hence,

$$P_n X_{n+h} = \mu + \sum_{i=1}^{n} a_i (X_{n+1-i} - \mu), \tag{1.48}$$

where $\boldsymbol{a_n}$ satisfies 1.45. From 1.48 the expected value of the prediction error $X_{n+h} - P_n X_{n+h}$ is zero, and the mean square prediction error is

$$E(X_{n+h} - P_n X_{n+h})^2 = \gamma(0) - 2\sum_{i=1}^{n} a_i \gamma(h+i-1) + \sum_{i=1}^{n}\sum_{j=1}^{n} a_i \gamma(i-j) a_j = \gamma(0) - \boldsymbol{a}_n' \gamma_n(h). \tag{1.49}$$

Coming back to the ARIMA models introduced at the beginning of the paragraph and using P_n to forecast values, we can write

$$P_nX_{n+1} = P_n(X_0 + Y_1 + ... + Y_{n+1}) = P_n(X_n + Y_{n+1}) = X_n + P_nY_{n+1}. \quad (1.50)$$

To evaluate P_nY_{n+1} it is necessary to know $E(X_0Y_j), j = 1, ..., n+1$ and EX_0^2. We shall assume that our observed process $\{X_t\}$ satisfies the difference equations

$$(1 - B)^s X_t = Y_t, \quad t = 1, 2, ..., \quad (1.51)$$

where $\{Y_t\}$ is a casual ARMA(p, q) process, and the random vector $(X_{1-d}, ..., X_0)$ is uncorrelated with $Y_t, t > 0$. The difference equation can be rewritten in the form

$$X_t = Y_t - \sum_{j=1}^{d} \binom{d}{j} (-1)^j X_{t-j}, \quad t = 1, 2, ... \quad (1.52)$$

Now, to make prediction, let's assume that we observed $X_{1-d}, X_{2-d}, ..., X_n$. We use P_n to denote the best linear prediction in terms of observations up to time n.
Our goal is to apply the operator P_n to each side of 1.52 (with $t = n + h$) and using the linearity of P_n to obtain

$$P_nX_{n+h} = P_nY_{n+h} - \sum_{j=1}^{d} \binom{d}{j} (-1)^j P_nX_{n+h-j}. \quad (1.53)$$

Assuming that $(X_{1-d}, ..., X_0)$ is uncorrelated with $Y_t, t > 0$, the predictor P_nX_{n+1} is obtained from 1.53 by noting that $P_nX_{n+1-j} = X_{n+1-j}$ for each $j \geq 1$. The predictor PX_{n+2} can be found from 1.53 using the previous calculated value P_nX_{n+1}. The predictors $P_nX_{n+3}, P_nX_{n+4}, ...$, can be computed recursively in the same way.
So the h-step predictor

$$g(h) := P_nX_{n+h} \quad (1.54)$$

satisfies the homogeneous linear difference equations:

$$g(h) - \phi_1^* g(h-1) - ... - \phi_{p+d}^* g(h-p-d) = 0 \;\; h > q, \quad (1.55)$$

where $\phi_1^*, ..., \phi_{p+d}^*$ are the coefficients of $z, ..., z^{p+d}$ in

$$\phi^*(z) = (1 - z)^d \phi(z). \quad (1.56)$$

The solution of 1.55 is well known from the theory of linear difference equations. If we assume that zeros of $\phi(z)$ (denoted $\chi_1, \chi_2, ..., \chi_p$) are all distinct, then the solution is

$$g(h) = a_0 + a_1 h + ... + a_d h^{d-1} + b_1 \chi^{-h} + ... + b_p \chi_p^{-h}, \quad h > q - p - d, \quad (1.57)$$

where the coefficients $a_1, ..., a_d$ and $b_1, ..., b_p$ can be determined from the $p+d$ equations obtained by equating the right-hand side of 1.57 for $q - p - d < h \leq q$ with the corresponding value of $g(h)$ computed numerically.

2 Forecasting with recurrent networks

2.1 Introduction

In our road towards anomaly detection, the first steps concern the forecast of our time series.What we see till now is a classical approach. A method like ARIMA fit a single model to each time series we have. Then we can use the model to extrapolate the time series into the future.
In cybersecurity applications (and in other areas, such as capacity management) we have many similar time series across a set of cross-sectional units: time series for server loads, requests for webpages, bytes exchange, access to relevant applications. For this cases, it's better to train a single model jointly over all the time series. In order to achieve this objective we will conjugate the classical approach with the recurrent networks.

2.2 Neural networks

Let's briefly introduce neural networks. We can represent a neural network with L layers as the composition of L functions $f_i : E \times H_i \to E_{i+1}$ where E_i, H_i and E_{i+1} are inner product spaces for all $i \in [L]$. We will refer to variables $x_i \in E_i$ as state variable, and $\theta_i \in H_i$ as parameters.
The output of a neural network for a generic input $x \in E_1$ is a function $F : E_1 \times (H_1, ..., H_L) \to E_{L+1}$ according to

$$F(x; \theta) = (f_L \circ ... \circ f_1)(x), \quad (2.1)$$

The goal of a neural network is to optimize some loss function J with respect to the parameters θ over a set of n network inputs $\mathscr{D} = \{(x_{(1)}, y_{(1)}), ..., (x_{(n)}, y_{(n)})\}$,

