ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

Detección de anomalías en series multivariable con modelos generativos.

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Gastón García González

en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Ingeniería Eléctrica.

Directores de Tesis

Alicia Fernández ..... Universidad de la República
Gabriel Gómez Sena ..... Universidad de la República

Tribunal

Federico Lecumberry ..... Universidad de la República
José Acuña ..... Universidad de la República
Marcelo Fiori ..... Universidad de la República
Pedro Casas ..... Austrian Institute of Technology
Rafael Molina ..... Universidad de Granada

Director Académico

Alicia Fernández ..... Universidad de la República

Montevideo
viernes 25 septiembre, 2020

Detección de anomalías en series multivariable con modelos generativos., Gastón García González.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 91 páginas.
Compilada el viernes 25 septiembre, 2020.
http://iie.fing.edu.uy/

"Siempre estoy haciendo lo que no puedo hacer, para poder aprender cómo hacerlo".

Pablo Picasso

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Mediante estas líneas quiero agradecer a todas las personas, organizaciones e institutos, que me han apoyado para llegar hasta estas instancias.

A Eugenia por transitar este camino conmigo, sin su amor y compañía todo hubiera sido más difícil. A mis padres Darío y Rosario, que se esforzaron día a día para educarnos a mi y a mi hermana Melé y darnos la libertad de hacer los que nos gusta. A mis abuelos, Aurora y Ramón, Nely y Miguel, Alicia y Miguel, por enseñarme valores de sencillez y respeto, y por todo el cariño que me han dado. A mis tíos, primos y amigos por hacer de mis logros los suyos y por acompañarme siempre.

También quiero agradecer a todas las personas que conforman el Instituto de Ingeniería Eléctrica, por la camaradería y el profesionalismo, por hacer del IIE un lugar de trabajo excepcional, donde se mezcla amistad y pasión por el trabajo.

A Alicia y Gabriel, que no solo han sido tutores de esta tesis, sino tutores en todo mi desarrollo profesional, velando siempre por mi formación y mi bienestar, y fueron fundamentales para el desarrollo de este trabajo.

Quiero agradecer a la universidad de Granada (UGR) por abrirme sus puertas y en especial a los chicos del DB3 por su amistad y buena onda.

También agradecer a la empresa Austrian Institute of Technology (AIT), por darme la oportunidad de trabajar con ellos, y por todo lo aprendido allí.

Este trabajo también hubiese sido muy difícil llevarlo a cabo sin el apoyo de la empresa Telefónica, la Agencia Nacional de Investigación, y la Comisión Sectorial de Investigación Científica.

Esta página ha sido intencionalmente dejada en blanco.

A Eugenia, mis padres, familia y amigos.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

La detección de anomalías es un campo de estudio relevante para muchas aplicaciones y contextos. En el monitoreo de sistemas, la recopilación de múltiples variables es esencial para tener un conocimiento del estado del sistema y resolver a tiempo eventuales problemas. Un análisis eficiente de anomalías puede ser útil para detectar problemas de rendimiento, fallas, ataques externos e intentos de fraude.

Aunque la detección de anomalías en series temporales es un área de investigación madura, la aparición de grandes plataformas de datos que permiten el procesamiento de cantidades masivas y diversas de datos, junto con la reciente gran exploración científica de nuevas herramientas para aplicación de aprendizaje profundo, plantean nuevas oportunidades y desafíos para investigar en el tema. En particular, la detección de anomalías en series mutivariables es un desafío, ya que generalmente los métodos de detección tienen dos esquemas: el análisis univariable, ejecutando un detector independiente para cada serie de tiempo, o el análisis multivariable, tomando a cada instante de tiempo de manera independiente. En este trabajo se plantea la idea de monitorear todas las series de un sistema con un solo modelo teniendo en cuenta la relación temporal. Para esto se recurrió al uso de modelos generativos no-supervisados basados en redes neuronales, los cuales han demostrado una gran capacidad para aprender la distribución de datos complejos. Además, el uso de estas herramientas ayuda a resolver otros dos grandes problemas en la detección de anomalías que son: el alto desequilibrio entre los datos normales y anómalos, y la falta de etiquetas para fines de aprendizaje y validación.

Se implementaron dos métodos, el primero basado en el error de reconstrucción utilizando *Variational Auto-Encoders (VAE)*, y el segundo utilizando redes recurrentes entrenadas bajo el enfoque de las *Generative Adversarial Networks (GAN)*, explotando no solo las propiedades generativas, sino también las discriminativas.

Como un aporte importante con respecto al estado del arte, en este trabajo se logra visualizar tanto la capacidad de detección de los métodos como la capacidad de generación que es la base de los mismos. Las evaluaciones fueron hechas en dos conjuntos diferentes de datos reales, uno propio y otro público, obteniéndose muy buenos resultados. Las implementaciones fueron realizadas con la librería *keras*, logrando que la arquitectura del código[1] sea compacta y sencilla de entender.

[1]El código se encuentra disponible en: `https://github.com/GastonGarciaGonzalez/GAN_and_VAE_for_anomaly_detection_in_multivariate_time-series.git`

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Agradecimientos III

Resumen VII

1. Introducción 1

2. Marco teórico 3
2.1. Series temporales . . . 3
2.2. Definición de anomalías . . . 3
2.3. Detección de anomalías . . . 5
2.4. Modelos generativos . . . 6
2.4.1. VAEs . . . 8
2.4.2. GANs . . . 9
2.4.3. RNNs . . . 10
2.4.4. Optimización de parámetros . . . 12
2.5. Métricas . . . 13
2.5.1. Discrepancia Media Máxima (MMD) . . . 13
2.5.2. Métricas para evaluar la clasificación . . . 14

3. Enfoque: Métodos para la detección de anomalías 15
3.1. Trabajos relacionados . . . 16
3.2. Métodos implementados . . . 18
3.2.1. Método con VAE . . . 19
3.2.2. Método con GAN . . . 21

4. Arquitectura, Implementación y Conjuntos de datos 27
4.1. Arquitectura para el método con GAN . . . 27
4.2. Arquitectura para el método con VAE . . . 29
4.3. Pre-procesamiento . . . 30
4.4. Conjuntos de datos . . . 30
4.4.1. SWaT. . . . 30
4.4.2. Tel2020 . . . 31

5. Análisis y resultados 35
5.1. Generación de series . . . 35
5.2. Detección . . . 47

5.3. Validación de los métodos 51
5.3.1. Método con GAN 52
5.3.2. Método con VAE 56

6. Conclusiones y Trabajo a futuro. **61**

A. "Trucos" para el entrenamiento de las GANs **65**

B. Herramientas para el aprendizaje profundo **67**
B.1. Funciones de activación 67

Referencias **69**

Índice de tablas **72**

Índice de figuras **74**

Capítulo 1

Introducción

Las series temporales se pueden encontrar en diferentes áreas de estudio, como la economía, el clima, la industria, las comunicaciones, entre varias. Estas generalmente son utilizadas para representar la evolución temporal de las diferentes variables que componen un sistema. Como por ejemplo en una estación climática, estas variables pueden ser la humedad, la temperatura, la velocidad del viento, etc. En un un sistema de telefonía, como se verá más adelante, son relevantes por ejemplo la cantidad, la duración o el costo de las llamadas por minuto. Esta representación ayuda a tener un registro del comportamiento de las variables, observar cómo se están comportando en el instante actual y en algunos casos proyectar cómo será el comportamiento en el futuro.

Uno de los casos de estudio más importantes de las series temporales, y el cual se aborda en esta tesis, es la detección de anomalías. Detectar comportamientos inusuales que pueden estar vinculados a fallas internas del sistema o ataques de externos.

Actualmente existen diversas plataformas para el manejo y almacenamiento de grandes volúmenes de datos, que permiten monitorear decenas o centenas de variables a la vez. Esto requiere de una detección automática y rápida, que permita determinar en qué instante y variable se produjo la anomalía y así evitar que los problemas se prolonguen en el tiempo o pasen a mayores.

Los métodos tradicionales de detección generalmente están pensados para atender series univariable, lo que hace muy difícil escalar la detección en sistemas con múltiples variables donde todas se comportan de manera diferente. Por otro lado, existen otros métodos pensados para una gran cantidad de variables, pero estos tratan los datos de manera independiente para cada instante de tiempo, sin tener en cuenta la relación temporal. Este enfoque empobrece la detección ya que se pierde la información del contexto temporal, que muchas veces determina si un valor es anómalo o no.

Todas las variables de un sistema se pueden ver como una sola serie temporal multivariable que determina la evolución de todo el sistema a lo largo del tiempo. El trabajo que se describe a continuación surge de la idea de desarrollar y probar métodos de detección para este tipo de series de manera que sólo se requiera de ajustar un modelo para monitorear todas las distintas variables de un sistema,

facilitando la puesta en producción en grandes sistemas de monitoreo. Para esto se recurrió al aprendizaje profundo, implementándose dos métodos diferentes de detección, ambos enfocados a aprender el comportamiento de las series cuando el sistema se encuentra funcionando de manera normal mediante modelos generativos. El primero se basa en *Variational Auto-Encoders* (VAEs) para capturar dicho comportamiento y así mediante la reconstrucción de pequeños segmentos de serie evaluar el apartamiento de dicho comportamiento. El segundo es más complejo y utiliza *Recurrent Neural Networks* (RNNs), las cuales son redes especialmente pensadas para trabajar con secuencias de datos. En este método dos RNNs son entrenadas bajo el enfoque de las *Generative Adversarial Networks* (GANs), lo que permite obtener dos herramientas para la detección secuencial, una generativa y otra discriminativa.

Este documento está organizado de la siguiente manera. En el capítulo 2 se hace una breve introducción a las series temporales y la detección de anomalías, se describen las distintas herramientas del aprendizaje profundo utilizadas, y se definen las métricas que se usarán para medir el desempeño de los métodos. En el capítulo 3 se comentan algunos trabajos relacionados con las ventajas y desventajas que estos presentan, y se explica el funcionamiento de los dos métodos implementados. Luego en el capítulo 4, se describe la implementación de los métodos, la arquitectura de las redes, y se hace una descripción de los datos utilizados. En el capítulo 5 se hace una análisis del funcionamiento de los métodos a través de diferentes pruebas, y se presentan y discuten los resultados del desempeño en la detección sobre los conjuntos de datos. Por último en el capítulo 6 se desarrollan las conclusiones que surgieron a partir de lo aprendido y de los resultados obtenidos, y se plantean los diferentes trabajos a futuro que se pueden realizar a partir de esta investigación.

Capítulo 2

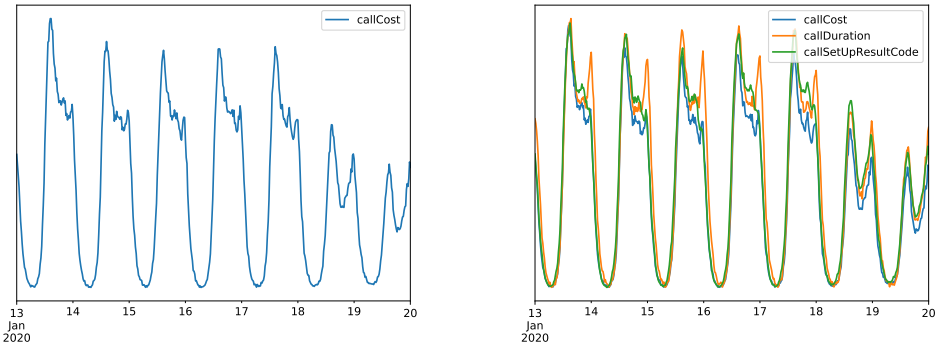
Marco teórico

2.1. Series temporales

Una serie temporal es una sucesión de uno o más valores ordenadas cronológicamente, las cuales se pueden clasificar como univariables o multivariables [4]. Las primeras corresponden a una sola variable por lo tanto para cada instante de tiempo t se obtiene un solo valor, como puede ser la temperatura de una caldera, o como se muestra en la figura 2.1.a el costo de las llamadas de una central telefónica. Las segundas corresponden a más de una variable, por lo tanto para cada instante t devuelve un vector de valores como pueden ser las medidas de un acelerómetro que mide la aceleración en los tres ejes $[x(t), y(t), z(t)]$ para cada segundo, como también la combinación de variables de un solo sistema como se muestra en la figura 2.1.b, donde además del costo se agregan otras variables como la duración de las llamadas.

2.2. Definición de anomalías

En procesamiento de datos, una anomalía es una observación o varias dependiendo del contexto, que no se ajustan a una noción bien definida de comportamiento normal. Por lo tanto, un enfoque directo de detección de anomalías es definir una región que represente un comportamiento normal y declarar cualquier observación que no pertenezca a la misma como una anomalía [4]. Según este mismo artículo, existen tres tipos de anomalías: puntuales, contextuales, y colectivas. Si una instancia de datos se consideran anómala con respecto a todo el resto de los datos, entonces la instancia se denomina anomalía puntual. Por ejemplo en la figura 2.2, las muestras o_1 y o_2 son anomalías puntuales si se toman como normales las muestras de los conjuntos N_1 y N_2. Por otro lado las anomalías contextuales, son aquellos datos que dado el contexto en que se producen son tomados como anómalos, pero en otro caso podrían ser tomados como normales. En el ejemplo de la figura 2.3 el costo de las llamadas en los tiempos t_1 y t_2 son el mismo, sin embargo uno es normal y el otro es anómalo respectivamente. El tercer y último tipo son las anomalías colectivas. Si una colección de instancias de datos relacionadas



(a) Univariable. (b) Multivariable.

Figura 2.1: Ejemplos de una serie univariable (a) y otra multivariable (b) extraídos del conjunto Tel2020 (Sección 4.4.2).

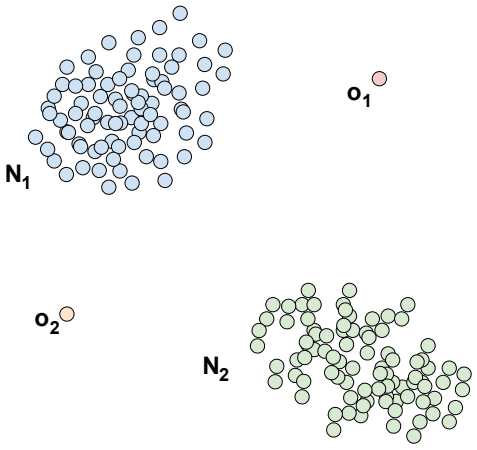


Figura 2.2: Ejemplo de dos anomalías en un problema de dos dimensiones.

es anómala con respecto a todo el conjunto de datos, se las denomina como una anomalía colectiva. Las instancias de datos individuales en una anomalía colectiva pueden no ser anomalías en sí mismas. En la figura 2.4 se puede ver un ejemplo sobre la misma serie del ejemplo anterior, pero que esta vez presenta una anomalía donde se mantiene un mismo valor para una colección de datos a partir de t_3, sin embargo si se ve cada dato de esta colección por separado no se los vería como valores anómalos.

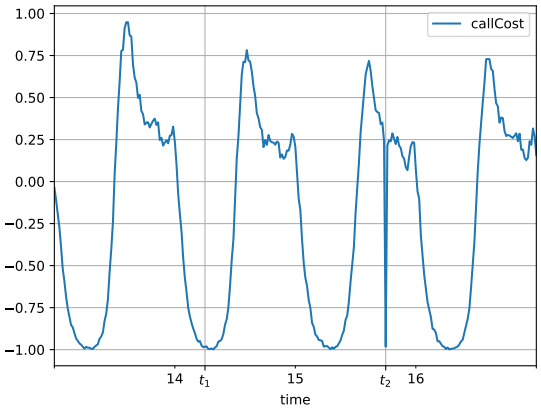


Figura 2.3: Ejemplo de una anomalía contextual en una serie univariable, que muestra que un mismo valor dependiendo del contexto es anómalo o no.

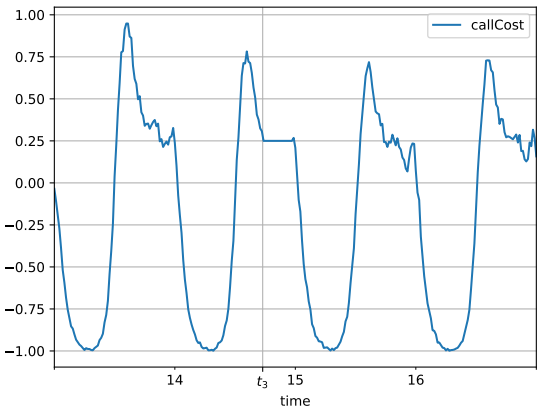


Figura 2.4: Ejemplo de una anomalía colectiva en una serie univariable, donde lo anómalo es que en varios instantes consecutivos se presente el mismo valor.

2.3. Detección de anomalías

Para la detección de anomalías en general existen diferentes métodos, y la elección de estos depende del tipo de datos con el que se va a trabajar: series temporales, imágenes, texto, etc. Pero además, es muy importante saber si los datos están etiquetados o no. Las etiquetas indican cuáles de los datos del conjunto son anómalos. Para que estas etiquetas tengan sentido tienen que haber sido marcadas por un profesional que entienda los datos con los que se están trabajando, por lo que tener una base de datos etiquetada puede ser muy costoso. Algunos trabajos científicos como [21], y [25] para obtener una base etiquetada capturan datos de un sistema donde estén seguros que el mismo está trabajando de forma normal, y lo atacan desde el exterior en intervalos de tiempo preestablecidos. De esta

forma es posible etiquetar los datos utilizando el tiempo, donde se etiquetan como anómalos a todos los datos dentro de los períodos de ataque. Esto es muy útil pero presenta algunas desventajas. Si el sistema es un sistema real, cabe la posibilidad que dentro de las muestras tomadas como normales haya anomalías producto de fallas no intencionales. Además generar los ataques de forma manual hace muy difícil cubrir todas las posibilidades de anomalías.

Dependiendo de si el conjunto de datos con el que se va a trabajar cuenta con etiquetas o no, es qué tipo de método se va aplicar para hacer la detección. Por esto, una forma de clasificar los métodos es bajo las categorías: supervisados, semisupervisados, y no supervisados.

- Detección de anomalías de forma supervisada. En este caso se cuenta con una base de datos donde cada uno está etiquetado como anómalo o normal. Un típico enfoque es utilizar métodos de clasificación utilizando estas dos etiquetas como clases. Un problema que se presenta para estos métodos es el desbalance entre clases. Dado que las anomalías como tal son eventos raros, lo más común es que se tengan muchos más datos normales que anómalos. Este desbalance hace que la probabilidad de ocurrencia de una muestra de la clase normal sea mucho más grande que una de la clase anómala, por lo que el clasificador puede quedar con un sesgo hacia la clase normal. Existen diferentes técnicas para balancear los conjuntos de datos, algunas de ellas se pueden ver en [16], [14]. Las *Generative Adversarial Networks* (GAN), método que veremos en detalle más adelante, son utilizadas también para generar nuevas muestras anómalas a partir de las que ya se tienen para balancear las bases de datos, ejemplo de esto es el trabajo realizado en [6].

- Detección de anomalías de forma semisupervisada. En este caso sólo se conoce un fragmento de los datos donde todos están etiquetados como normales. El enfoque típico es usar un método para construir un modelo que se ajuste a estos datos, para luego hacer la detección dependiendo de cómo se ajusten a este modelo los datos no vistos. Todos los métodos analizados en esta tesis, están pensados bajo este enfoque como se detalla en el siguiente capítulo.

- Detección de anomalías de forma no supervisada. Por último están los métodos utilizados cuando no se tiene ninguna información sobre los datos. Las técnicas en esta categoría hacen la suposición implícita de que las instancias normales son mucho más frecuentes que las anómalas y por lo tanto están dentro de un conjunto denso, mientras que las anómalas se encuentren en una zona esparsa del domino de los datos. Es por esto que los métodos en estos casos están enfocados a la distancia entre los datos, como K vecinos más cercanos, como se ve en [7] o *clustering* como se utilizó en [19].

2.4. Modelos generativos

Este trabajo está enfocado a la detección de anomalías mediante métodos semisupervisados, donde se usan modelos generativos de aprendizaje profundo de ma-

nera de aprender el comportamiento normal de los datos.

Dependiendo del método, los modelos generativos son modelos que se entrenan con el objetivo de dado un conjunto de datos reales $X = \{x_1, x_2, ..., x_N\}$, aprender a generar muestras sintéticas $X^* = \{x_1^*, x_2^*, ..., x_N^*\}$ que sean difíciles de diferenciar con las del conjunto X. O en otro caso el objetivos es suponiendo que las muestras X proviene de una distribución de probabilidad $p_X(x)$, aprender una distribución $p_{model}(x)$ similar a la de los datos reales. El problema es que previo al aprendizaje profundo, lograr semejanza entre los datos sintéticos y los reales, cuando estos últimos eran datos complejos y de gran dimensión era muy difícil. Es a partir de la popularización del aprendizaje profundo que los modelos generativos son capaces de conseguir resultados sorprendentes como la generación de rostros sintéticos muy similares a los reales, o la generación de nuevos gestos faciales o movimientos a partir de muy pocas imágenes de una persona. Las herramientas que aparecen generalmente detrás de estos resultados son: Variational Auto-Encoders (VAE) y Generative Adversarial Networks (GAN). En esta sección se explica qué son estas herramientas, cómo funcionan.

Espacio de latencia

Uno de los métodos determinísticos más usados para reducir la dimensión de los datos es *Principal Component Analysis* (PCA). Asumiendo que las muestras $\boldsymbol{x} \in \mathbb{R}^N$ son bien representadas por una factorización lineal, PCA consiste en una proyección lineal sobre un espacio de menor dimensión $\boldsymbol{z} \in \mathbb{R}^M$ $(M < N)$. Esto generalmente se utiliza para reducir la complejidad de los problemas, o simplemente para poder visualizar los datos sin perder demasiada información del espacio original. A las proyecciones $\boldsymbol{z}$ se las llama variables latentes ya que no fueron observadas sino que son inferidas por datos que sí lo fueron, y al espacio al que éstas pertenecen se lo llama espacio latente. Luego mediante otra proyección lineal a partir de las variables latentes, se puede volver al espacio real como muestras $\boldsymbol{x}^* \in \mathbb{R}^N$ equivalentes a las muestras originales.

Una variante de esta herramienta es *Probabilistic PCA* (PPCA). En este caso se asume que se tiene información a priori sobre la distribución de las variables latentes $p(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \mathbb{I})$ y que las muestras observadas $\boldsymbol{x}$ dependen de manera condicional de éstas, $p(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{z}, \sigma\mathbb{I})$. Una vez calculados los parámetros $\boldsymbol{W}$, $\boldsymbol{\mu}$, y σ, por el principio de máxima verosimilitud, se puede calcular la distribución a posteriori $p(\boldsymbol{z}|\boldsymbol{x})$ que también es una distribución normal. Esto tiene la ventaja sobre PCA de que por cada muestra observada se obtiene toda una distribución en el espacio de latencia y no un vector determinísitco. Pero lo interesante de esta herramienta es que una vez calculados los parámetros, se tiene un modelo probabilístico de la distribución de los datos observados $p(\boldsymbol{x})$. Donde a partir de ruido Gaussiano, como son las variables latentes $\boldsymbol{z}$, se pueden obtener muestras sintéticas $\boldsymbol{x}^*$ similares a las observadas. Esto último le da un enfoque generativo a la herramienta PPCA. De igual forma el modelo que se obtiene es lineal, lo cual para datos complejos generalmente no es suficiente. Los métodos y modelos que se describen a continuación, utilizan este mismo enfoque de suponer que los

datos observados están condicionados a variables que provienen de espacios más pequeños, pero utilizando redes neuronales para aprender el comportamiento o la distribución de los mismos.

2.4.1. VAEs

Variational Auto-Encoders (VAEs) [18] son una variante dentro de la familia de los *Auto-Encoders* (AEs). Estos últimos son redes neuronales que intentan copiar la entrada a la salida. Esta redes constan de dos partes, la primera llamada *Encoder* es una función de $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{z}$ que lleva la muestras a un espacio latente y la segunda llamada *Decoder* es una función que reconstruye las muestras $\boldsymbol{x}^* = g_{\boldsymbol{\theta}}(\boldsymbol{z})$. Si un modelo logra simplemente aprender a reconstruir el conjunto de datos de entrada $g(f(x)) = x$, entonces no es especialmente útil. En cambio, los AEs están diseñados para no poder aprender a copiar perfectamente, sino que están restringidos a aprender propiedades útiles de los datos [11]. Los AEs solamente son capaces de reconstruir los datos de entrada al igual que PCA. Por otro lado los VAEs, están pensados para generar muestras a partir de aproximar la distribución real $p(x)$ como PPCA, pero utilizando el enfoque de los AEs.

Los VAEs también asumen que la distribución de las variables latentes es $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{II})$ y que las muestras observadas dependen condicionalmente de éstas $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$, con el objetivo de aproximarse a la distribución de los datos reales $p(\boldsymbol{x})$. Para eso se pretende maximizar en $\boldsymbol{\theta}$ la función de verosimilitud $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta})$ de la ecuación 2.1, para un conjunto de datos de entrenamiento $\boldsymbol{X} = \{\boldsymbol{x}_1, .., \boldsymbol{x}_n\}$.

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}_i)) \tag{2.1}$$

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) d\boldsymbol{z} \tag{2.2}$$

Para maximizar la ecuación 2.1 se necesita calcular la distribución $p_{\boldsymbol{\theta}}(\boldsymbol{x})$, que se muestra en la ecuación 2.2. El problema es que esta integral no se puede calcular simplemente, ya que se requiere mucho poder y tiempo de cálculo. Por lo que el método VAE propone maximizar un cota inferior de la verosimilitud. Esta cota es la que se muestra en la ecuación 2.3, donde $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ es un distribución normal $\mathcal{N}(\mu_{\boldsymbol{z}|\boldsymbol{x}}, \sigma_{\boldsymbol{z}|\boldsymbol{x}})$, que aproxima la *posterior* $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, y $D_{KL}(q||p)$ es la divergencia de *Kullback-Leibler* la cual es una función positiva que mide la discrepancia entre dos distribuciones. La aproximación es necesaria dado que como no se conoce $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ no se puede calcular $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$.

$$\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\phi}) = E_{\boldsymbol{z}}\left[\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}_i|\boldsymbol{z}))\right] - D_{KL}\left(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_i)||p(\boldsymbol{z})\right) \tag{2.3}$$

Asumiendo que $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ también es un distribución normal $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{x}|\boldsymbol{z}}, \sigma_{\boldsymbol{x}|\boldsymbol{z}})$, se utiliza la parte de la red que corresponde al *Decoder* para calcular los parámetros

de esta distribución. Por otro lado el *Encoder* será el encargado de calcular los parámetros de la distribución $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. Luego la red compuesta por el *Encoder* y el *Decoder*, se entrena optimizando los parámetros que maximicen a $\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\phi})$.

Una vez finalizado el entrenamiento, el modelo del *Decoder* que representa la distribución $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ es capaz de devolver muestras sintéticas $\boldsymbol{x}^*$ que se parezcan a las muestras de $\boldsymbol{X}$, simplemente a partir de ruido Gaussiano $\mathcal{N}(\mathbf{0}, \mathbb{I})$.

2.4.2. GANs

Un nuevo sistema para estimar modelos generativos, fue presentado en [12], *Generative Aversarial Networks* (GANs). A través de un proceso de confrontación se entrenan simultáneamente dos redes neuronales: el Generador (G) la cual es un red que busca captura la distribución de los datos de entrenamiento, y el Discriminador (D) el cual es un red discriminativa que estima la probabilidad de que una muestra provenga de los datos de entrenamiento y no de G. El proceso de confrontación se da cuando G intenta aproximarse a la distribución de los datos tratando de "engañar" a D a la vez que éste se entrena para descubrirlo. La distribución capturada por el generador p_g sobre el espacio de las muestras $\boldsymbol{X}$, la aprende a partir de una distribución de entrada a priori $p(\boldsymbol{z})$, por lo tanto el generador es una función de mapeo $G(\boldsymbol{z}, \boldsymbol{\theta}_g)$. Generalmente en la práctica esta función es un red *perceptron* multicapa con parámetros $\boldsymbol{\theta}_g$. También para $D(\boldsymbol{x}, \boldsymbol{\theta}_d)$ se usa este tipo de red, donde la salida es un simple escalar $D(\boldsymbol{x})$ que es la probabilidad de que $\boldsymbol{x}$ provenga de los datos reales y no de p_g. Cuando se entrena a D para maximizar la correcta asignación de las etiquetas de ambos tipos de datos, a la misma vez se entrena G para minimizar $log(1-D(G(\boldsymbol{z})))$, por lo que la función de valor $V(D, G)$ queda determinada como:

$$\min_G \max_D V(D,G) = \mathbf{E}_{\boldsymbol{x}\sim p_{data}}\left[log\Big(D(\boldsymbol{x})\Big)\right] + \mathbf{E}_{z\sim p_g}\left[log\Big(1 - D\big(G(\boldsymbol{z})\big)\Big)\right] \quad (2.4)$$

En el pseudo código 1, se muestra de una manera simple cómo es el procesamiento de entrenamiento de este tipo de modelos.

Una vez finalizado el entrenamiento, se espera que $G(\boldsymbol{z})$ sea capaz de generar muestras sintéticas, las cuales sean muy difícil de diferenciar de las reales. Mientras que de $D(\boldsymbol{x})$ se espera que solo devuelva probabilidades cercanas a 1 cuando los datos solamente provengan de la distribución real.

Algorithm 1: Entrenamiento de una GAN.

for *Un número de epocas* **do**
  for *k pasos* **do**
    Se toman m muestras $\{\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)}\}$ del prior $p_g(\boldsymbol{z})$;
    Se toman m muestras $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)}\}$ de la distribución real $p_{data}(\boldsymbol{x})$;
    Con los parámetros de G fijos se actualizan los parámetros del discriminador por ascenso por gradiente:

$$\nabla_{\boldsymbol{\theta}_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$$

    ;
  end
  Se toman m muestras $\{\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)}\}$ del prior $p_g(\boldsymbol{z})$;
  Con los parámetros de D fijos, se actualizan los parámetros del generador por descenso por gradiente:

$$\nabla_{\boldsymbol{\theta}_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

  ;
end

2.4.3. RNNs

Muchos datos existen en forma de secuencias, las oraciones son secuencias de palabras, los videos son secuencias de imágenes, y las series temporales son secuencias de valores. Todos estos tienen una cierta relación de contexto, las muestras actuales son dependientes de las anteriores, y las futuras serán dependientes de las actuales. La idea detrás de las *Recurrent Neural Networks* (RNN) [11] es hacer uso de esta información secuencial, a diferencia de las redes neuronales tradicionales que suponen que todas las entradas (y salidas) son independientes entre sí.

Las RNN se denominan recurrentes porque realizan la misma tarea para cada elemento de una secuencia, aplicando el mismo vector de pesos para cada paso. Además la salida depende de los cálculos anteriores. Esta formulación recurrente da como resultado el intercambio de parámetros a través de un gráfico computacional muy profundo.

Estas redes operan con secuencias de vectores $\boldsymbol{x}^{(t)}$ como entrada donde t indica la posición del mismo dentro de la secuencia de largo T ($t : [1, T]$). La forma en cómo se conectan, intercambian y actualizan sus parámetros los elemento de la red, es lo que se le llama propagación hacia adelante. Existen diferentes configuraciones, la mas común y la que se usa en este trabajo es la que se bosqueja en la figura 2.5,

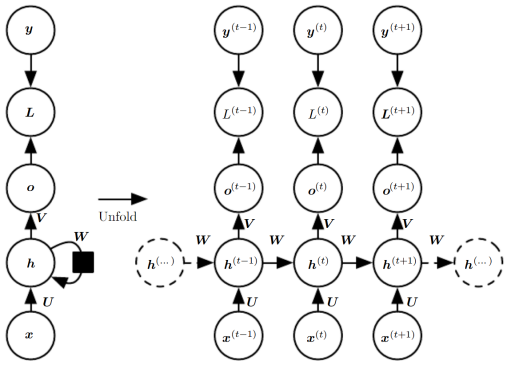


Figura 2.5: Grafo computacional de como las RNN comparten los parámetros a lo largo de la secuencia. Imagen extraída de [11].

donde se produce una salida $\boldsymbol{o}^{(t)}$ para cada paso t, y la conexión recurrente es a través de los vectores ocultos $\boldsymbol{h}^{(t)}$. Para cada tiempo t la propagación hacia adelante también consiste en calcular las siguientes ecuaciones de actualización:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \tag{2.5}$$

$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{a}^{(t)}) \tag{2.6}$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \tag{2.7}$$

$$\hat{\boldsymbol{y}}^{(t)} = softmax(\boldsymbol{o}^{(t)}) \tag{2.8}$$

El vector de estados ocultos $\boldsymbol{h}^{(t)}$ (2.6) es el encargado de transmitir al paso siguiente, toda la información del paso actual y de todos los anteriores. Para esto depende del vector $\boldsymbol{a}^{(t)}$ que es una combinación lineal de la entrada actual $\boldsymbol{x}^{(t)}$ y el estado anterior $\boldsymbol{h}^{(t-1)}$, más un vector *bias* $\boldsymbol{b}$. Luego a este se le aplica una función de activación que generalmente se utiliza tangente hiperbólica (B.1) para obtener el nuevo vector de estado. Una vez concentrada toda la información en $\boldsymbol{h}^{(t)}$ la salida pondera a éste por un vector de pesos más otro *bias* $\boldsymbol{c}$, para luego utilizar otra función de activación, la cual puede ser la función *softmax* (B.1) si se espera una predicción. Aquí los parámetros de la red a optimizar son ambos *bias*, b y c, y las matrices de pesos U, V y W que son las matrices que conectan: entrada a capa oculta, capa oculta a la salida, y capa oculta a capa oculta, respectivamente. Estos parámetros son los mismos a lo largo de toda la secuencia como se muestra en la figura 2.5.

Una vez que se aplicaron las ecuaciones de actualización para toda la secuencia, se comparan las salidas estimadas $\hat{\boldsymbol{y}}^{(t)}$ con las salidas reales $\boldsymbol{y}^{(t)}$ a través de la función de pérdida, la cual se quiere minimizar hallando los parámetros óptimos. Para converger a dicha solución se aplica un algoritmo de optimización (los cuales se explican en la sección 2.4.4) para el cual es necesario hallar el gradiente de la función de pérdida. Así como existe la propagación hacia adelante para calcular la salida de la red en cada paso, existe la propagación hacia atrás para calcular el gradiente de la función de pérdida. Esto es muy costoso computacionalmente, y el

calculo no se puede paralelizar, ya que la propagación es inherentemente secuencial, por lo que se debe calcular desde el paso $t = 1$, hasta el paso $t = T$. Otro problema para secuencias muy largas es que el gradiente diverja o desaparezca, dado que al hacer propagación hacia atrás se multiplica T veces por la matriz $\boldsymbol{W}$, entonces si el valor propio más grande de esta matriz $\lambda_{max}(\boldsymbol{W}) > 1$ el gradiente se va a valores gigantescos, mientras que si es al revés $\lambda_{max}(\boldsymbol{W}) < 1$ el gradiente tenderá a desaparecer. Una solución a esto es cambiar la arquitectura para cada elemento de la red.

Long Short-Term Memory (LSTM)

Las LSTM [15] no tienen una arquitectura diferente de los RNN, pero usan una función diferente para calcular el estado oculto $\boldsymbol{h}^{(t)}$. La elementos en las LSTM se llaman celdas y se pueden pensar como cajas negras que toman como entrada el estado anterior $\boldsymbol{h}^{(t-1)}$ y la entrada actual $\boldsymbol{x}^{(t)}$. Internamente, estas celdas deciden qué guardar (y qué borrar) de la memoria de manera de evitar problemas con el gradiente al hacer propagación hacia atrás. Resulta que este tipo de unidades son muy eficientes para capturar dependencias a largo plazo, y es el tipo de red recurrente que se utiliza en este trabajo para el método con GAN.

2.4.4. Optimización de parámetros

El entrenamiento de las redes neuronales consiste en hallar los parámetros óptimos que minimizan una cierta función de pérdida. Esto se hace de manera iterativa, y para esto existen algoritmos que permiten acercarse a estos valores en una dirección óptima.