where $x_{(j)} \in E_1$ is the j^{th} input data point with associated response or target $y_{(j)} \in E_{L+1}$. Most optimization methods are gradient-based, meaning that we must calculate the gradient of J with respect to the parameters at each layer $i \in [L]$.
We now introduce the loss function. We will take derivatives of this loss function for a single data point $(x, y) = (x_{(j)}, y_{(j)})$ for some $j \in [n]$ and then present the error backpropagation in a concise format. Then, we will discuss how to perform gradient descent steps and how to incorporate the ℓ_2-regularization, known as *weight decay*. We write

$$x_i = \alpha_{i-1}(x) \tag{2.2}$$

for ease of notation.
Let's analyze the case of regression models. In this case, the target variable can be any generic vector of real numbers. So, for a single data point, the most common loss function to consider is the squared loss, given by:

$$J_R(x; y; \theta) = \frac{1}{2}||y - F(x; \theta)||^2 = \frac{1}{2}\langle y - F(x; \theta), y - F(x; \theta)\rangle. \tag{2.3}$$

In this case the network prediction $\hat{y}_R$ is given by the network output $F(x; \theta)$. We can calculate the gradient of J_R with respect to θ according to a theorem, the demonstration of which is out of the scope of this paper.

Theorem 2.1. *For any $x \in E_1$,$y \in E_{L+1}$, and $i \in [L]$,*

$$\nabla_{\theta_i} J_R(x; y; \theta) = \nabla^*_{\theta_i} f_i(x_i) \cdot D^* \omega_{i+1}(x_{i+1}) \cdot (\hat{y_R} - y), \tag{2.4}$$

where $x_i = \alpha_{i-1}(x)$,J_R has been defined in 2.3, $\hat{y_R} = F(x; \theta)$, $\alpha_i = f_i \circ ... \circ f_1$ is the head map and $\omega_i = f_L \circ ... \circ f_i$ is the tail map.

This implies that the derivative map above is a linear functional, i.e. $\nabla_{\theta_i} J_R(x; \theta) \in \mathscr{L}(H_i, \mathbb{R})$. Here with H_1 we denote a Hilbert space.
So loss functions define the quality of the prediction we are going to make. The choice of loss function has to be done in relation of the problem and the data we want to analyze. For example, a choice has to be done when deciding if the loss function should be symmetric. This means that a negative prediction error is judged as causing the same loss as a positive error of the same absolute value. Another example is about the convexity of the loss function, which means that the differences between high prediction errors

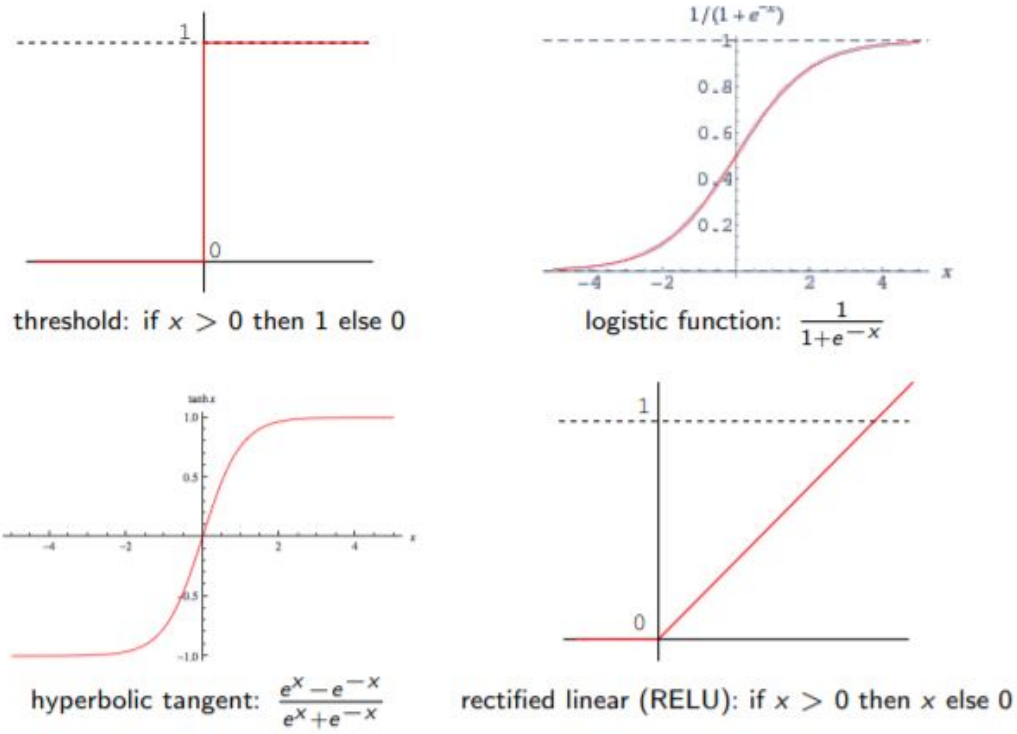

Figure 3: Some kind of activation functions

are assessed as more important than differences between small prediction errors.

A neural network is a collection of artificial neurons connected together. Neurons are organized in layers. Neural networks are made in a way that each neuron in a layer takes multiple inputs and produces a single output (that can be passed as input to other neurons). To introduce a thresholding mechanism to activate the neuron, an *activation function* is used (see figure 3 for the different kind of activation function).

In a multilayered network, when an input is taken by a neuron , it gets multiplied by a weight value. For example, a neuron with 3 inputs has 3 weight values which are adjusted during training time. The weight space is the set of all the possible values of the weights.

The *backpropagation* algorithm looks for the minimum of the error function in weight space using the method of *gradient descent*. Since this method requires computation of the gradient of the error function at each iteration

step, it's important for the error function to be continuous and differentiable. To determine the output of a neural network we use the activation function (which is used to check for neuron output value and decide if it should be fired or not). For backpropagation network the most used activation function is the *sigmoid*, a real function $s_C : \mathbb{R} \to (0, 1)$ defined by the expression

$$s_C(x) = \frac{1}{1 + e^{-cx}} \tag{2.5}$$

The constant c can be selected arbitrarily. The derivative of the sigmoid with respect to x is

$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = s(x)(1 - s(x)). \tag{2.6}$$

Many other kinds of activation functions can be chosen, the important thing is that a differentiable activation function makes the function computed by a neural network differentiable, since the network computes only function compositions. The error function also become differentiable. Following the gradient descent to find the minimum of this function we have not to find regions for which the error function is flat. As the sigmoid has positive derivative, the slope of the error function provides a greater or lesser descent direction which can be followed.
Our goal is to find the optimal combination of weights so that the network function ϕ (the composite function from input to output space) approximate a function f as closely as possible.
When an input pattern x_i from the training set is presented to the network, it produces an output $\hat{y}_i$ different in general from the target y_i. What we want is to make $\hat{y}_i$ and y_i identical. More precisely we want to minimize the error function E. The weights in the network are the only parameters that can be modified to make E as low as possible. We can thus minimize E by using an iterative process of gradient descent, for which we need to calculate the gradient

$$\nabla E = (\frac{\partial E}{\partial \omega_1}, \frac{\partial E}{\partial \omega_2}, ... \frac{\partial E}{\partial \omega_\ell}). \tag{2.7}$$

Each weight is updated using the increment

$$\Delta\omega_i = -\gamma \frac{\partial E}{\partial \omega_i} \quad \text{for} \quad i = 1, ..., \ell, \tag{2.8}$$

where γ is a learning costant, defining the step length of each iteration in the negative gradient direction.
Let's now formulate the backpropagation algorithm.
Consider a network with a single real input x and network function F. The derivative $F'(x)$ is computed in two phases:

- Feed-forward: the input x is fed into the network. The primitive functions at the nodes and their derivatives are evaluated at each node, then they are stored.
- Backpropagation: the constant 1 is fed into the ouput unit and the network is run backwards. Incoming information to a node is added and the result is multiplied by the value stored in the left part of the unit. The result is transmitted to the left of the unit. The result collected at the input unit is the derivative of the network function with respect to x.

The backpropagation step computes the gradient of E with respect to the input $\partial E/\partial \hat{y}_i \omega_{ij}$.
So after choosing the weights randomly, the backpropagation algorithm is used to compute the necessary corrections. The steps of the algorithm are:

- Feed-forward computation
- Backpropagation to the output layer
- Backpropagation to the hidden layer
- Weight updates

The algorithm is stopped when the value of the error function has become sufficiently small.

2.3 LSTM

Gradient-based learning methods have a problem: in some cases, the gradient will be small and prevents the weight from changing its values and this may stop the neural network from further training. This is known as the vanishing gradient problem. To address this problem, we introduce the LSTM (Long Short-Term Memory) network.

LSTMs ensure that a constant error is mantained to allow the RNN (recurrent neural network) to learn over long time steps, which enables it to associate problems and its effects remotely. This is particularly useful when we need more context during the analysis. In fact, RNNs are unable to learn as the gap between two information grows.
The default behavior of LSTMs is remembering information for long periods of time. LSTMs have the same chain-like structure of an RNN but, instead of having a single neural network layer, there are four which interact in a specific way. Let's start from the cell state, that is the conveyor of the information and can be view as an horizontal line that has only minor linear interactions. As the data flows inside the cell state, LSTM can add or remove information and this action is regulated by structures called gates. Gates are a way to let information through, with a sigmoid neural network layer and a pointwise multiplication operation. As already seen, the sigmoid layer outputs numbers between zero and one, describing how much of the component should be let through: zero means the block of the information, one means that all the information passes through. A LSTM has three gates to protect and control the cell state.
Let's see step by step how the information flows in the LSTM, first of all in a rough manner, then more precisely.
The focus is on what information to throw away from the cell state. This decision is done by the sigmoid layer called the "forget gate layer". So, in the cell state C_{t-1}, the sigmoid outputs a number between 0 and 1 (1 is "completely keep the information"). The next step is to decide what new information we are going to store in the cell state. This is done in two parts: a sigmoid layer called the "input gate layer" decides which values we'll update. Then a *tanh* layer creates a vector of new candidate values $\tilde{C}_t$, that could be added to the state. Then, combining the former two steps, we can create an update to the state.
So the old cell state C_{t-1} is updated into the new cell state C_t. All has been decided in the previous steps. We multiply the old state by a function and forget the thing we decided to forget earlier. Then we add $\tilde{C}_t$ multiplied by the output of the sigmoid. This is the new candidate value, scaled by how much we decide to update each state.
Finally we decide what we are going to output by running a sigmoid layer and putting the cell state trough *tanh* (the values will be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