Suponiendo que se está trabajando en un modelo predictivo, y que se cuenta con muestras como pares $z = (\boldsymbol{x}, y)$ donde $\boldsymbol{x}$ es la entrada e y la salida correspondiente. Y por otro lado se considera una función de pérdida $l(\hat{y}, y)$ que mide el costo de predecir $\hat{y}$ cuando la salida es y, y una familia $\mathcal{F}$ de funciones $f_{\boldsymbol{\omega}}$ parametrizadas por un vector de pesos $\boldsymbol{\omega}$. El objetivo es encontrar la función $f^*_{\boldsymbol{\omega}} \in \mathcal{F}$ tal que minimice la esperanza de la función de pérdida $Q(z, \omega) = E[l(f_{\boldsymbol{\omega}}(\boldsymbol{x}), y)]$, como se muestra en la ecuación 2.9. Para esto es necesario calcular la esperanza en todo el espacio de $\boldsymbol{x}$, lo cual en la práctica es imposible. Una forma de aproximar esta esperanza es mediante el promedio sobre las muestras observadas (2.10), al cual se le llama riesgo empírico.

$$f^*_{\boldsymbol{\omega}} = \arg\min_{\boldsymbol{\omega}} Q(z, \omega) \tag{2.9}$$

$$E_n(f_{\boldsymbol{\omega}}) = \frac{1}{n}\sum_{i=1}^{n} l(f_{\boldsymbol{\omega}}(\boldsymbol{x}_i, y_i)) \tag{2.10}$$

Para minimizar el riego empírico se utiliza el descenso por gradiente (GD), donde para cada iteración actualiza los pesos ω utilizando el gradiente de $E_n(f_{\boldsymbol{\omega}})$, como

se muestra en la siguiente ecuación.

$$\boldsymbol{\omega}_{t+1} \quad = \quad \boldsymbol{\omega}_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\omega}} Q(z_i, \boldsymbol{\omega}_t) \tag{2.11}$$

Al parámetro γ se lo llama paso de aprendizaje y es el que controla el avance del descenso.

Por otro lado el algoritmo de descenso por gradiente estocástico (SGD) [11] es una drástica simplificación de GD. En lugar de calcular el gradiente de $E_n(f_{\boldsymbol{\omega}})$ exactamente, en cada iteración se estima este gradiente sobre un único ejemplo aleatorio z_t:

$$\boldsymbol{\omega}_{t+1} \quad = \quad \boldsymbol{\omega}_t - \gamma_t \nabla_{\boldsymbol{\omega}} Q(z_t, \boldsymbol{\omega}_t) \tag{2.12}$$

Dado que el algoritmo estocástico no necesita recordar qué muestras se visitaron durante las iteraciones anteriores, puede procesar muestras sobre la marcha en un sistema implementado. En tal situación, el descenso de gradiente estocástico optimiza directamente el riesgo esperado, ya que las muestras se extraen aleatoriamente de la distribución real.

2.5. Métricas

En la siguiente sección se definen las métricas que se utilizan más adelante en este trabajo.

2.5.1. Discrepancia Media Máxima (MMD)

Discrepancia Media Máxima (MMD) es un *test* estadístico propuesto en [13], para validar la hipótesis de que dos distribuciones p y q definidas en el mismo espacio son distintas. La manera de hacer esto es encontrando una función suave, que sea grande en las muestras de p, y pequeña en las de q tal como se define en la siguiente ecuación:

$$MMD[\mathcal{F}, p, q] \quad := \quad \sup_{f \in \mathcal{F}} \left(\mathbf{E}_{\boldsymbol{x} \sim p}[f(\boldsymbol{x})] - \mathbf{E}_{\boldsymbol{x}^* \sim q}[f(\boldsymbol{x}^*)] \right) \tag{2.13}$$

donde $\mathcal{F}$ es una clase de funciones $f : \mathcal{X} \to \mathbb{R}$ y $\mathcal{X}$ es el espacio donde están definidas p y q. La definición de la ecuación 2.13 se puede aproximar de forma empírica, sustituyendo las distribuciones p y q por un conjunto de muestras de ambas, X y X^* respectivamente, tal como se muestra en la ecuación 2.14. Bajo ciertas hipótesis sobre el espacio $\mathcal{X}$, y la clase $\mathcal{F}$, que se pueden encontrar en [13], esta forma empírica se puede llevar a una forma más computable como se muestra en la ecuación 2.15, donde K es la función suave (núcleo) previamente seleccionada.

$$MMD[\mathcal{F}, X, X^*] \quad := \quad \sup_{f \in \mathcal{F}} \left(\frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{x}_i) - \frac{1}{m} \sum_{j=1}^{m} f(\boldsymbol{x}_j^*) \right) \tag{2.14}$$

$$MMD[K, X, X^*] = \left[\frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{2}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} K(\boldsymbol{x}_i, \boldsymbol{x}_j^*) + \frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} K(\boldsymbol{x}_i^*, \boldsymbol{x}_j^*) \right]^{1/2} \quad (2.15)$$

En el trabajo realizado por [8] sobre la generación de series temporales a través de GANs, se elige el cuadrado de la ecuación 2.15 ($MMD()^2$) como medida de discrepancia entre las muestras reales y las muestras generadas, tomando la función RBF como núcleo $K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{(2\sigma^2)}\right)$, donde $\boldsymbol{x}$ e $\boldsymbol{y}$ son vectores en el caso de las series univariables y matrices en el caso multivariable. Siguiendo a [8], en este trabajo de tesis se usa la misma medida para medir el "éxito" del modelo GAN como generador de series sintéticas. Durante el entrenamiento se calcula el promedio del valor de MMD entre los *batches*[1] de muestras reales y generadas para cada época, lo que permite visualizar la evolución del modelo como generador de una mejor manera que las funciones de pérdida de ambas redes. A modo de simplificar las cuentas, en este trabajo el ancho σ del núcleo se calcula como la desviación estándar de las muestras reales utilizadas en el entrenamiento.

2.5.2. Métricas para evaluar la clasificación

En este trabajo de detección de anomalías se asume que se tiene dos clases de datos: *negativos*, que representa la clase predominante asociada a los datos considerados normales, y *positivos*, que representa la clase minoritaria asociada a los datos anómalos. Se define $C = \{c_+, c_-\}$ como el conjunto de las clases posibles, donde se denomina TP (*true positive* por sus siglas en inglés) el número de datos $x_i \in c_+$ correctamente clasificados, TN (*true negative*) el número de $x_i \in c_-$ correctamente clasificados, FP (*false positive*), y FN (*false negative*), el número de $x_i \in c_-$, y $x_i \in c_+$ mal clasificados respectivamente. A partir de estos contadores, TP, TN, FP, FN, se pueden calcular diferentes indicadores para evaluar el desempeño de la clasificación. *Accuracy*: es la relación entre el total de datos bien clasificados y el total en general. *Recall*: restringiéndose sólo a la clase *positivos* es la relación entre los datos bien clasificados y el total. *Precision*: restringiéndose sólo a los clasificados como *positivos* es la relación entre los bien clasificados y el total. *F1*: es la media armónica entre *Recall* y *Precision*.

- *Accuracy*: $ACC = \frac{TP+TN}{TP+TN+FP+FN}$
- *Recall*: $REC = \frac{TP}{TP+FN}$
- *Precision*: $PRE = \frac{TP}{TP+FP}$
- *F1* $= \frac{REC\ PRE}{PRE+REC}$

[1] Pequeños lotes.

Capítulo 3

Enfoque: Métodos para la detección de anomalías

El enfoque de esta tesis es estudiar la detección de anomalías en series temporales multivariables (STM), utilizando modelos generativos de aprendizaje profundo para la implementación de métodos semi-supervisados. La principal motivación de utilizar estos modelos, es poder desarrollar métodos capaces de monitorear las diferentes variables de un sistema a la misma vez, sin la necesidad de extraer características, ni de ajustar hiperparámetros específicos para cada variable de las series.

Por otro lado, en un sistema real las anomalías en general son eventos poco frecuentes, la mayor parte del tiempo el sistema se comporta de forma normal. Por lo que la detección de anomalías generalmente es un problema desbalanceado. El volumen de datos que generalmente se maneja en un sistema de monitoreo, presenta otro problema que es la falta de etiquetas para entrenar un modelo supervisado. Por lo tanto, una buena manera de hacer frente al desbalance y a la falta de etiquetas es implementar un método semi-supervisado, que consista en entrenar un modelo con datos que se saben que provienen de un comportamiento normal. Con esto se busca construir una línea de base de este comportamiento, y así simplemente hacer las detecciones comparando los datos nuevos con esta línea de base.

Dos de las herramientas de generación del aprendizaje profundo más relevantes de los últimos años: *Variational Auto-Encoders* (VAE) y *Generative Adversarial Networks* (GAN); han demostrado a lo largo de una una gran cantidad de trabajos [1, 20, 22, 23, 29, 30] el poder para aprender la distribución de un conjunto de datos y ser capaces de generar muestras sintéticas que parecen reales. Es por esto que se propone crear la línea de base del comportamiento normal de las STM, a través de estas herramientas.

3.1. Trabajos relacionados

Como se dijo al principio la detección de anomalías es un problema investigado en diferentes áreas y dominios, el trabajo realizado en [4] es uno de los más citados en el campo de la detección de anomalías, allí se resumen algunas de estas áreas y dominios, como también los diferentes tipos de anomalías y técnicas de detección, ubicando a estas últimas en cuatro grandes categorías: clasificación, vecino más cercano, *clustering*, y técnicas estadísticas. Desde la popularización del aprendizaje profundo y la capacidad de manejar grandes cantidades de datos otro abanico de técnicas y métodos han aparecido. En [3] se resumen diferentes métodos para la detección de anomalías al igual que [4] pero con un enfoque hacia técnicas de aprendizaje profundo: redes adversarias (GAN), recurrentes (RNN) y convolucionales (CNN), *Auto-encoders* (VAE), y aprendizaje por refuerzo (RL).

En el dominio de las series temporales se pueden encontrar estas técnicas en diferentes trabajos. Por la cantidad de datos y la falta de etiquetas de las anomalías, generalmente estas técnicas son técnicas semi-supervisadas, donde se asume que todo los datos de entrenamiento son datos sin anomalías, de manera de ajustar un modelo sobre estos. Luego en la fase de aplicación todo dato que no se ajuste a este modelo se toma como anomalía.

Previo a esta tesis se trabajó sobre métodos para la detección de anomalías en series univariable. Parte de este trabajo se puede ver en [24], donde se presentan los dos métodos utilizados. El primero mediante modelos auto-regresivos SARIMA aprende la estadística de las series, y luego mediante filtros de Kalman es capaz de hacer predicciones sobre los cuales compara los nuevos datos. El segundo método usa ventanas de Parzen para ajustar la distribución de las series en pequeños intervalos de tiempo, y así comparar los nuevos datos con la distribución aprendida. La desventaja de estos tipos de métodos es que cuenta con hiperparámetros que inciden directamente en el modelo, que para ser ajustados se requieren un estudio previo del comportamiento de cada serie, como la varianza, la estacionaridad, entre otros, haciéndolos muy costosos en cuanto al trabajo previo que requieren para analizar un sistemas con muchas variables diferentes entre sí.

Por otro lado las técnicas de aprendizaje profundo no tienen este tipo de problemas, siendo capaces de extraer diferentes características de los datos y aprender un modelo de los mismos de forma automática como se explica en [11]. Las CNN son una de estas técnicas, donde por ejemplo en [26] a partir de un pequeña secuencia histórica es capaz de predecir los valores siguientes y compararlos con los reales para determinar si estos son anómalos o no, además de obtener buenos resultados para series estacionarias, también funcionan muy bien con series no-estacionarias. El problema es que si se quisiera implementar en un sistema con muchas variables como los que aparecen en [21, 25, 28], se debería entrenar un modelo para cada variable, lo que puede ser arduo para el usuario al momento de tener que poner en marcha el entrenamiento de cada modelo.

Por otro lado existen trabajos pensados para problemas multivariables como [2] donde se implementa una técnica de reducción de la dimensión y posterior reconstrucción utilizando PCA, para una gran cantidad de variables como en [21] (más

de 100) pero sin tener en cuenta la marca de tiempo, es decir, tomando todas las variables de cada instante como vectores de características independientes entre sí. Llevar el problema a un domino espacial descartando la relación del entorno de los datos, hace que se pierda un información importante, y puede llevar a que el método no sea capaz de detectar anomalías de contexto. Trabajos similares son presentados en [1,30] donde se cambia la herramienta PCA por un modelo VAEs, haciendo la detección a partir del error de reconstrucción y la probabilidad de reconstrucción respectivamente. Una forma de trabajar con las VAEs teniendo en cuenta el entorno temporal, es creando muestras a partir de pequeñas secuencias de datos multivariables mediante ventanas deslizantes en el tiempo. Es en [29] que se propone esta técnica sencilla pero efectiva, y la cual dio lugar a uno de los enfoque de esta tesis que se desarrolla más adelante.

En el aprendizaje profundo las herramientas que están pensadas específicamente para trabajar con secuencias de datos son las redes neuronales recurrentes. Los métodos LSTM-AD y LSTM Encoder-Decoder presentados en [23] y [22] respectivamente, son dos métodos de detección de anomalías en series multivariables, ambos basados en las populares redes recurrentes LSTM. Al igual que los métodos que se vieron al principio de esta sección, LSTM-AD predice el valor para el instante de tiempo siguiente y los compara, mientras que LSTM *Encoder-Decoder* bajo el enfoque de *Auto-Encoders* entrena dos redes recurrentes LSTM de manera de reconstruir tramos de series temporales y hacer la detección a partir del error de reconstrucción. Ambos métodos son capaces de determinar en qué variable se dio la anomalía y en qué instante de tiempo, lo que permite una detección más específica en sistemas con múltiples variables. Como se puede ver en el último método, no solo existen diferentes variantes dentro del aprendizaje profundo sino que estas se pueden combinar, lo que amplía aún más el espectro de métodos para atacar esta problemática. Otro ejemplo donde se combinan dos herramientas del aprendizaje profundo es el método RGAN, presentado en [8] como un método de generación de series médicas sintéticas, y posteriormente utilizado como detector de anomalías en [20]. En este último ambas redes son utilizadas para detectar anomalías, por un lado el generador es capaz de generar secuencias sintéticas lo que permite calcular el residuo entre una secuencia real y la generada que más se le parezca. La diferencia con [22] es que en este caso el Generador devuelve muestras sintéticas a partir de ruido por lo que es necesario hacer una búsqueda de la muestra sintética correcta. Por el otro lado el Discriminador es un detector en sí ya que devuelve la probabilidad de que el dato sea real o no, y al ser entrenado solamente con datos normales ante una anomalía este debería devolver una probabilidad baja. Según [20] para la aplicación de este método es necesario reducir la dimensión de las series aplicando PCA. En el caso de que se presente un anomalía en solo una o algunas variables, la reducción de la dimensión impediría identificar qué variables fueron afectadas. En este trabajo de tesis se tomó esta idea de manera de analizar el rendimiento de este método sin reducir la dimensión de los datos y se compararon con los resultados obtenidos en [20].

3.2. Métodos implementados

Tal como se anticipa en la sección anterior, en este trabajo se implementaron dos métodos de detección de anomalías en series multivariables. Ambos están basados en trabajos previos, con algunas modificaciones con respecto a los métodos originales, que se describen en las próximas secciones.

Preprocesamiento

Previo al entrenamiento de los métodos, es necesario realizar un preprocesamiento a los datos. Generalmente los conjuntos de datos que pueden ser convertidos a series temporales se encuentran en formato de tabla donde las columnas son las variables del sistema y las filas los valores para cada instante de tiempo. Una de las columnas está asociada a la marca de tiempo (*timestamp*), muchas veces esta marca puede corresponder al momento en que los valores fueron emitidos, o al momento que fueron agregados a la tabla. Entre estas marcas muchas veces no hay un paso fijo, y pueden estar desordenados temporalmente, lo que puede ser un problema si se quiere tener en cuenta la correlación temporal. Por lo tanto, lo primero es aplicar una función de agrupamiento que puede ser: suma, promedio, mediana, máximo, etc., y hacer un nuevo muestreo con paso fijo. Por ejemplo, si se selecciona un paso de 1 minuto, a todos los valores entre un minuto y otro se le aplicara la función de agregado y se le asignará el resultado al minuto superior, así para todas las variables, quedando el conjunto con una cadencia de 1 minuto, y de forma ordenada.

Muestras

Para poder trabajar con ambos métodos y que estos puedan capturar información de la correlación temporal, se debe trabajar con secuencias de datos de largo T como muestras. Es decir, suponiendo que la STM con la que se está trabajando contiene N variables, los datos para cada instante se representan como un vector columna $\boldsymbol{x}_t \in \mathbb{R}^N$, de modo que para crear una muestra para estos métodos se deben agrupar T datos consecutivos, quedando una secuencia $\{\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, ..., \boldsymbol{x}_{t+T}\}$[1]. Las siguientes muestras se obtienen por desplazamiento $W \leq T$ hacia adelante: $\{\boldsymbol{x}_{t+W}, \boldsymbol{x}_{t+1+W}, ..., \boldsymbol{x}_{t+T+W}\}$. Así sucesivamente se va construyendo el espacio muestral para este método. En el caso de que se cumpla que $W < T$ habrá solapamiento entre las muestras consecutivas, y en el caso $W = T$ no compartirán ningún dato entre sí. Para trabajar de forma más sencilla se toman estos conjuntos como matrices de N filas y T columnas, quedando el espacio muestral de la siguiente forma:

[1] Los sub-indices t y T, no son valores temporales. Una vez que los datos están ordenados y tiene una cadencia fija, se puede hacer referencia a la posición de estos solo con índices enteros, por lo tanto t y T son valores enteros.

$$\begin{aligned} \boldsymbol{X}_r &= [\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, ..., \boldsymbol{x}_{t+T}] \\ \boldsymbol{X}_{r+1} &= [\boldsymbol{x}_{t+W}, \boldsymbol{x}_{t+1+W}, ..., \boldsymbol{x}_{t+T+W}] \\ \boldsymbol{X}_{r+2} &= [\boldsymbol{x}_{t+2W}, \boldsymbol{x}_{t+1+2W}, ..., \boldsymbol{x}_{t+T+2W}] \\ \vdots &= \qquad\qquad \vdots \end{aligned}$$

Para el método con GAN, las dimensiones del espacio de latencia también están determinadas por T. Cada elemento de la RNN de G tiene como entrada un vector $\boldsymbol{z}_t \in \mathbb{R}^M$ $(M < N)$ que está generado a partir de una distribución normal isotrópica[2] $\mathcal{N}(\mathbf{0}, \mathbb{I})$ donde la media es un vector de ceros de largo M, y la matriz de covarianza es la matriz identidad de dimensiones $M \times M$. Al igual que los datos, estas variables aleatorias se agrupan en secuencias de largo T: $\{\boldsymbol{z}_t, \boldsymbol{z}_{t+1}, ..., \boldsymbol{z}_{t+T}\}$. En este caso no se hace un corrimiento de la ventana T, sino que para cada muestra se generan T variables aleatorias cada vez. De esta manera el espacio de latencia $\mathcal{Z}$ está determinado por matrices $\boldsymbol{Z} = [\boldsymbol{z}_t, \boldsymbol{z}_{t+1}, ..., \boldsymbol{z}_{t+T}] \in \mathcal{Z} \subset \mathbb{R}^{M \times T}$. En el caso del método con VAE en el espacio de latencia simplemente se trabaja con vectores de largo M.