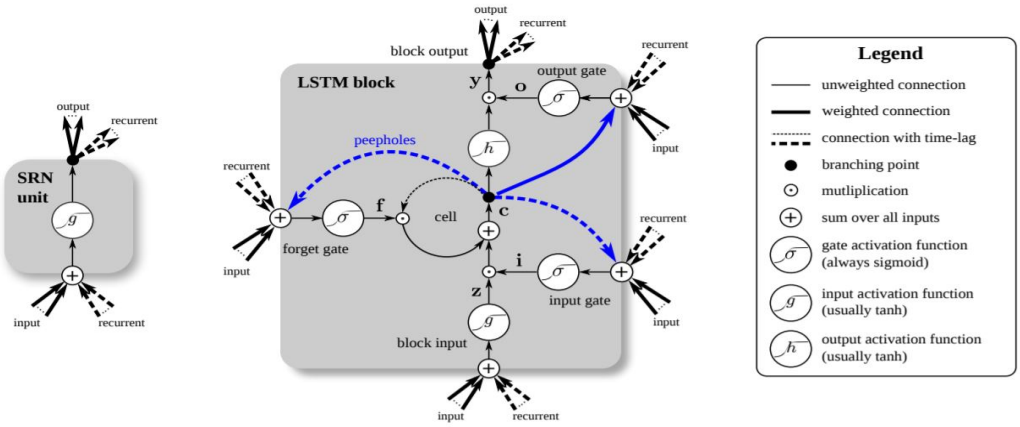

Figure 4: Schematic view of a Simple recurrent network (left) and a Long Short-Term Memory block (right) as used in the hidden layers of a recurrent neural network.

Let's be more precise.

A LSTM block has three gates (input, forget, output), a block input, a single cell (the Constant Error Carousel), an output activation function, and peephole connections. The output of the block is recurrently connected back to the block input of all the gates. In figure 4 a schematic view of a LSTM is showed.
Let x^t be the input vector at time t, N be the number of LSTM blocks and M the number of inputs. We have the following weights for an LSTM layer:

- Input weights: $\mathbb{W}_z, \mathbb{W}_i, \mathbb{W}_f, \mathbb{W}_o \in \mathbb{R}^{N \times M}$
- Recurrent weights: $\mathbb{R}_z, \mathbb{R}_i, \mathbb{R}_f, \mathbb{R}_o \in \mathbb{R}^{N \times M}$
- Peephole weights: $p_z, p_i, p_f, p_o \in \mathbb{R}^N$
- Bias weights: $b_z, b_i, b_f, b_o \in \mathbb{R}^N$

Then the vector formulas is written as:

$$
\begin{aligned}
&\bar{z} = W_z X^t + R_z y^{t-1} + b_z \\
&z^t = g(\bar{z}^t) && \textit{block input} \\
&\bar{i}^t = W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i \\
&i^t = \sigma(\bar{i}^t) && \textit{input gate} \\
&\bar{f}^t = W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f \\
&f^t = \sigma(\bar{f}^t) && \textit{forget gate} \\
&c^t = z^t \odot i^t + c^{t-1} \odot f^t && \textit{cell} \\
&\bar{o}^t = W_o x^t + R_o y^{t-1} + p_o \odot c^t + b_o \\
&o^t = \sigma(\bar{o}^t) && \textit{output gate} \\
&y^t = h(c^t) \odot o^t && \textit{block output}
\end{aligned}
$$

σ, g and h are point-wise non-linear activation functions. The logistic sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$) is used as gate activation function and the hyperbolic tangent ($g(x) = h(x) = tanh(x)$) is used as the block input and output activation function. Point-wise multiplication of two vectors is denoted by $\odot$.
Then the deltas inside the LSTM block are calculated as:

$$
\begin{aligned}
&\delta y^t = \Delta^t + R_z^T \delta z^{t+1} + R_i^T \delta_i^{t+1} + R_f^T \delta f^{t+1} + R_o^T \delta o^{t+1} \\
&\delta \bar{o}^t = \delta y^t \odot h(c^t) \odot \sigma'(\bar{o}^t) \\
&\delta c^t = \delta y^t \odot o^t \delta h'(c^t) + p_o \odot \delta \bar{o}^t + p_i \odot \delta \bar{i}^{t+1} + p_f \odot \delta \bar{f}^{t+1} + \delta c^{t+1} \odot f^{t+1} \\
&\delta \bar{f}^t = \delta c^t \odot c^{t-1} \odot \sigma'(\bar{f}^t) \\
&\delta \bar{i}^t = \delta c^t \odot z^t \odot \sigma'(\bar{i}^t) \\
&\delta \bar{z}^t = \delta c^t \odot i^t \odot g'(\bar{z}^t)
\end{aligned}
$$

Δ is the vector of deltas passed down from the layer above. E is the loss function and formally corresponds to $\frac{\partial E}{\partial y^t}$, but not includes the recurrent dependencies. The deltas for the inputs are only needed if there is a layer below that needs training, and can be computed as follows:

$$\delta x^t = W_z^T \delta \bar{z}^t + W_i^T \delta \bar{i}^t + W_f \delta \bar{f}^t + W_o \delta \bar{o}^t \tag{2.9}$$

The gradient for the weights are calculated as follows, where $\star$ can be any

of $\{\bar{z}, \bar{i}, \bar{f}, \bar{o}\}$, and $\langle \star_1, \star_2 \rangle$ is the outer product of two vectors:

$$\delta W_\star = \sum_{t=0}^{T} \langle \delta_\star^t, x^t \rangle \qquad \delta p_i = \sum_{t=0}^{T-1} c^t \odot \delta \bar{i}^{t+1}$$

$$\delta R_\star = \sum_{t=0}^{T-1} \langle \delta \star^{t+1}, y^t \rangle \qquad \delta p_f = \sum_{t=0}^{T-1} c^t \odot \delta \bar{f}^{t+1}$$

$$\delta b_\star = \sum_{t=0}^{T} \delta_\star^t \qquad \delta p_o = \sum_{t=0}^{T} c^t \odot \delta \bar{o}^t$$

2.4 DeepAR

Let's show now the principal model used we will use for anomaly detection. The model will do probabilistic forecasting, estimating the probability distribution of a time series' future given its past, by applying deep learning. Today the forecasting is done by using small groups of time series and independently estimating parameters from past observations.
In recent years, a new problem has arisen: the need to predict thousand or millions related time series. In this context, a great amount of data on past behavior of similar time series can be leveraged for making a forecast for an individual time series. Relations between time series allows fitting a more complex models.
The model used in our search for anomaly is DeepAR, that is based on autoregressive recurrent network, which learns a global model from *all time series* in the data set. The most important problem that DeepAR can solve is that multiple time series differ widely and the distribution is skewed. If for individual time series we can use ARIMA models and exponential smoothing, the use of multiple time series is difficult because of the heterogeneous nature of data. The characteristics that make probabilistic forecasting the right choice for anomaly detection is that with this model we see the full predictive distribution and, in order to obtain accurate distributions, we use a negative Binomial likelihood, which improves accuracy.
Let's denote the value of time series i at time t by $z_{i,t}$. We want to model the conditional distribution:

$$P(z_{i,t_0:T} | z_{i,1:t_0-1}, x_{i,1:T}) \tag{2.10}$$

of the future of each time series $[z_{i,t_0}, z_{i,t_0+1}, ..., z_{i,T}] := z_{i,t_0:T}$ given its past $[z_{i,1}, ..., z_{i,t_0-2}, z_{i,t_0-1}] := z_{i,1:t_0-1}$ where t_0 is the time point from which

we assume $z_{i,t}$ to be unknown at prediction time and $x_{i,1:T}$ are the covariates that are assumed to be known for all time points. We will avoid "past" and "future", we will only refer to time ranges $[1, t_0 - 1]$ and $[t_0, T]$ as the conditioning range and prediction range, respectively. During training, both ranges have to lie in the past so that $z_{i,t}$ are observed, but during prediction $z_{i,t}$ is only available in the conditioning range.
The model is based on an autoregressive recurrent network architecture. The assumption is that our distribution $Q_\Theta(z_{i,t_0:T}|z_{i,1:T}, x_{i,1:T})$ consists of a product of likelihood factors

$$Q_\Theta(z_{i,t_0:T}|z_{i,1:t_0-1}, x_{i,1:T}) = \prod_{t=t_0}^{T} Q_\Theta(z_{i,t}|z_{i,1:t-1}, x_{i,1:T}) = \prod_{t=t_0}^{T} \ell(z_{i,t}|\theta(h_{i,t}, \Theta)) \tag{2.11}$$

parametrized by the output $h_{i,t}$ of an autoregressive recurrent network

$$h_{i,t} = f(h_{i,t-1}, z_{i,t-1}, x_{i,t}, \Theta), \tag{2.12}$$

where f is a function implemented by a multi-layered recurrent neural network with LSTM cells. The model is autoregressive because takes the observations at the last time step $z_{i,t-1}$ as input, as well recurrent because the previous output of the network $h_{i,t-1}$ is fed back as input at the next time step. The likelihood $\ell(z_{i,t}|\theta(h_{i,t}))$ is a fixed distribution whose parameters are given by a function $\theta(h_{i,t}, \Theta)$ of the network output $h_{i,t}$.
Information about the observation in the conditioning range $z_{i,1:t_0-1}$ is transferred to the prediction range through the initial state h_{i,t_0-1}. Given the model parameters Θ we can obtain joint samples $\hat{z}_{i,t_0:T} \sim Q_\Theta(z_{i,t_0:T}|z_{i,1:t_0-1}, x_{i,1:T})$ through ancestral sampling: we obtain h_{i,t_0-1} by computing 2.12 for $t = 1, ..., t_0$. For $t = t_0, t_0 + 1, ..., T$ we sample $\hat{z}_{i,t} \sim \ell(\cdot|\theta(\hat{h}_{i,t}, \Theta))$ where $\hat{h}_{i,t} = f(h_{i,t-1}, \hat{z}_{i,t-1}, x_{i,t}, \Theta)$ initialized with $\hat{h}_{i,t_0-1} = h_{i,t_0-1}$ and $\hat{z}_{i,t_0-1} = z_{i,t_0-1}$. Samples from the model obtained in this way are used to calculate quantiles of the distribution of the sum of values for some time range in the future.
The likelihood should be chosen to match the statistical properties of the data. Multiple likelihood can be chosen, such as Gaussian, negative binomial or Bernoulli. We can choose mixture, because we can easily obtain samples from the distribution, and log-likelihood and its gradients can be evaluated.
Gaussian likelihood is parametrized by its mean and standard deviation,