3.2.1. Método con VAE

El siguiente método está basado en el trabajo realizado en [29]. Este método utiliza la herramienta VAE para crear la línea de base del comportamiento normal de las series, utilizando pequeñas secuencias de datos.

Arquitectura

Tal como se vio en 2.4.1, las VAE son una variante en la familia de los AE, por lo cual en el proceso de aprender la distribución de la entrada, reduce las muestras al espacio de latencia Z, para luego hacer la reconstrucción. Las muestra en este espacio son vectores de dimensión M, por lo tanto a la entrada de la capa previa al espacio de latencia se requiere que sea un vector de dimensión mayor y no una matriz como las muestras. Cuando se utiliza VAE en problemas con matrices, como por ejemplo imágenes, generalmente se utilizan capas convolucionales para extraer características de estas matrices a la vez que se va reduciendo la dimensión hasta obtener un vector. Para simplificar la arquitectura y el modelo en este método, simplemente se transformó las muestras poniendo todas las filas en una. Por ejemplo, si la STM cuenta con tres variables $(N = 3)$ y se selecciona un largo de secuencia igual a tres $(T = 3)$ también, una muestra tendría la forma que se muestra en 3.1, y luego de alinear sus filas esta muestra quedaría un vector de

[2]Una distribución isotrópica es aquella que se distribuye de igual forma en todas las direcciones.

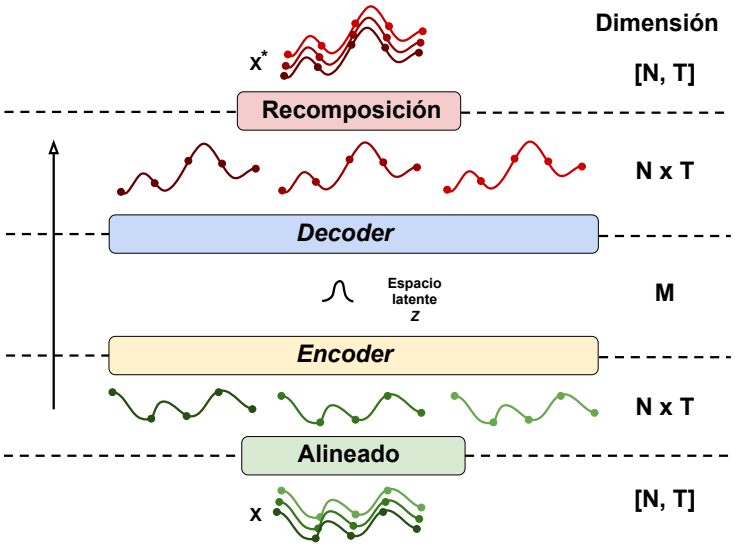


Figura 3.1: Esquema del método con VAE. En éste se muestra el proceso que sufren las muestras para que el modelo de VAE trabaje con vectores, junto con los cambios de dimensiones en las diferentes capas del modelo.

dimensión nueve ($NT = 9$) como se muestra en 3.2.

$$\boldsymbol{x}_t = \begin{bmatrix} x_t^{(1)} & x_{t+1}^{(1)} & x_{t+2}^{(1)} \\ x_t^{(2)} & x_{t+1}^{(2)} & x_{t+2}^{(2)} \\ x_t^{(3)} & x_{t+1}^{(3)} & x_{t+2}^{(3)} \end{bmatrix} \tag{3.1}$$

$$\boldsymbol{x} = [x_t^{(1)}, x_{t+1}^{(1)}, x_{t+2}^{(1)}, x_t^{(2)}, x_{t+1}^{(2)}, x_{t+2}^{(2)}, x_t^{(3)}, x_{t+1}^{(3)}, x_{t+2}^{(3)}] \tag{3.2}$$

A este proceso generalmente se lo llama en inglés aplanar (*flatten*), en esta tesis se hace referencia a esta acción como Alíneado, y se lo considera por fuera de la red de la VAE, ya que no cuenta con parámetros que deban ser optimizados. Luego la VAE también devuelve las reconstrucciones de las muestras como vectores de largo $N \cdot T$, por lo que se hace un proceso de Recomposición donde se lleva estas reconstrucciones a la dimensión original de las muestras. En resumen, para el modelo las muestras de entrada y por lo tanto las reconstrucciones son vectores de dimensión $N \cdot T$, y el espacio de latencia un vector de dimensión M. Este proceso se puede apreciar en la figura 3.1.

Luego llevadas las muestras a un vector, estas están en condiciones para la entrada del *Encoder*.

Encoder

El *Encoder* cuenta con tres capas: una oculta, y dos de salida. Todas estas son capas totalmente conectadas (*fully connected*), y solo la capa oculta tiene una función de activación que puede ser *relu* o *tanh*, y la cantidad de unidades de esta

capa se fija previamente. Las capas de salida se encuentran a la misma profundidad en la red y no tienen función de activación. Sus salidas corresponden a la media μ_z y a la varianza σ_z de la posteriori $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, donde la dimensión de salida de estas es M.

Decoder

En el caso del *Decoder* sus entradas provienen del espacio de latencia, por lo que las muestras se obtiene a partir de la salida del *Decoder* y el "truco" de reparametrización: $z^* = \mu_z + \sigma_z\epsilon$, donde ϵ proviene de una normal de media cero y varianza uno. Luego este tiene solo una capa oculta al igual que el *Encoder* con una cantidad de unidades a determinar. Por último, la capa de salida es totalmente conectada y como función de activación se usa la función *tanh* dado que sus valores se encuentran entre $[-1, 1]$ al igual que los datos de entrenamiento. La salida del *Decoder* $\boldsymbol{x}^*$ pertenece al mismo espacio que la entrada $\boldsymbol{x}$, donde por último se la lleva al formato de las muestras originales tal como se muestra en la figura 3.1.

Una vez entrenadas las redes, se espera que el modelo solo devuelva reconstrucciones que se parezcan a las muestras de entrenamiento. Es decir, si a la entrada se pone un muestra anómala, la reconstrucción debería ser la muestra normal que más se le parezca.

La detección en este caso se hace a partir del error de reconstrucción. La forma en que se hace esto es al igual que en [30], con un instancia de validación para la cual se reserva un porcentaje de la cantidad de datos destinados al entrenamiento. La idea es determinar la media μ_e y la desviación estándar σ_e del error de reconstrucción para cada variable en cada instante.

Luego con estos valores una vez puesto el método a detectar, para una muestra x se utiliza el modelo para obtener la muestra de reconstrucción x^*, y se calcula el error cuadrático de reconstrucción $\mathbf{e}_t = (\boldsymbol{x}_t - \boldsymbol{x}_t^*)^2$. Luego se compara a este con la media y la varianza del error calculados en el etapa de validación, donde si se cumple que $\mathbf{e}_t \leq \mu_e + K\sigma_e$ se considera como normal, y en caso contrario se considera anomalía. Para el caso multivariable esto se hace para cada variable de las secuencias.

3.2.2. Método con GAN

Basado en los trabajos de [8] y [20] se implementó un método que combina GANs con RNNs, de manera de crear una línea de base del comportamiento normal de las series, teniendo en cuenta la correlación temporal.

Como se muestra en la sección 2.4.2, las GANs ponen dos redes neuronales a competir entre sí, de manera que entre ambas se potencien en alcanzar sus objetivos. Estas redes se las nombra por su función: Generador (G), y Discriminador (D). La primera a partir de ruido aleatorio (espacio de latencia) genera muestras falsas intentando imitar las reales que provienen del conjunto de entrenamiento de manera de engañar a D. Mientras que D intenta diferenciar entre las muestras reales y las falsas, devolviendo una probabilidad cercana a 1 cuando está seguro de

que la muestra es real y 0 en caso contrario. Una vez finalizado el entrenamiento, G es capaz de generar muestras muy similares a la naturaleza de las que provienen del conjunto de entrenamiento, a la vez que D puede determinar con precisión que muestras son reales o no.

Si se entrena la GAN para crear la línea de base, G debería ser capaz de crear una infinidad de muestras que se comporten de forma normal, y por otro lado D sólo conocerá muestras reales que provienen de datos normales, por lo que debería asignarle una probabilidad baja a toda muestra que se aparte del comportamiento normal.

Fase de entrenamiento

El modelo sigue la arquitectura de una GAN regular, donde las redes del generador (G) y el discriminador (D) han sido sustituidos por redes neuronales recurrentes (RNN). El esquema se puede ver en la figura 3.2. La arquitectura elegida para ambas redes RNN es *Long Short-Term Memory* (LSTM), y la cantidad de celdas usada para ambas redes es la misma y está determinada por el largo de secuencia T. Todas las celdas tienen un solo estado oculto y una capa densa que determina la salida de cada una. Las entradas y salidas son diferentes para las dos redes. Las entradas de las celdas de G son las muestras aleatorias $\boldsymbol{z}$, y las salidas son vectores de la misma dimensión que los datos, $\boldsymbol{x}_n$. De esta manera la salida de G devuelve matrices X^*, que corresponden a las muestra sintéticas (falsas) de la misma dimensión que las muestras reales X_r del conjunto de entrenamiento. Las entradas de las celdas de D son de dimensión n para coincidir con las muestras, y las salidas son de dimensión 1, las cuales devuelven la predicción (un valor entre $[0, 1]$) para cada instante de tiempo.

Ciclo

El ciclo de entrenamiento se hace para una cantidad de épocas determinada donde en cada una el modelo ve todos los datos de entrenamiento de manera secuencial. Para entrenar tanto D como G se busca minimizar la función de entropía cruzada 3.3 entre las predicciones de $D(\boldsymbol{x}) = y_p$ y los vectores de etiquetas y: $\mathbf{1}$, $\mathbf{0} \in \mathbb{R}^T$, que indican si la secuencia es real o falsa respectivamente. En el caso de D, este recibe dos *batches*, uno de muestras reales proveniente del conjunto de entrenamiento, y otro de muestras falsas provenientes del Generador. Para cada *batch* se busca que la predicción disimile lo menos posible con su correspondiente etiqueta, por lo tanto la función de pérdida a minimizar es la que se muestra en 3.4.

$$CE(\mathbf{y}_p, \mathbf{y}) = -\frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_i \log(\mathbf{y}_{pi}) \tag{3.3}$$

$$loss_D = CE(D(\boldsymbol{x}), \mathbf{1}) + CE(D(\boldsymbol{x}^*), \mathbf{0}) \tag{3.4}$$

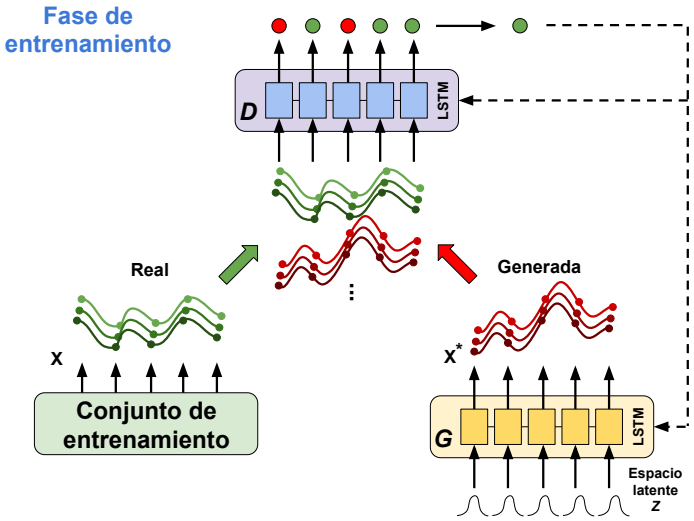


Figura 3.2: Esquema de la fase de entrenamiento. El discriminador D recibe muestras tanto del conjunto de entrenamiento, como las generadas por G. Luego las predicciones que este devuelve son utilizadas tanto por él mismo como por G para ajustar sus parámetros.

Para optimizar los parámetros de D se busca minimizar la función $loss_D$ en función de los mismos, utilizando el método de descenso por gradiente estocástico (SGD). El gradiente se aproxima utilizando propagación hacia atrás (*Backpropagation*).

En el caso de G dado que este pretende engañar a D, se busca minimizar la disimilitud entre las predicciones de las muestras que genera, con el vector de etiquetas de las reales, esto es:

$$\begin{aligned} loss_G &= CE(D(\boldsymbol{x}^*), \mathbf{1}) \\ &= CE(D(G(\boldsymbol{z})), \mathbf{1}) \end{aligned} \tag{3.5}$$

Para encontrar los parámetros óptimos de G se busca minimizar la función $loss_G$, utilizando el método de optimización ADAM [17] el cual es una variación de SGD. El gradiente se calcula de la misma forma para D pero sólo usando *batches* de muestras falsas.

Así, de a pequeños *batches* de muestras reales que son consecutivas y de muestras falsas generadas a partir de ruido aleatorio, se recorre todo el conjunto de entrenamiento para cada época. La cantidad de épocas a utilizar se puede determinar con la evolución de las funciones de pérdida. Cuando el valor de estas varía poco entre épocas consecutivas, se puede decir que las redes convergieron a una solución.

Fase de aplicación

Los datos futuros que se quieran analizar deben ser pre-procesados al igual que los datos para el entrenamiento. En el caso que se pretenda analizar los datos "en

línea"(con el sistema monitoreado funcionando), es necesario esperar hasta haber completado una muestra, esto depende del paso fijo elegido para hacer el agregado de los datos, y del desplazamiento W.

Clasificación: Discriminación y Residuo.

Una vez entrenadas las redes, ambas se utilizan para determinar si una muestra contiene anomalías o no. En el caso de D, éste es un detector por naturaleza ya que devuelve una probabilidad cercana a 0 para todo instante de tiempo que se aparte de la línea de base aprendida.

$$D(\boldsymbol{x}) \quad = \quad [y_{p1}, y_{p2}, ..., y_{pT}] \tag{3.6}$$

Tal como se muestra en 3.6, donde $y_{pi} \in [0, 1]$ con $i = 1, ..., T$, el Discriminador devuelve un vector de probabilidades para cada instante de tiempo el cual es un valor de discriminación. Utilizando un mismo valor u_D como umbral para todos los instantes de tiempo se puede determinar para cada uno si es anomalía o no. Este umbral se calcula a partir de un etapa de validación, posterior al entrenamiento, donde para un pequeño conjunto de muestras se calcula la probabilidad mínima para cada instante t. Luego se halla la media y la desviación estándar de estos valores mínimos a lo largo de la secuencia. Suponiendo que los valores calculados son: μ_{minT} y σ_{minT}, para la media y la desviación estándar, entonces el umbral de detección se fija como se muestra en 3.7.

$$u_D \quad = \quad \mu_{minT} - \sigma_{minT} \tag{3.7}$$

En el caso de G, éste permite generar infinidad de muestras sintéticas similares a las reales sólo a partir de ruido aleatorio. Entonces dada una muestra real a analizar, se debe encontrar cuál de las muestras sintéticas se parece más a ésta, y calcular la diferencia entre ambas para obtener un valor de residuo. La cantidad de muestras a generar se debe preestablecer y ésta puede estar vinculada al problema. Para determinar la más parecida, se calcula la correlación entre las variables de la muestra real y las generadas. Luego promediando estos valores para cada muestra, se selecciona aquella para la cual se obtuvo el promedio más grande. Luego el umbral de error u_G se calcula de la misma forma que el método con VAE.

En la figura 3.3 se muestra el esquema de esta fase de aplicación. En éste se muestra la combinación de la discriminación y el residuo como una suma, la cual corresponde a una suma binaria (OR), por lo que alcanza con que uno de los valores indique que es una anomalía para determinar definitivamente un instante como anómalo. Pero también se puede sustituir a ésta por una multiplicación (AND), y así se determinará anomalía sólo si ambos indicadores lo determinan. Los resultados de estas combinaciones como ambos por separado se muestran en el capítulo 5.

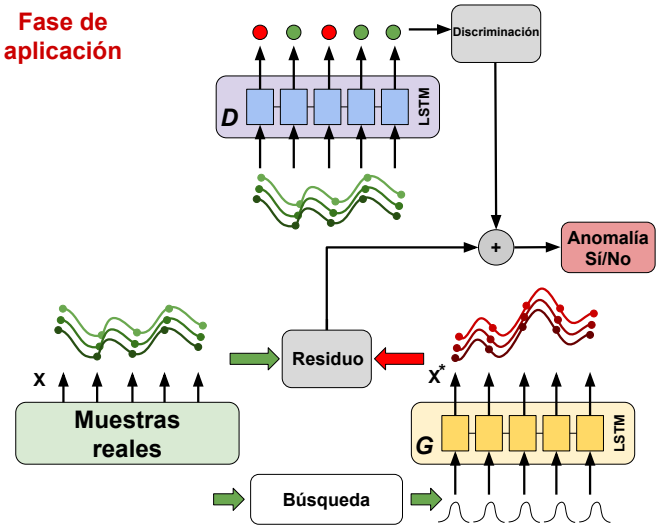


Figura 3.3: Esquema de la fase de aplicación. Una vez entrenados D y G ambos pueden ser utilizados para la detección. Por un lado con G se busca la muestra que más se asemeja a la muestra real que se está analizando y se calcula el residuo entre ambas para saber cuan cerca se está de la distribución aprendida. Por otro lado de D se espera que devuelva probabilidades altas cuando los valores se ajusten a lo que él considera como real, y probabilidades bajas para valores que muestren patrones no vistos en el entrenamiento.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

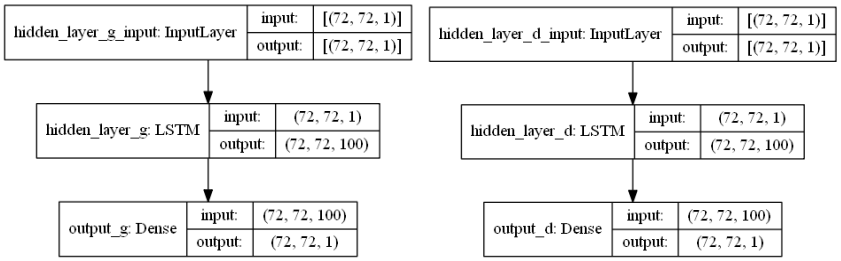
Arquitectura, Implementación y Conjuntos de datos

En el siguiente capítulo se muestran algunos detalles de la implementación de los métodos, y se especifican qué parámetros el usuario debe manipular previo al entrenamiento. Ambos métodos fueron implementados con la librería *keras*[1]. Esta esta pensada para simplificar la comprensión y manipulación de herramientas de aprendizaje profundo, minimizando la cantidad de acciones necesarias para casos comunes y proporciona mensajes de error claros y procesados. También cuenta con una extensa documentación.