$\theta = (\mu, \sigma)$:

$$\ell_G(z|\mu,\sigma) = (2\pi\sigma^2)^{1/2}\exp(-(z-\mu)^2/(2\sigma^2))$$
$$\mu(h_{i,t}) = w_\mu^T h_{i,t} + b_\mu \quad \text{and} \quad \sigma(h_{i,t}) = \log(1+\exp(w_\sigma^T h_{i,t} + b_\sigma)).$$

The negative binomial distribution is instead parametrize by its mean $\mu \in \mathbb{R}^+$ and a shape parameter $\alpha \in \mathbb{R}^+$,

$$\ell_{NB}(z|\mu,\alpha) = \frac{\Gamma(z+\frac{1}{\alpha})}{\Gamma(z+1)\Gamma(\frac{1}{\alpha})}\left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}}\left(\frac{\alpha\mu}{1+\alpha\mu}\right)^z$$
$$\mu(h_{i,t}) = \log(1+\log(1+\exp(w_\mu^T h_{i,t} + b_\mu))) \quad \text{and} \quad \alpha(h_{i,t}) = \log(1+\exp(w_\alpha^T h_{i,t} + b_\alpha)).$$

In this parameterization of the negative binomial distribution the shape parameter α scales the variance to the mean: $\text{Var}[z] = \mu + \mu^2\alpha$ and this fact leads to fast convergence.
So, given a data set of time series $\{z_{i,1:T}\}_{i=1,\dots,N}$ and associated covariates $x_{i,1:T}$, obtained by choosing a time range such that $z_{i,t}$ in the prediction range is known, the parameter Θ of the model, consisting of the parameters of the RNN $f(\cdot)$ as well as the parameter of $\theta(\cdot)$, can be learned by maximizing the log-likelihood

$$\mathscr{L} = \sum_{i=1}^{N}\sum_{t=t_0}^{T}\log\ell(z_{i,t}|\theta(h_{i,t})). \tag{2.13}$$

As $h_{i,t}$ is a deterministic function of the input, all quantities required to compute 2.13 are observed, so that no inference is required and 2.13 can be optimized directly via stochastic gradient descent by computing gradients with respect to Θ.
For each time series in the dataset we can generate multiple training instances by selecting windows with different start points from the original time series. The total length T is kepts as well as the relative length of the conditioning and prediction ranges. For example , if (as in our case) the total available range for a given time series ranges from 2018-10-01 to 2019-05-14, we can create training examples with $t = 1$ corresponding to 2018-10-01, 2018-10-02,2018-10-03 and so on. When choosing these windows we ensure that the entire prediction range is always covered by the available ground truth data, but we may choose $t = 1$ to lie *before* the start of the time series, for example 2018-09-01 in our case, padding the unobserved target with zeros. This allows the model to learn the behavior of "new" time series taking

into account all other available features. By augmenting the data using this window procedure, we ensure that information about absolute time is only available to the model through covariates, but not through the relative position of $z_{i,t}$ in the time series.

3 Application: anomaly detection in bank application use

Let's now see how we apply the models introduced above in cybersecurity case; obviously the applications in cybersecurity or in other fields (like capacity monitoring) can be extended to a great amount of cases.

Problem: Anomalies in logs regarding applications used by bank branches and related to bank operations are generally crucial for cybersecurity monitoring. Even a little change in the behavior of the person working on this apps can be symptom of something malicious happening, like for example a suspicious operator intent or a keylogger connected to the branch pc.

In order to circumscribe the problem and to be able to let the model run even on personal workstation, we consider only a branch with 10 operators working on sensitive apps, we collect the log starting from 2018-10-01 until 2019-05-14 and we see the model predictions day by day, starting from 2019-05-02 in order to find anomalies that can be indicator of ongoing malicious activity.

The most important time series considered are related to the aspects of the activity (all done hourly and all reduced to numeric values): general operations done by the user, specific app used, specific function of the app used, value returned from the app.

The models introduced above are used to forecast future values, see how much the hourly observed values differs from the predicted ones and, after that, find time-points with the largest negative log-likelihood. These points (if found) are the anomalies we want to detect.

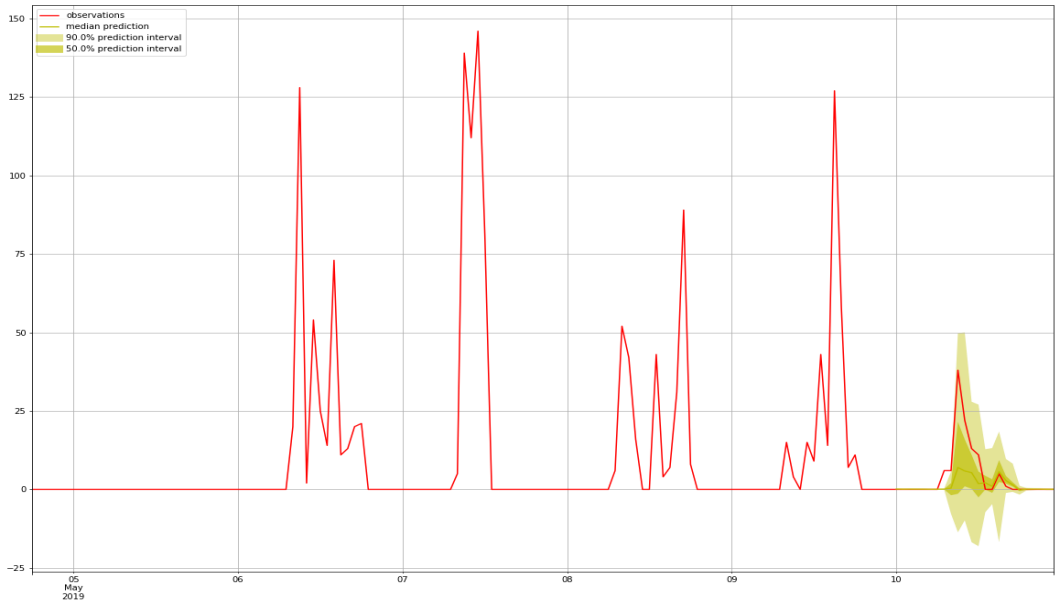

Figure 5: Normal behavior of bank operator

First of all we normalize all the features in order to have values between zero and one and to make this values suitable for our gradient-based estimator. Our prediction length, i.e. the forecast horizon, will be made of 24 hours multiplied by the days we want to predict: from the 2^{th} of May to the 14^{th} of May, so 24×13.

We will plot the 90% prediction interval, the 50% prediction interval and the median prediction. These values will be compared with the observed values in our test data in order to find suspicious behavior related to bank applications usage.

As we see in figure 5, normal user activity on the 10^{th} of May falls inside the prediction interval and it's an indicator of absence of anomalies.
In figure 6 we can clearly see anomalous behavior. In order to better evaluate the anomaly, we extend the period as in figure 7 and we see that the prediction made by our model and based on our time series is totally different from what's happening.
We introduce figure 8, where red line indicates the observed behavior found by the model on larger scale (based on the largest negative log-likelihood),

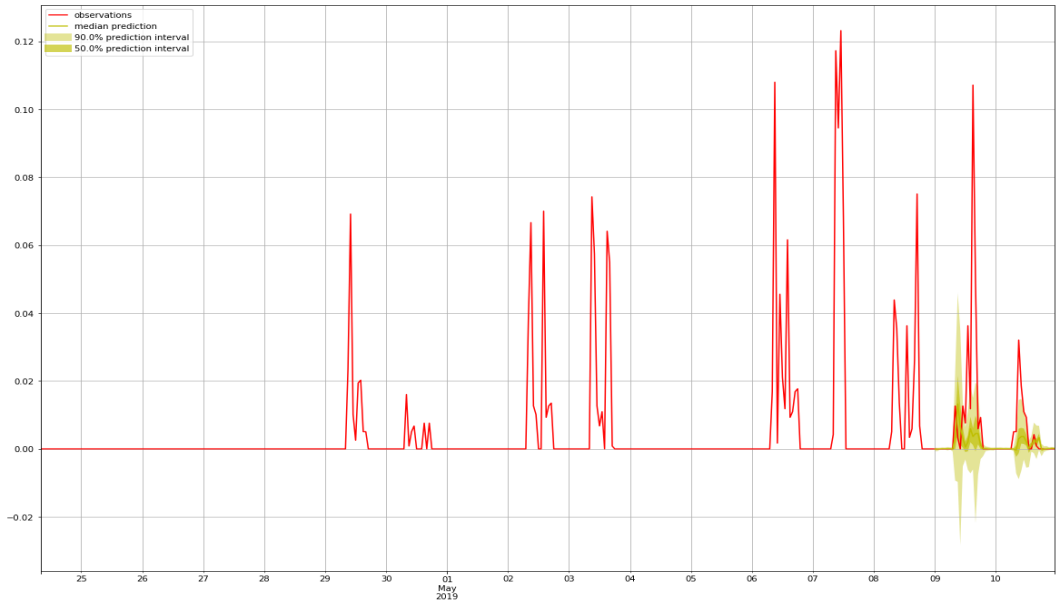

Figure 6: Anomalous behavior of bank operator with 2-days forecasting

only to clearly show how the model investigates the behavior on different dimensions and doesn't made a "naive" analysis *sic et simpliciter* on one-dimensional peaks.
Let's finally look at the evaluation of the model. We get a MSE (Mean squared error) of $0,02$, a MASE(Mean Absolute Error) of $1,13$ and an rMSE of $0,15$.