4.1. Arquitectura para el método con GAN

Este método emplea el uso de dos redes recurrentes LSTM, con la misma profundidad determinada por el largo de secuencia T. También la profundidad para cada elemento de la secuencia se seleccionó igual para ambos, con una capa entrada, un capa oculta, y una capa de salida. Esto último se puede ver en el esquema de la red que se muestra en la figura 4.1 a modo de ejemplo, con las dimensiones para uno de los casos de prueba del capítulo siguiente. La cantidad de unidades de las capas de entrada y salida quedan determinadas por la dimensión de las muestras, tanto en el espacio real como en el espacio de latencia. Lo que el usuario sí puede seleccionar, es la cantidad de unidades de la capa oculta. Estos parámetros a nivel de código se los nombró como *hidden_units_g* y *hidden_units_d*, para el Generador y Discriminador respectivamente. Por otro lado, las funciones de activación en la capa de salida se eligieron diferentes. Para el Generador, dado que éste a la salida debe devolver muestras que se parezcan a las reales, se utiliza tanh() dado que esta varía entre $[-1, 1]$ al igual que las muestras una vez normalizadas. Para el Discriminador se seleccionó la función *sigmoid*() dado que para cada elemento se espera una probabilidad que indique si el instante es anómalo o no. Luego de seleccionados dichos parámetros el modelo de ambas redes queda determinado.

[1] `https://keras.io/`



(a) Generador. (b) Discriminador.

Figura 4.1: Arquitectura del modelo GAN, donde se indica: el tipo, la entrada y la salida para cada capa del Generador (a) y el Discriminador (b). Ambos tiene la misma estructura, lo único que cambia son las dimensiones de las entradas y salidas.

Para las funciones de perdida se usa la misma para ambos métodos *Binary-Crossentropy()*, la cual calcula el valor de discrepancia entre las etiquetas y las salidas del Discriminador. Para un *batch* de muestras reales y otro de generadas se minimiza la salida del Discriminador del primer *batch* con etiquetas de valor 1, junto con la salida del Discriminador para el segundo *batch* con etiquetas de valor 0. Luego se suman y ese valor se usa para optimizar los parámetros del Discriminador. Por otro lado el Generador optimiza sus parámetros con el valor obtenido de comparar la salida del Discriminador para muestras generadas, con etiquetas de valor 1 de manera de engañar al Discriminador.

Para encontrar los parámetros que minimizan las funciones de pérdida se utilizaron los optimizadores SGD para el discriminador y ADAM [17] para el Generador, siguiendo los consejos de [10]. Estos al menos requieren de que el usuario inicialice el paso de aprendizaje (*learning_rate*).

Stateful

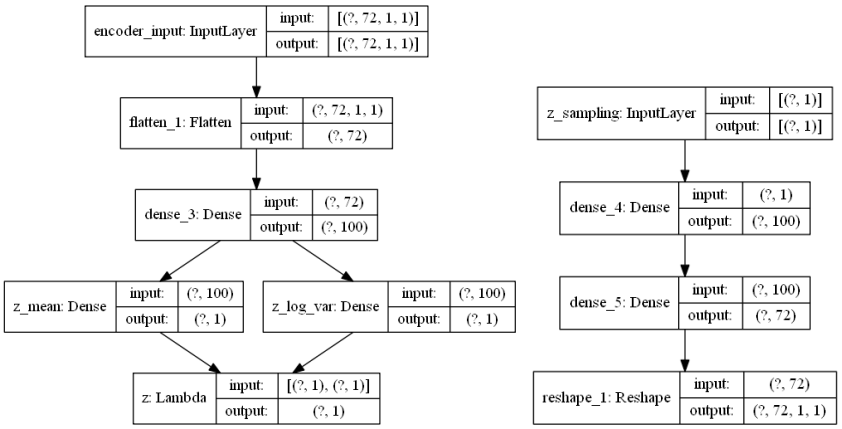
Entre varios parámetros que se pueden fijar para las redes LSTM, existe uno en particular llamado *stateful* que es importante. Activando este parámetro, se le indica al modelo que para cada muestra de cada *batch* mantenga el último estado h_T correspondiente a la última celda de las redes LSTM, para inicializar el estado inicial h_0 correspondiente a la primer celda de la muestra que ocupa el mismo lugar en el *batch* siguiente. Esto permite aprovechar la estacionaridad de los datos, cuando estos son estacionarios.

La implementación de este método está inspirada en los códigos publicados por los trabajos [8][2] y [20][3] los cuales fueron los que presentaron el método. De igual manera para comprender cómo trabajan las redes se implementó un código propio[4], el cual es más simplificado en cuanto a la cantidad de lineas ya que se

[2] `https://github.com/ratschlab/RGAN`

[3] `https://github.com/LiDan456/GAN-AD`

[4] `https://github.com/GastonGarciaGonzalez/GAN_and_VAE_for_anomaly_detection_in_multivariate_time-series`



(a) *Encoder*. (b) *Decoder*.

Figura 4.2: Arquitectura del modelo VAE, donde se indica: el tipo, la entrada y la salida de cada capa del *Encoder* (a) y el *Decoder* (b). También se puede apreciar las conexiones entre capas, y la perfundida a la que se encuentra cada una.

implementó utilizando la librería *keras*[5].

4.2. Arquitectura para el método con VAE

Para construir el modelo de este método se pueden definir el *Encoder*, y el *Decoder* por separado. El esquema de ambos se puede ver en la figura 4.2, con las dimensiones de un ejemplo de las pruebas que se muestran en el capítulo siguiente. Empezando por el *Encoder* este cuenta con cuatro capas distintas si no se tiene en cuenta el Alineado. La primera es una capa de entrada, luego tiene un capa oculta previo al espacio de latencia, y despúes dos capas a la misma profundidad, las cuales corresponden a la media y la varianza de la *posterior*. La última es la capa encargada del "truco de reparametrización"[18], ésta a partir de la salida de las dos capas anteriores, devuelve una muestra $\boldsymbol{z}$, y guarda el valor en memoria para cuando se requiera para hallar el gradiente.

Por otro lado el *Decoder* cuenta con tres capas si no se cuenta la de recomposición. El orden de estas capas es: entrada, oculta, y salida.

Las función de pérdida tiene dos términos como se vio en la sección 2.4.1. El término de reconstrucción en la practica se simplifica por el error cuadrático mínimo entre la muestra y la reconstrucción. Mientras que el segundo término relacionado a la sensibilidad del modelo, se calcula explícitamente con las salidas del *Encoder*.

Para optimizar los parámetros se utiliza el optimizador ADAM al igual que el método anterior.

[5]Ejemplo para entrenar una GAN en *keras*: `https://colab.research.google.com/github/khipu-ai/practicals-2019/blob/master/3b_generative_models.ipynb`

4.3. Pre-procesamiento

Previo al entrenamiento es necesario determinar el tamaño de las muestras. Muchas veces los conjuntos no tienen un paso temporal fijo, por lo que primero es recomendable aplicar una función de agregado, mediante una ventana fija que determine el espacio entre los nuevos datos como puede ser segundos, minutos, u horas, esto dependerá del tipo de series que se este analizando y la cantidad de datos con que se cuente. Hecho esto se debe fijar el largo de secuencia que junto con la cantidad de variables de la serie determinarán la entrada para ambos métodos de detección. Por último el corrimiento de la secuencia, que dependiendo de si es menor o igual al largo de ésta, las muestras se solaparán o no. A nivel de código a estos parámetros se les asignó un nombre, en este caso: *aggregate*, *seq_lenght*, y *seq_step*, siguiendo el orden en que se los describió.

Pre-procesammiento de las etiquetas

Hasta ahora no había sido necesario discutir qué sucede con las etiquetas del conjunto que indican si un dato es anómalo o no. En este trabajo se tomarán las etiquetas con valor 0 para los datos normales y con valor 1 para los anómalos. Para acompañar el cambio de los datos cuando son procesados para generar las muestras, es necesario decidir qué hacer con las etiquetas. Si en el agregado se aplica una función de promedio a los datos al igual que los datos y luego un redondeo, el criterio es similar a una votación, el tipo de etiqueta que prevalezca dentro de la ventana de agregado será el que designará la etiqueta al nuevo dato agregado. En este caso el inconveniente que puede surgir es en el caso que dentro del agregado haya datos con valores considerables como para que el nuevo dato pueda ser considerado una anomalía pero sin embargo quede etiquetado como normal, por la cantidad de datos normales era superior. Esto puede afectar al desempeño del método, ya que los resultados pueden ser incongruentes. Otro criterio es si dentro de la ventana de agregado existe al menos un dato etiquetado como anómalo, entonces el nuevo dato quedará etiquetado como tal. Esto se puede hacer utilizando la función máximo en el agregado. El problema con este criterio es que si dentro de la ventana de agregado hay muy pocos datos etiquetados como anómalos, el valor de éstos puede quedar escondido dentro del promedio y el nuevo dato parecerse a uno normal, sin embargo quedará etiquetado como anómalo. Esto también puede afectar al rendimiento del método. A la vista los dos problemas es necesario tomar una decisión. En este caso se tomará el último criterio.

4.4. Conjuntos de datos

4.4.1. SWaT.

Este conjunto de datos presentado en [9], fue creado para respaldar la investigación en el diseño de Sistemas Ciber Físicos (CPS) seguros. El proceso de recopilación de datos se implementó en un banco de pruebas de tratamiento seguro

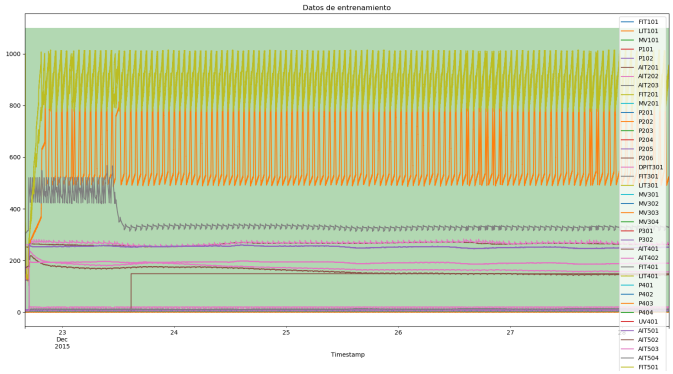


Figura 4.3: Datos correspondientes a los primeros 7 días sin ataques. Las series presentan un transitorio al principio hasta un estado completamente operativo. En éste último se puede ver un comportamiento estacionario y estacional de las series.

de agua (SWaT). Este representa una versión reducida de una planta de tratamiento de agua industrial del mundo real. Esta planta permitió la recolección de datos bajo dos modos de comportamiento: normal y atacado. SWaT se ejecutó sin parar desde su estado en vacío hasta su estado completamente operativo durante un total de 11 días. Durante este período, los primeros 7 días el sistema funcionó normalmente, es decir, sin ataques ni fallas. Durante los días restantes, se lanzaron ciertos ataques cibernéticos y físicos a SWaT mientras continuaba la recopilación de datos. El conjunto de datos que se informa aquí contiene las propiedades físicas relacionadas con la planta y el proceso de tratamiento del agua, así como el tráfico de red en el banco de pruebas. Los datos de las propiedades físicas y del tráfico de red contienen ataques que fueron creados y generados por el equipo de investigación.

El conjunto de datos describe las propiedades físicas del banco de pruebas en modo operativo. En total, se recolectaron 946,722 muestras con un paso de 1 segundo que comprenden 51 atributos durante 11 días.

En la figura 4.3 y 4.4 se puede ver cómo son las series temporales del sistema SWaT. La figura 4.3 corresponde al período donde se dejó el sistema funcionando de forma normal. Como se puede ver al principio estos datos tienen un transitorio que corresponde al encendido del sistema. La figura 4.4 muestras los datos del período donde empezaron los ataques hasta que terminó la prueba.

4.4.2. Tel2020

En el marco de un convenio de colaboración entre el Instituto de Ingeniería Eléctrica (IIE) de la Facultad de Ingeniería y la empresa Telefónica se ha venido trabajando en el desarrollo de algoritmos de detección de anomalías en series de telecomunicaciones. Producto de este convenio se tuvo acceso a diferentes variables, de las cuales se seleccionaron 10 para conformar el conjunto Tel2020. Éstas corresponden a valores sumados de los servicios utilizados por todo los usuarios

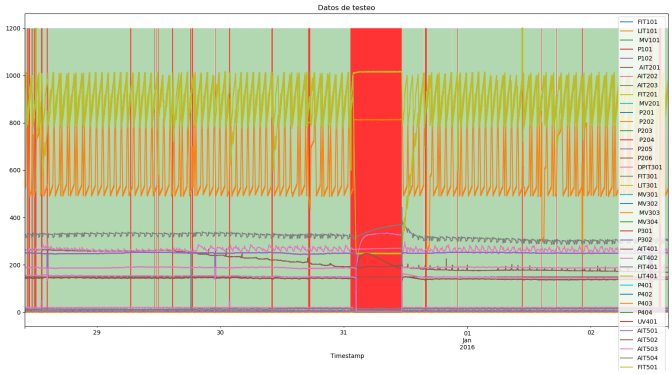


Figura 4.4: Datos correspondientes a los días en el que el sistema fue atacado. En color rojo se marca las zonas donde se produjeron los ataques.

en Uruguay, como costo y duración de llamadas, hasta largo de mensajes, entre otros. De estas variables se tienen disponibles los meses de enero, marzo, abril y junio, algo inusual cuando se compara con las bases de datos disponibles en forma pública. El contar con series temporales largas se considera esencial para capturar estacionariedad e información de contexto para detectar las anomalías. Los datos fueron normalizados entre $[0, 1]$ para conservar la privacidad de los mismos, y agregados cada 1 minuto. Estos datos no contienen etiquetas. En la tabla 4.1 se resumen los nombres de las 10 variables que componen el conjunto, y en la figura 4.5 se muestra un ejemplo, de manera de visualizar estos datos.

Variables
callCost
callDuration
callDuration_Hou
callDuration_Sec
callSetUpResultCode
consumptionMeter
creditNotCharged
messageLength
montoImpuesto
totalFactura

Tabla 4.1: Variables que componen el conjunto Tel2020.

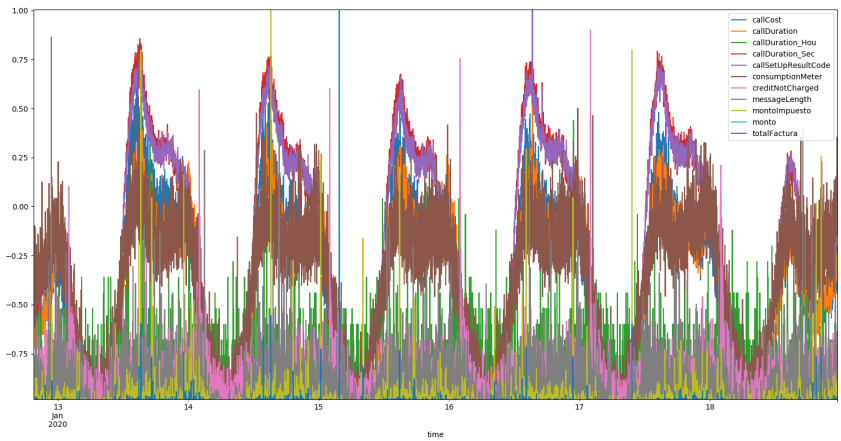


Figura 4.5: Ejemplo de una semana de datos del conjunto Tel2020. Aquí se puede apreciar la variedad de las variables, y que todas presentan un comportamiento estacional y estacionario.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Análisis y resultados

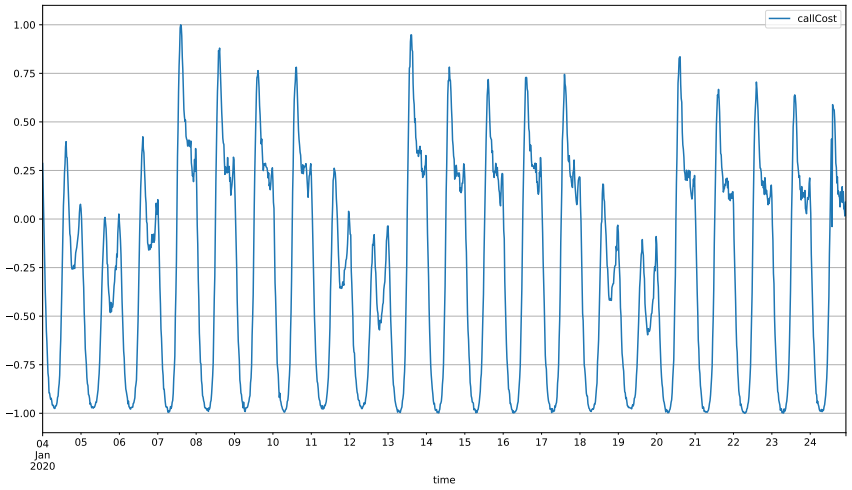
En el siguiente capítulo se pueden ver los diferentes experimentos realizados para los dos enfoques de detección de anomalías, a través de diferentes preguntas que se plantearon sobre los problemas que pueden surgir en la detección multivariable, y sobre la capacidad que pueden alcanzar los métodos planteados.