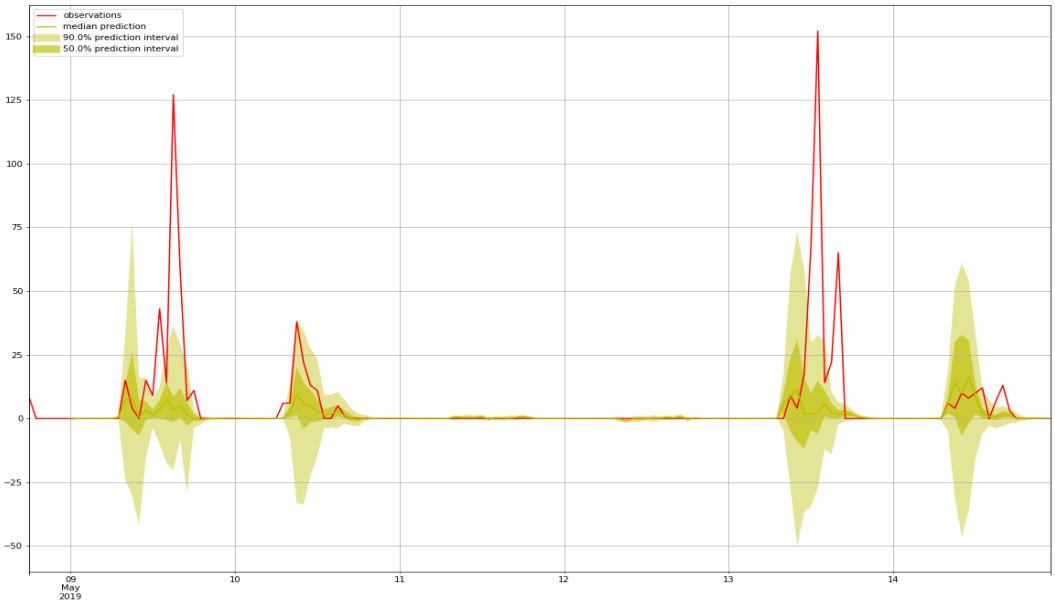

Figure 7: Anomalous behavior of bank operator with a week forecasting

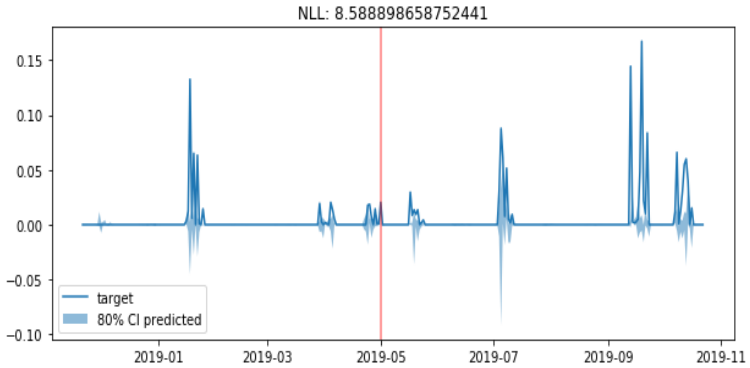

Figure 8: Anomalous behavior of bank operator during longer period. In red the anomaly. As we can see, the model looks for the anomalies on different plans, not only on the immediately visible peaks.

4 Conclusion

Anomaly detection using time series and deep learning methods is a powerful tool not only in cibersecurity but in other IT fields. In this paper the experiment has been reduced in order to make it work on personal workstation, but the results and the applications can be improved in different ways.
In view of the fact that the LSTM network can perform better with a great amount of data, a big number of series can be added to the few ones considered here. With the addition of other historical data related to other aspects of the problem we are facing, we can drastically improve the prediction power. Other ways of forecasting time series using deep learning techniques, such as Deep State or Deep Factors, can also be considered and developed.
Concerning cybersecurity, anomaly detection can help in finding suspicious behavior in nearly every part of the infrastructure. Here we have considered only bank applications used by branches, but this research for anomaly can be done in antifraud fields, on firewalls, on proxy traffic, on VPN traffic and, outside cybersercurity for example, in capacity management and performance monitoring.

References

[1] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. C. Maddix, S. Rangapuram, D. Salinas, J. Schulz, L. Stella, A. C. Türkmen, and Y. Wang, *GluonTS: Probabilistic Time Series Modeling in Python*, arXiv preprint arXiv:1906.05264 (2019).

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton, *Layer normalization*, arXiv preprint arXiv:1607.06450 (2016).

[3] Michèle Basseville, Igor V Nikiforov, et al., *Detection of abrupt changes: theory and application*, vol. 104, Prentice Hall Englewood Cliffs, 1993.

[4] Peter J Brockwell and Richard A Davis, *Introduction to time series and forecasting*, springer, 2016.

[5] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber, *Lstm: A search space odyssey*, IEEE transactions on neural networks and learning systems **28** (2016), no. 10, 2222–2232.

[6] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom, *Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding*, Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 387–395.

[7] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff, *Lstm-based encoder-decoder for multi-sensor anomaly detection*, arXiv preprint arXiv:1607.00148 (2016).

[8] Daehyung Park, Hokeun Kim, Yuuna Hoshi, Zackory Erickson, Ariel Kapusta, and Charles C Kemp, *A multimodal execution monitor with anomaly classification for robot-assisted feeding*, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2017, pp. 5406–5413.

[9] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski, *Deepar: Probabilistic forecasting with autoregressive recurrent networks*, International Journal of Forecasting (2019).

[10] Alex Sherstinsky, *Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network*, arXiv preprint arXiv:1808.03314 (2018).

[11] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka, *A multi-horizon quantile recurrent forecaster*, arXiv preprint arXiv:1711.11053 (2017).