Ambos métodos están enfocados a analizar el residuo entre el dato normal que se está analizando y su representación mediante el modelo aprendido. Es por esto que lo primero que se quiere probar es la capacidad de generación, y reconstrucción para los métodos con GAN, y con VAE, respectivamente. Se espera que el Discriminador una vez entrenado devuelva una probabilidad baja para datos con los que no fue entrenado como son los datos anómalos, por lo que se prueba cómo es la reacción a diferentes tipos de anomalías (puntuales, de contexto, y colectivas) como pueden ser puntuales, de contexto, colectivas, para una sola variable o para varias. Al final se prueba el desempeño de ambos métodos para la detección de anomalías en la serie multivariable proveniente del conjunto SWaT por ser la que cuenta con las etiquetas de los datos anómalos.

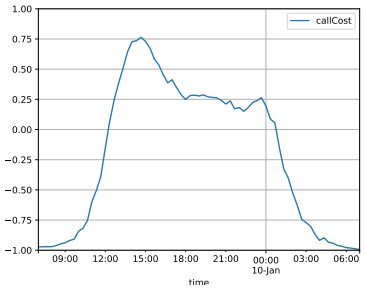
5.1. Generación de series

Previo a la detección de anomalías, es importante ver cómo funcionan los métodos para generar datos que se parezcan a los reales, ya sea a través de la generación y/o reconstrucción, dado que para que la detección tenga sentido es necesario que los métodos sean capaces de aproximarse a la distribución de los datos reales y normales.

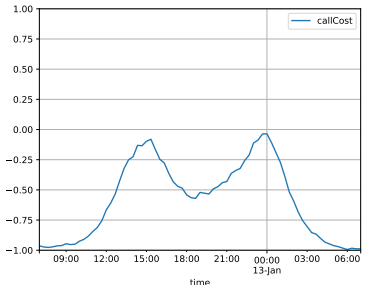
Los primeros ejemplos de generación, se realizaron sobre el conjunto Tel2020, descrito en 4.4.2, los cuales son datos reales pertenecientes a un mismo sistema de monitoreo. Este conjunto contiene diez variables en total con un paso temporal entre datos consecutivos del orden de los segundos, y cuenta con más de cuatro meses de información. Debido a esta cantidad y la estacionaridad que presentan los mismos (con un distinguible comportamiento diario y semanal) es que se eligió este conjunto para los primeros ejemplos, ya que permiten distinguir de forma fácil entre el comportamiento normal y el anómalo.



(a) Variable *callCost* posterior al agregado.



(b) Muestra correspondiente a los días laborales.



(c) Muestra correspondiente a un días no laborables.

Figura 5.1: La figura (a) permite observar el comportamiento de la variable *callCost* para aproximadamente 20 días donde se puede apreciar una diferencia en la forma entre los días laborales y los no laborales. Las figuras (b) y (c) muestran un ejemplo de ambas formas.

De manera de capturar la estructura del día (picos) y la noche (valles) en cada muestra, sin tener un largo de secuencia excesivo, se hizo un agregado promediando cada 20 minutos (1200 segundos), de esa manera para un largo de secuencia de 72 datos nuevos se obtienen muestras correspondientes a 24 horas. En la figura 5.1.a se puede ver como queda la primer variable de la serie (*callCost*) posterior al agregado, junto a dos ejemplos de muestras. La muestra de la figura 5.1.b corresponde a un día entre lunes y viernes, y la muestra de la figura 5.1.c se obtuvo de un fin de semana. Como se puede ver ambas tiene formas distintas por lo que es importante remarcarlo ya que se espera que los métodos sean capaces de generar muestras sintéticas de ambas. En la tabla 5.1 se especifican los parámetros para crear las muestras y determinar el conjunto de prueba.

Parámetro	Valor
aggregate	1200
seq_length	72
seq_step	1

Tabla 5.1: Parámetros para la creación de las muestras correspondiente al conjunto Tel2020.

Series univariable con escasos datos de entrenamiento

En el mundo del aprendizaje profundo, se sabe que los métodos funcionan bien, siempre y cuando se cuente con un volumen considerable de datos, y mientras más datos mejor se ajustan los modelos a la realidad, pero esto conlleva a más capacidad de almacenamiento, mayor poder de cómputo si se quiere procesar los mismos en un tiempo razonable. Para un sistema de monitoreo que maneja una gran cantidad de variables esto puede ser costoso. Una forma de resolver esto es tener un método de detección que sea capaz de actualizar sus parámetros de forma periódica o continua para un pequeña fracción de datos sin la necesidad de frenar la detección en tiempo real. Para las primeras pruebas solamente se utilizó el mes de enero tal como se muestra en la figura 5.1, quedando un conjunto de entrenamiento de 1436 muestras. Si bien pueden parecer insuficientes, es necesario probar si los métodos así mismo son capaces de acercarse a la distribución de los datos normales, de manera de que se puedan poner en producción y a medida que pase el tiempo y se vayan recolectando más datos, continuar con el aprendizaje del modelo.

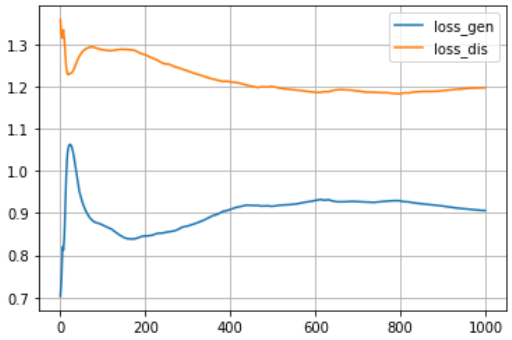
Fijando la misma cantidad de unidades para todas las redes $hidden\_units = 100$ de ambos métodos, y el espacio de latencia en la dimensión de los reales: $latent\_dim = 1$ la cantidad de parámetros totales para cada modelo de los métodos se puede ver en la tabla 5.2.

Para estas primeras pruebas se utilizaron 1000 épocas (*epochs*) para el entrenamiento de ambos métodos, donde en cada una el modelo utiliza todas las muestras respetando el orden temporal. También se usó el mismo tamaño de *batch*= 72 para ambos, de manera que dado que se tiene el mismo largo de secuencia y el corrimiento de la ventana es de a una muestra por vez, cada lugar dentro del *batch* mantendrá la misma hora del día. Esto permite a las RNN hacer uso del parámetro *stateful* y aprovechar el comportamiento que presentan estos datos.

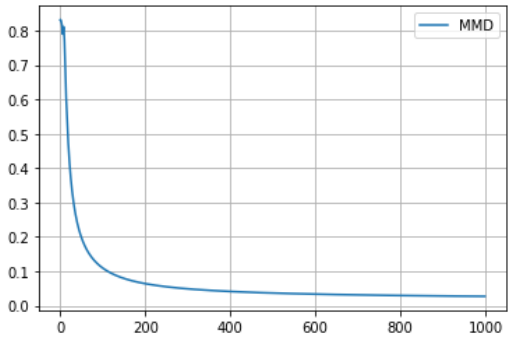
Durante el entrenamiento una manera de ver la evolución de los modelos es observar la evolución de sus funciones de pérdida a lo largo de las épocas. En la figura 5.2.a se pueden ver las funciones que corresponden al Generador (*loss_gen*) y al Discriminador (*loss_dis*) del método con GAN, donde de éstas se puede apre-

Modelo	Cantidad de parámetros
GAN	81802
VAE	14974

Tabla 5.2: Cantidad de parámetros a entrenar para cada modelo.



(a) Funciones de pérdida.



(b) MMD.

Figura 5.2: La figura (a) permite observar la evolución de las funciones de pérdida a través de las épocas para el Generador y el Discriminador. Donde se puede apreciar la competencia entre ambas redes, ya que cuando una crece la otra decrece y viceversa. En la figura (b) se puede apreciar la evolución del valor de MMD a través de las épocas. En este caso, esta gráfica permite observar como rápidamente el modelo es capaz de generar muestras que se parezcan a las reales, ya que la gráfica muestra una caída abrupta hasta la época 200, para luego estabilizarse.

ciar la competencia que se da entre las dos redes, ya que cuando una crece, la otra decrece y viceversa. Muchas veces el criterio para frenar el entrenamiento de las GAN es a través de visualizar la salida del Generador y determinar por parte del usuario si realmente dichas muestras parecen reales. En este caso al igual que [20] y [8] se utilizó la función MMD (ecuación 2.15) para medir la discrepancia entre las muestras generadas y las muestras de entrenamiento, la cual aporta más información con respecto a la generación que las funciones de pérdida, y además permite ver la evolución en la capacidad de generación del modelo. En la figura 5.2.b se puede ver cómo el valor de MMD decrece rápidamente a medida que crece la cantidad de épocas, hasta prácticamente mantenerse casi constante. Comparando las figuras 5.2.a y 5.2.b es interesante observar que si bien los valores de las funciones de pérdida no parecen estabilizarse en un valor particular, el valor de MMD se mantiene decreciendo la gran mayoría del tiempo.

En el caso de la función de pérdida para el modelo de la VAE, que no sólo

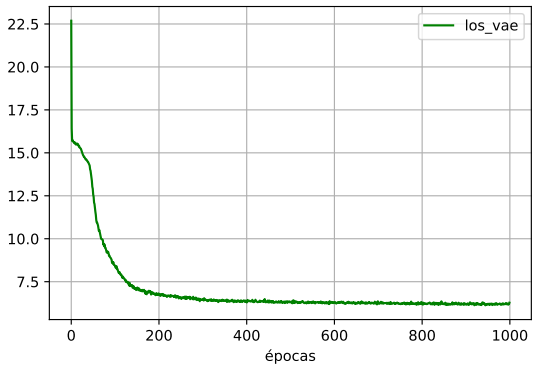


Figura 5.3: Evolución de la función de pérdida para el método con VAE. Generalmente en la práctica esta función de pérdida es: $-\mathcal{L}(\boldsymbol{x}_i, \boldsymbol{\theta}, \boldsymbol{\phi})$ (ecuación 2.3), por lo tanto lo que se busca es un mínimo de esta función. En este caso la gráfica muestra como el modelo converge a este mínimo, lo que indica que las reconstrucciones se acercan a las muestras reales.

representa el residuo entre las muestras y sus reconstrucciones sino que también la discrepancia entre la posterior $p(\mathbf{z}|\mathbf{x})$ y la priori $p(\mathbf{z})$, se puede constatar a través de la figura 5.3, cómo decrece rápidamente hasta mantenerse en un valor determinado. Este comportamiento y el del valor de MMD en la figura 5.2.b muestran que 1000 épocas fueron más que suficientes para el entrenamiento de ambos métodos en este caso.

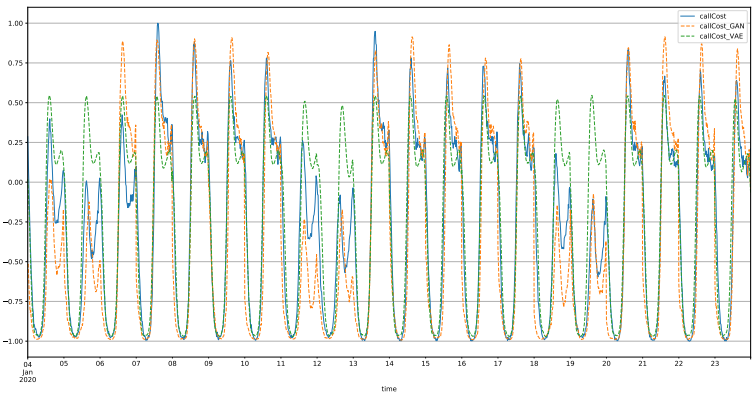
Una vez finalizado el entrenamiento, lo primero que se quiere ver es cuán parecidas son las muestras sintéticas y las reconstrucciones, a las muestras generadas. Como se explicó, el VAE a partir de una muestra a la entrada reconstruye la misma a la salida del modelo, por lo que alcanzaría con tomar cada muestra y pasarla por el modelo para obtener la reconstrucción. Pero además lo que hace interesante al modelo VAE con respecto a otros modelos de *Auto-Encoders* es que el *Encoder* devuelve una distribución en el espacio de latencia para cada muestra de entrada en vez de una sola muestra. Esto permite sortear cuantas muestras se quiera en este espacio para luego pasarlas por el *Decoder* y obtener tantas reconstrucciones como muestras sorteadas en el espacio de latencia. Si el modelo funciona bien, estas reconstrucciones no deberían variar demasiado entre sí dado que al fin y al cabo son reconstrucciones de la misma muestra. En el caso del modelo GAN, las muestras son generadas a partir de ruido, por lo tanto para encontrar la muestra sintética que más se parezca a la real es necesario hacer una búsqueda, que implica definir un criterio y los limites de hasta cuándo buscar. Al igual que en el entrenamiento se toma un *batch* de muestras aleatorias del espacio de latencia y se genera un *batch* de muestras sintéticas, luego para todas éstas se calcula la correlación entre la muestra real y las generadas y por último se selecciona la que devuelva el mayor valor, lo mismo se repite la misma cantidad que el tamaño de un *batch_size* para asegurarse de que se probó con todos los defasajes posibles. Este criterio se tomó así ya que no se encontró un patrón entre las muestras generadas de un mismo *batch*, como podría haber sido el corrimiento *seq_step* de igual forma

que los *batches* de muestras reales, y porque es necesario ponerle un límite de manera que cuando esto se utilice en la detección no lleve demasiado tiempo. Para ser consistente con el método de GANs, para el método de VAE por cada muestra de entrada se muestrea la misma cantidad de reconstrucciones que el tamaño del $batch\_size = 72$, y se selecciona la reconstrucción que tenga el mayor valor de correlación en 0 con la muestra real.

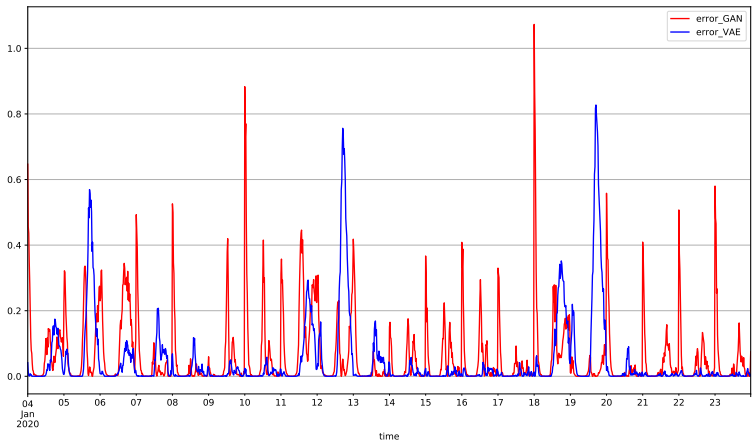
Primero se probó con las mismas muestras utilizadas en el entrenamiento para ver la generación con ambos modelos. En la figura 5.4.a como se puede ver, tanto las muestras reconstruidas por el modelo VAE (verdes), como las muestras generadas por la GAN (naranjas), se acercan bastante a la forma de las muestras reales (azules) para los días laborales, sin embargo no es tan así en los fines de semana, y el 6 de enero. Esto también se puede ver en la figura 5.4.b donde se muestra el error cuadrático para cada instante de tiempo.

Como se puede observar también existen una cierta cantidad de errores puntuales sobre todo para las muestras con GANs (rojo), en las zonas de crecimiento y decrecimiento de la serie, las muestras generadas no se ajustan tan bien a las reales y esto produce errores que se pueden ver con más detalle en la figura 5.5. Si bien este es un ejemplo puntual, esto deja ver los problemas que pueden surgir con un eventual desajuste o desfasaje entre las muestras sintéticas y reales. Este tipo de problema se vería reflejado en detecciones falsas, deteriorando el desempeño del método.

La siguiente prueba es ver qué sucede con la generación con datos no vistos por el modelo. Para esta prueba se utilizaron los datos desde el 10 al 30 de marzo del conjunto Tel2020, donde en este intervalo de tiempo se pueden ver no sólo días en que la forma se parece a las muestras de entrenamiento, sino que también entre el intervalo del 16 y 26 de marzo los datos reales muestran un aumento considerable a lo que se tiene establecido por lo normal. La variable *call_Cost* esta relacionada a los costos de las llamadas realizadas dentro de los servicios de Telefónica, por lo que claramente este aumento esta relacionado con el aumento en la comunicaciones a raíz del comienzo del confinamiento tras los primeros casos de COVID-19 en Uruguay. Si bien esto corresponde a una situación atípica, deja la puerta abierta para plantear la discusión de qué debería pasar con aumentos como estos que pueden no sólo corresponder a este tipo de situaciones sino a cambios a lo largo del año sobre todo con este tipo de variables asociadas al consumo. ¿Deberían ser detectadas como anomalías? o ¿deberían ser aprendidas por el modelo? Esto siempre dependerá del tipo de problema, de igual forma estos modelos son capaces de una vez entrenados y puestos en producción, de continuar su entrenamiento con nuevos datos, por lo que se podría entrenar el modelo periódicamente, y que el mismo vaya aprendiendo ese tipo de cambios. Fuera de esto se ve que salvo estos casos y los fines de semana el resultado es similar que los vistos en la figura 5.4.

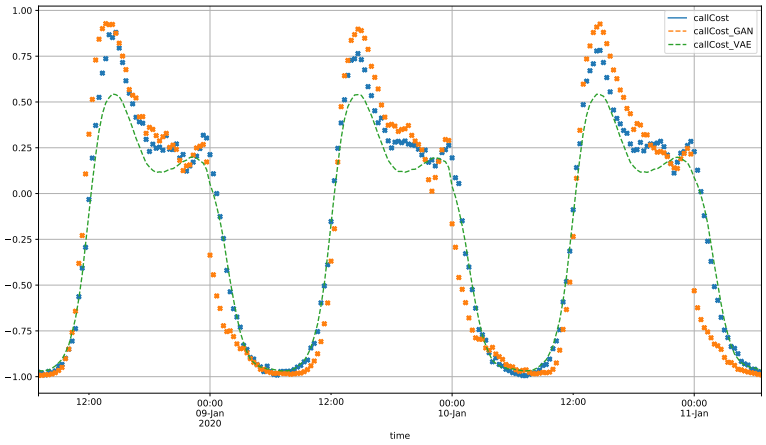


(a) Comparación entre las series sintéticas y la real para ambos métodos.

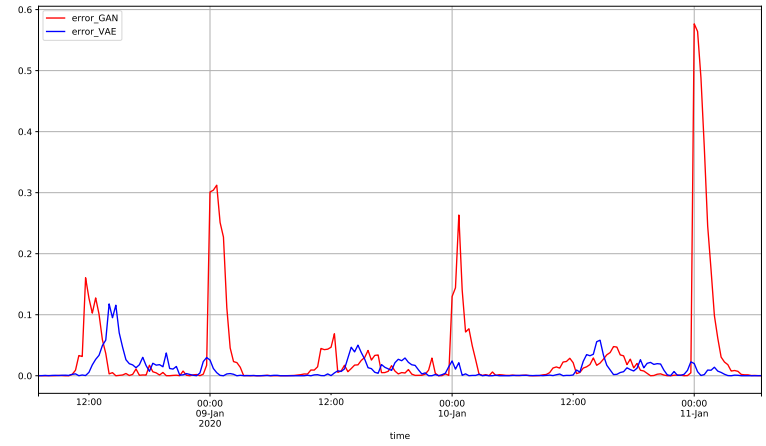


(b) Error cuadrático para cada instante de tiempo.

Figura 5.4: En la figura (a) se muestras tres series univariable, una es real y corresponde a datos utilizados en el entrenamiento de los modelos VAE y GAN, y las otras dos se construyeron a partir de muestras reconstruidas y generadas por estos modelos respectivamente. La figura (b) muestra el error cuadrático para cada instante entre la serie real y cada una de las series sintéticas.

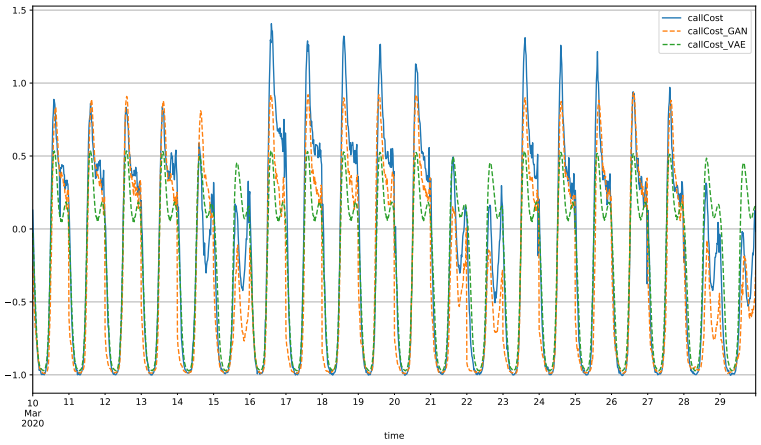


(a) Comparación entre las series sintéticas y la real para ambos métodos.

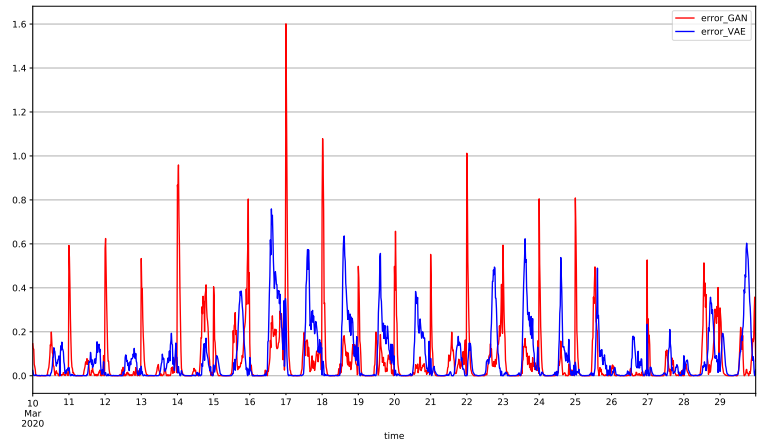


(b) Error cuadrático para cada instante de tiempo.

Figura 5.5: Acercamiento del ejemplo de la figura 5.4 donde se puede apreciar mejor las diferencias entre las series sintéticas y la real.



(a) Comparación entre las series sintéticas y la real para ambos métodos.

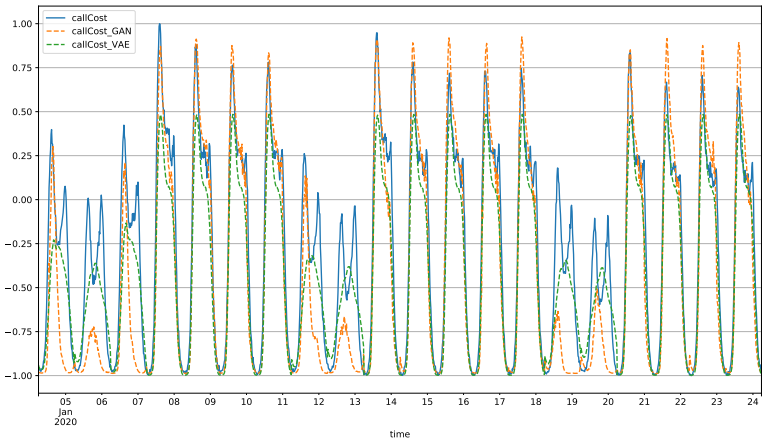


(b) Error cuadrático para cada instante de tiempo.

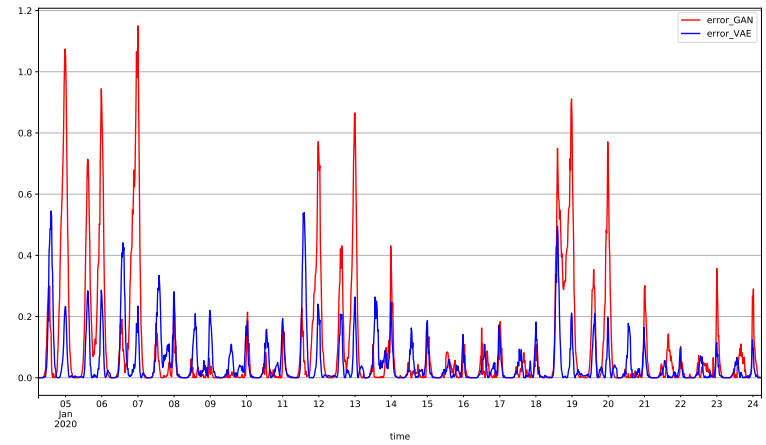
Figura 5.6: En este ejemplo también se compara una serie real con las series sintéticas creadas a partir de las muestras de ambos modelos, con la diferencia con respecto al ejemplo de la figura 5.4 que en este caso la serie real no fue utilizada en el entrenamiento de los modelos. En la figura (b) se puede apreciar que el error es mayor en comparación con el ejemplo anterior.

Series univariable agregando más muestras de entrenamiento

La siguiente prueba es para ver cómo reaccionan los modelos con un conjunto de entrenamiento mucho mayor. En este caso se utilizaron los meses de marzo, abril, y junio, para entrenar los modelos, quedando unas 6000 muestras, y el mes de enero para hacer la evaluación. Como se puede ver en la figura 5.7.a las muestras sintéticas se acercan a las muestras reales salvo en los fin de semana, como las demás pruebas. Esto último deja entrever un problema con el desbalance entre



(a) Comparación entre las series sintéticas y la real para ambos métodos.



(b) Error cuadrático para cada instante de tiempo.

Figura 5.7: En esta comparación también se utilizó una serie nunca antes vista por los modelos, pero estos fueron entrenados con un conjunto de datos mucho mayor en comparación con los ejemplos anteriores. Como se puede ver, se redujo el error con respecto a los días laborales si se compara con la figura 5.6, pero el error en los días no laborales es mayor.

tipos de muestras, dado que asimismo teniendo un conjunto de muestras razonable para el entrenamiento, nuevamente surge el mismo problema donde la generación parece ajustarse solo para el tipo de día más probable. Con respecto al resto de los días se puede ver en la figura 5.7.b que para esta prueba se mejoraron las generaciones del modelo GAN, ya que no se ven los valores de error puntuales tal como si se ven en las figuras 5.4.b y 5.6.b.

Series multivariable

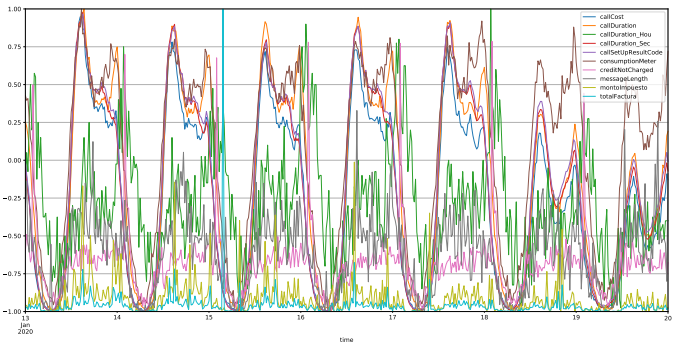
Vistos los alcances y las restricciones de los modelos para series unvivariables, lo que sigue es probar con series multivariables. Las siguientes pruebas se hicieron tomando todas las variables del conjunto Tel2020 de manera que el modelo aprenda el comportamiento de todas las variables del sistema a la misma vez. Se mantuvo tanto la cantidad de unidades de la capa oculta, como la dimensión del espacio de latencia con respecto a las pruebas anteriores, por lo tanto sólo cambian las entradas y salidas relacionadas a la cantidad de variables. Debido al crecimiento en las dimensiones, la cantidad de parámetros a optimizar aumenta. En la tabla 5.3 se muestra el total de los parámetros, donde comparando con la cantidad de parámetros de las pruebas anteriores (tabla 5.2) se puede ver como en el caso del modelo con GAN que utiliza series recurrentes hay un aumento de unos 8000 parámetros aproximadamente, mientras que para el modelo VAE con redes totalmente conectadas la cantidad de parámetros es diez veces mayor aproximadamente, pasando de 14974 a 145222. Esto es un punto de comparación interesante entre ambos métodos cuando se escala en la cantidad de variables.

En la figura 5.8.a se puede ver cómo es la forma de parte de los datos de entrenamiento utilizando todas las variables del conjunto Tel2020. Para el entrenamiento se mantuvo tanto la cantidad de épocas, como el tamaño de los *batches* con respecto a las pruebas anteriores. En la figura 5.8.b se ve un ejemplo de una serie a partir de muestras generadas por el modelo GAN, y en la figura 5.8.c, para el modelo VAE.

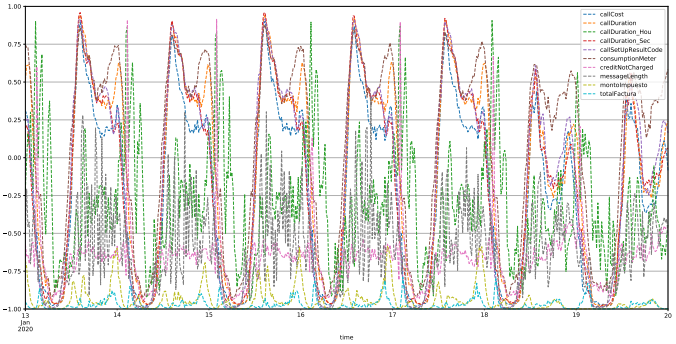
Como se puede ver en la figura 5.8, los resultados son similares a los alcanzados en las pruebas anteriores. En los días entre semana las muestras generadas se parecen bastante a las reales, las correspondientes al modelo GAN con menos ruido con respecto a las reales, y acompañan bien los valores de pico, mientras que las correspondientes al modelo VAE son mucho más suaves que las muestras reales y los valores en los picos están siempre por abajo. También entre las muestras del modelo GAN se ve una gran diferencia entre unas y otras, mientras que las muestras del modelo de las VAE se ven muy similares entre sí, producto de la suavidad que presentan.

Modelo	Cantidad de parámetros
GAN	89511
VAE	145222

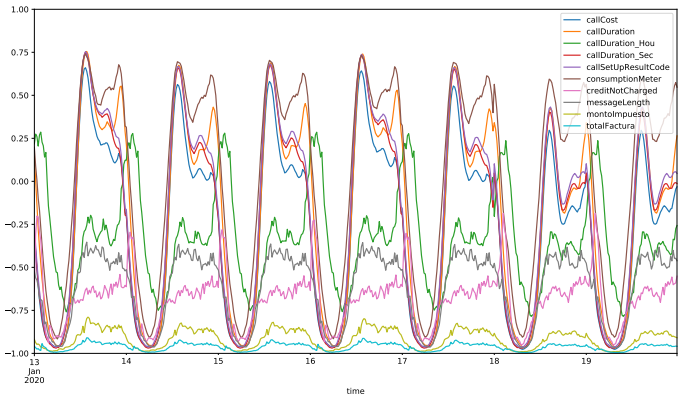
Tabla 5.3: Cantidad de parámetros a entrenar para cada modelo, cuando se utilizan todas la variables del conjunto Tel2020.



(a) Serie real con 10 variables.



(b) Serie generada a partir del modelo GAN.



(c) Serie generada a partir de las reconstrucciones del modelo VAE.

Figura 5.8: En la figura (a) se puede ver un ejemplo de una serie multivariable, que corresponde a utilzar todas la variables del conjunto Tel2020. Las figuras (b) y (c) muestran como queda esta misma serie según los modelos de GAN y VAE respectivamente. Comparando ambas series sintéticas se puede apreciar como la construida a partir de muestras generadas por el modelo GAN, captura muy bien la varianza de todas las variables. Donde por otro lado la generada a partir de muestras reconstruidas, se muestra mucho más "suave".

5.2. Detección

En la siguiente sección lo que se busca mostrar es si los métodos para la detección funcionan tal cual lo esperado.

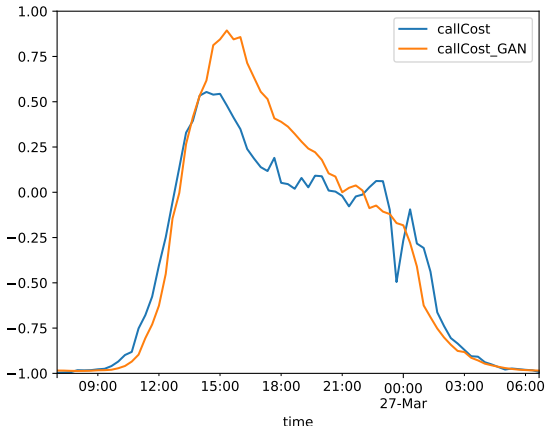
Como se explicó en la sección 3, el método con VAE basa la detección en la reconstrucción de las muestras, donde se mide el error cuadrático entre éstas para cada instante de tiempo, para luego mediante un determinado umbral decidir si este error es demasiado grande o no. Para crear este umbral en la fase de entrenamiento se debe reservar un conjunto de validación, y una vez entrenado el modelo se calcula la media y la desviación estándar del error cuadrático para este pequeño conjunto. Para la detección con el Generador en el método con GANs, se aplica el mismo criterio que en el método de VAEs. Siguiendo con los datos del conjunto Tel2020 si se toman los datos correspondiente a los meses de marzo, abril y junio en total se cuenta con 6000 muestras para entrenar el modelo, reservando las últimas 600 para la validación, se calculó el vector μ_{error} junto con la desviación estándar σ para ambos métodos. En la figura 5.9.a se muestra un ejemplo de comparación entre una muestra real de la variable *callCost* y la generada por el modelo GAN que le corresponde. En la figura 5.9.b se puede ver el error cuadrático entre estas para cada instante de tiempo, junto con la media del error μ_{error}, y los diferentes umbrales a partir de múltiplos de la desviación estándar relativos a la media. Una vez calculados los parámetros del error se determinará un valor anómalo para el instante i de la variable si se cumple la inecuación 5.1, donde $K \in \mathbb{R}^+$ controla la sensibilidad de la detección y debe ser seleccionado por el usuario. Esto se hace de manera independiente para todas las variables del sistema permitiendo no sólo saber en qué instante se produjo la anomalía sino en qué variable.

$$(x_i - x_i^*)^2 \quad > \quad \mu_{error} + K\sigma \tag{5.1}$$

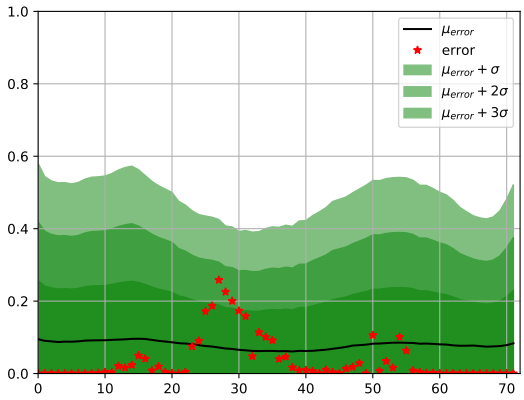
Lo siguiente es ver cómo funciona el Discriminador para la detección de anomalías y las ventajas y desventajas que presenta. En la sección 3, se mostró al Discriminador como un elemento que al igual que el Generador permitiría no sólo detectar el instante en que se produce un valor anómalo sino también en qué variable se presenta.

En la figura 5.10.a se puede observar una muestra multivariable perteneciente al mes de enero, que se considera que no tiene anomalías y no fue vista por el modelo. En la figura 5.10.b se muestra la salida del Discriminador para esta muestra, donde se la representa como una serie multivariable donde cada variable de ésta es la predicción correspondiente a cada variable de la muestra. Como se puede observar todas las predicciones están cerca del valor 1, lo que es razonable ya que la muestra es considerada normal.

En las siguientes pruebas se tomó la misma muestra y se le aplicaron anomalías sintéticas para observar cómo se comporta la salida. En la figura 5.11.a la muestra presenta una anomalía colectiva que afecta a todas las variables, donde en un intervalo grande de tiempo todas valen 1. En la figura 5.11.b se puede ver la salida del Discriminador, que claramente devuelve una probabilidad baja para este intervalo, y parte de sus alrededores.



(a) Comparación entre una muestra real y la generada por uno de los modelos.

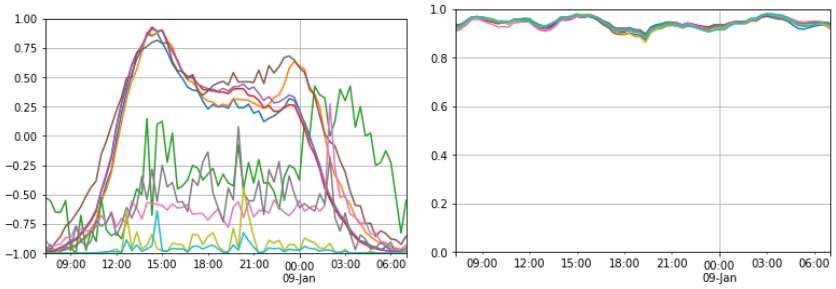


(b) Error entre las muestras para cada instante, junto con los umbrales de detección.

Figura 5.9: En este ejemplo se busca mostrar de forma gráfica, cómo es la detección a partir del residuo tanto para el método con VAE como para el método con GAN. En la figura (a) se compara una muestra real y una generada por uno de los dos modelos. En la figura (b) se ve el error cuadrático entre ambas para cada instante, junto con los parámetros calculados en la fase de validación: el error medio μ_{error} y diferentes umbrales a partir de sumar $K\sigma$.

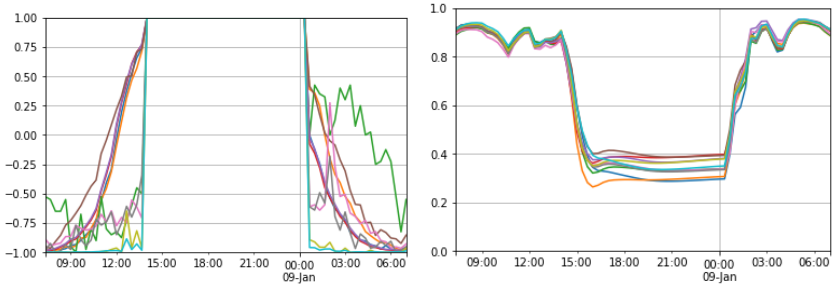
En la siguiente prueba para un determinado instante se llevó toda las variables al valor -1, simulando una anomalía puntual y de contexto como se muestra en la figura 5.12.a. En la figura 5.12.b se muestra la salida del Discriminador y se puede ver claramente un pico que indica una baja probabilidad para es instante como era de esperar.

Por último, se aplicaron anomalías puntuales en tres variables por separado como se muestra en la figura 5.13.a. Una para la primer variable (azul) antes de



(a) Muestra normal. (b) Salida del Discriminador.

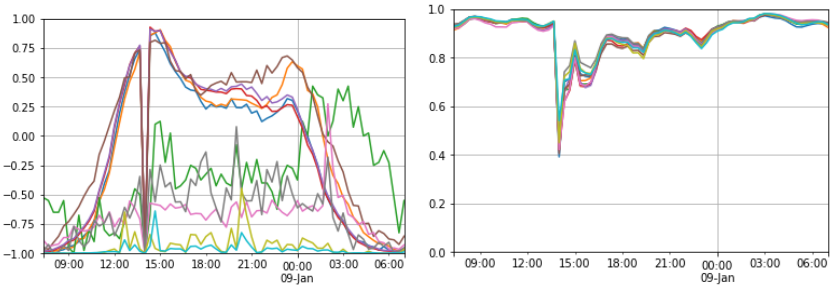
Figura 5.10: A partir de una muestra normal (a), se puede apreciar la salida del Discriminador (b). Como se puede ver la salida es cercana a 1 para todos los instantes lo que era de esperar ya que la muestra no presenta anomalías.



(a) Muestra con anomalía. (b) Salida del Discriminador.

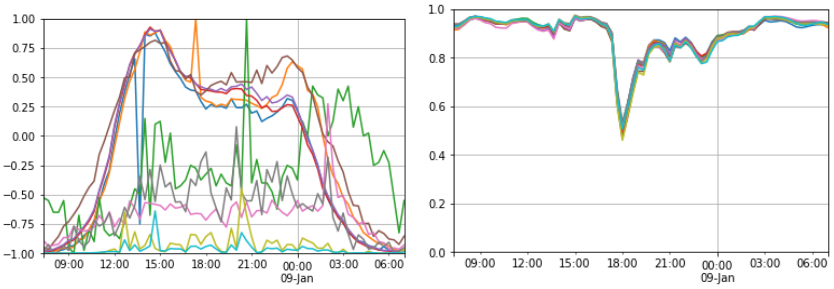
Figura 5.11: Forzando una anomalía colectiva en todas las variables a la vez sobre la muestra del ejemplo anterior como se muestra en (a), se puede apreciar en (b) como responde el Discriminador a la misma. Se puede ver que durante el intervalo en el que se produce la anomalía, la predicción devuelve valores bajos al rededor de 0.4 para todas las variables.

las 15:00 horas, otra en la segunda variable (naranja) cerca de las 18:00 horas, y otra a la tercer variable (verde) cerca de las 21:00 horas. Como se puede ver en la figura 5.13.b el discriminador sólo reacciona a una de las anomalías y lo hace para todas las variables. Efectuadas más pruebas como estas se determinó que el discriminador sólo puede ser capaz de detectar algunas anomalías que se presentan en una sola variable y de hacerlo lo manifiesta en todas las variables, lo que imposibilita discriminar en cuál de todas se dio el evento.



(a) Muestra con anomalía. (b) Salida del Discriminador.

Figura 5.12: Esta vez se forzó a que la muestra presente una anomalía puntual y de contexto en todas las variables a la vez como se ve en (a). Se puede ver en (b) que la salida del Discriminador responde con un valor bajo en el instante que se produce la anomalía, con respecto a los demás valores que componen la salida.



(a) Muestra con anomalías. (b) Salida del Discriminador.

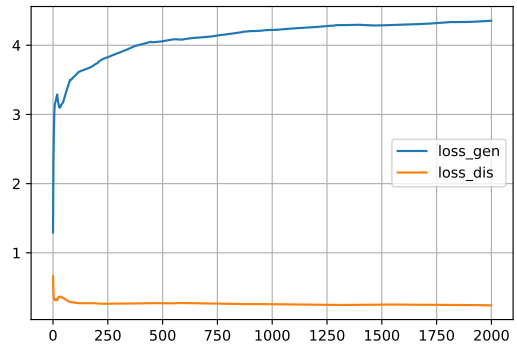
Figura 5.13: En este ejemplo se buscó observar la respuesta del Discriminador a anomalía que solo se presentan en alguna de las variables de la series. Forzando tres anomalías puntuales distintas en las variables: azul, naranja, y verde, se puede ver en (b) que el discriminador reacciona para una sola de ellas, y en este caso el valor no es tan pequeño como en las anomalías de los ejemplos anteriores.

5.3. Validación de los métodos

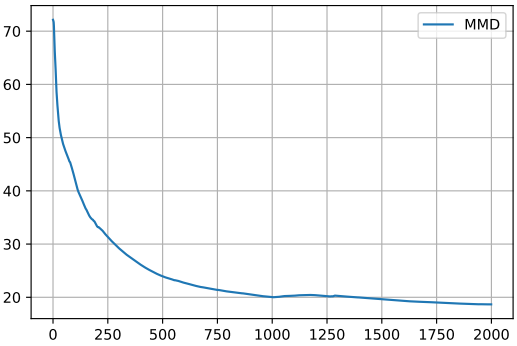
Para evaluar el desempeño de los métodos es necesario utilizar un conjunto de datos que esté etiquetado, por lo que para estas pruebas se utilizó el conjunto SWaT (4.4.1). Este conjunto pensado para la detección de anomalías cuenta con 51 variables que proviene de medidas de diferentes sensores y de las comunicaciones de los mismos, lo que lo hace un conjunto rico para evaluar el desempeño de los métodos. Este conjunto está dividido en dos subconjuntos, uno sólo con datos donde el sistema funcionó de manera normal (figura 4.3) el cual fue pensado como conjunto de entrenamiento, y el otro (figura 4.4) corresponde a datos don el sistema funcionando de manera normal fue atacado reiteradas veces. Todos los datos pertenecientes a los intervalos de ataque fueron etiquetados como anómalos, por lo tanto este subconjunto al tener datos de ambas clases, fue pensado por sus autores como conjunto de prueba. Ambos subconjuntos corresponden a unos poco días de captura de datos, y analizando los datos más detenidamente se puede observar una cierta periodicidad de 70 minutos en las variables de mayor varianza, teniendo esto en cuenta es que se aplicó el siguiente procesamiento. En 70 minutos hay 4200 segundos, por lo tanto con un agregado de 100 segundos (*aggregate*), en un período entran 42 datos lo cual es un largo de secuencia razonable ($T = length\_seq$). Si se combina con un paso de 1 dato ($W = seq\_step$) y un largo de *batch* igual ($batch\_size = 42$), se puede aprovechar el uso del parámetro *stateful*. Fijando estos parámetros, se tiene un total de 4098 muestras de entrenamiento y 4458 muestras para validación, estas últimas corresponden a 187236 datos nuevos, de los cuales 164593 están etiquetados como normales y 22643 como anómalos. Cabe notar que estás cantidades se deben al largo de secuencia y al solapamiento ente muestras, la gran mayoría de los datos se evaluará las misma cantidad de veces que el largo de secuencia, pero será evaluado en diferentes posiciones a lo largo de la secuencia. En la tabla 5.4 se resumen los parámetros de procesamiento, y en la tabla 5.5 la cantidad de datos nuevos para la validación de los métodos. Como se puede ver en esta última, el problema que se presenta con el conjunto SWaT es un problema desbalanceado como generalmente son los problemas de detección de anomalías. La normalización se hizo con la función *minmax[-1,1]*, y el resto de los parámetros se mantuvieron iguales con respecto a las pruebas anteriores.

Parámetro	Valor
aggregate	100
seq_length	42
seq_step	1

Tabla 5.4: Parámetros del preprocesamiento aplicado a los datos del conjunto SWaT.



(a) Funciones de pérdida.



(b) MMD.

Figura 5.14: Evolución de las funciones de pérdida (a) y de la función MMD (b) a través de las épocas del entrenamiento del método con GAN con los datos del conjunto SWaT. Sobre todo en (b) se puede ver que el modelo logra ajustarse a la distribución real de los muestras observadas.

5.3.1. Método con GAN

Para el entrenamiento del método con GAN se utilizaron 2000 épocas, como se puede ver en las figuras 5.14.a y 5.14.b, donde se muestran las evoluciones de las funciones de pérdida y del valor MMD respectivamente. En total se actualizaron 110852 parámetros y el entrenamiento duró aproximadamente 7 horas.

Una vez entrenados el Discriminador y el Generador, estos se pueden evaluar

	Datos	Porcentaje
Normales	164593	88 %
Anómalos	22643	12 %

Tabla 5.5: Cantidad de datos para la evaluación.

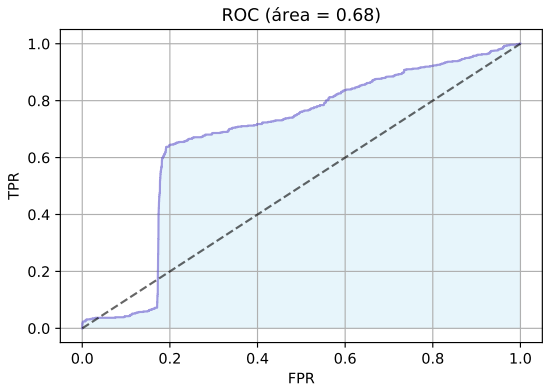


Figura 5.15: Curva ROC generada a partir de diferentes umbrales para la comparación de las detecciones con el Discriminador. Donde se muestra la relación entre el acierto en la detecciones de anomalías (TPR), y las falsas alarmas (FPR) para diferentes umbrales.

como detectores por separado. Es importante mantener el mismo preprocesamiento y el mismo criterio de normalización para los datos de evaluación con respecto a los de entrenamiento, de manera de mantener una coherencia sobre el domino en el que se está trabajando. Empezando por el Discriminador, tal como se explicó en 3.2.2 éste devuelve una secuencia de probabilidades $D(X) = [y_{p1}, ..., y_{pT}]$ para cada secuencia de entrada X, que indica la probabilidad de que el lugar que ocupan corresponda a un dato real. Para mantener el mismo criterio que las etiquetas, que valen $y_i = 1$ para los instantes anómalos y $y_i = 0$ para los normales, se tomaron como predicciones los complementos de las probabilidades, $\mathbf{1} - D(X) = [1 - y_{p1}, ..., 1 - y_{pT}]$. De esta forma, la predicciones altas corresponderán a valores anómalos y las bajas a valores normales. Luego es necesario elegir un umbral u_D que decida qué es una probabilidad alta y qué es una probabilidad baja. Utilizando diferentes valores de umbral, se graficó la curva ROC que muestra el desempeño en cuanto a las detecciones. En la figura 5.15 se puede ver su forma, y se observa que para cierto umbral se detecta un 60 % de las anomalías, estando por debajo del 20 % de falsos positivos.

Además en la figura 5.16 se grafica la relación entre los indicadores *Precision* y *Recall* para diferentes umbrales, junto con diferentes límites del valores de F1 que combina ambos indicadores a través de la media armónica. Como se puede ver, el valor máximo alcanzado es de $F1 = 0{,}40$, el cual corresponde a un umbral de probabilidad de $u_D = 0{,}98$. Esto último indica que en este caso se considera a un valor anómalo sólo cuando el Discriminador este muy seguro. Asimismo se tiene un bajo valor de *Precision* $= 0{,}29$, lo que indica que el discriminador devuelve muchas falsas predicciones sobre datos que son normales. El valor de *Recall*$= 0{,}65$, muestra que con este valor de umbral tan alto se deja una parte considerable de las anomalías sin detectar. Las cantidades de detecciones se pueden ver en los números representados en la matriz de confusión que se muestra en 5.17.

Por otro lado con el error de reconstrucción a partir del modelo del Generador

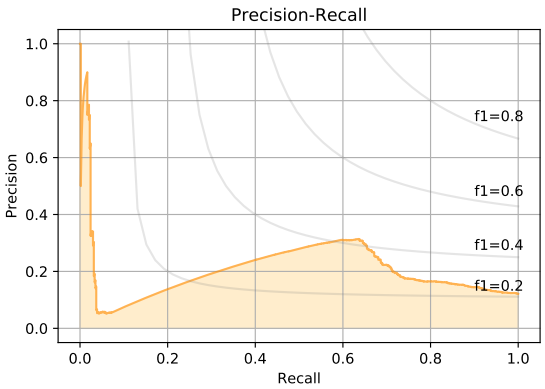


Figura 5.16: Curva *Recall-Precision*, la cual muestra la relación entre ambos indicadores a partir de diferentes umbrales para la comparación de las detecciones con el Discriminador. Además, se muestran diferentes umbrales de valores de F1, el cual es el indicador que se quiere maximizar para encontrar el umbral óptimo.

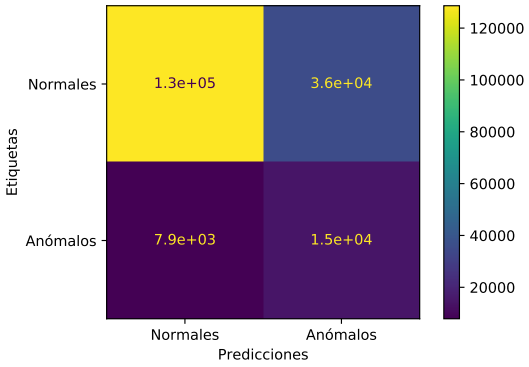


Figura 5.17: Matriz de confusión de la detección con el Discriminador, para un umbral de probabilidad de 0.98. Ésta permite ver en cantidades el desempeño de la clasificación entre Normales y Anómalos.

los resultados son diferentes. Dado que las etiquetas son por intervalo de tiempo, el error de reconstrucción entre las muestras reales y las generadas se calcula como el promedio a lo largo de todas las variables para cada instante de tiempo. Para encontrar la muestra generada que más se parezca a cada muestra real, se generaron por cada muestra real, 1764 ($batch^2$) muestras sintéticas. Esto último se debe a que cada vez que se invoca al Generador se deben utilizar todo un *batch* de muestras del espacio de latencia y se obtiene todo un *batch* de muestras generadas, luego la cantidad de *batches* que se quieran generar queda a criterio del usuario, en este caso se eligió el tamaño del *batch*.

En la figura 5.18 se muestra la curva ROC a partir de la detección por reconstrucción. Como se puede ver en este caso hay una notable mejora comparada con la curva de la figura 5.15, ya que el área en este caso es de 0.84. También la curva

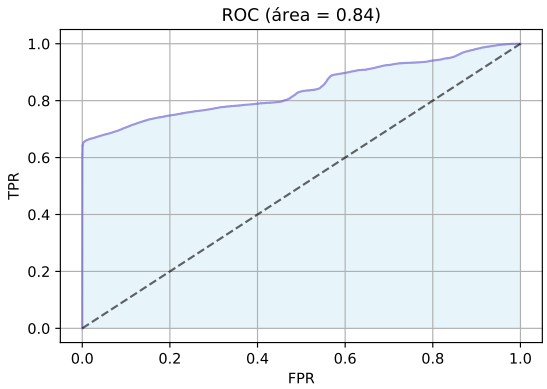


Figura 5.18: Curva ROC a partir de diferentes umbrales para la detección por residuo para el método con GAN. En este caso se puede ver que se alcanza un área mayor con respecto a la detección con el Discriminador, lo que indica un desempeño mejor.

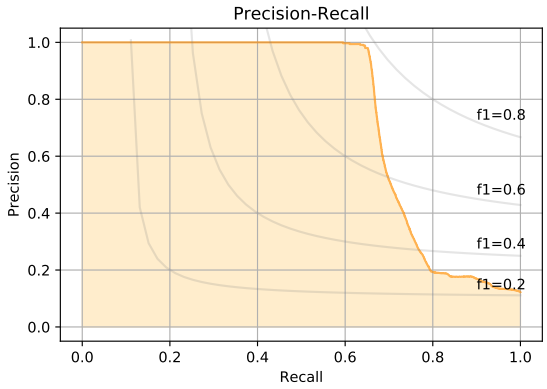


Figura 5.19: Curva Recall-Precision y valores de F1 a partir de diferentes umbrales para la detección por residuo para el método con GAN. En este caso se puede ver que se pueden alcanzar valores de F1 mucho más grandes en comparación con los resultados del Discriminador.

muestra que se pueden alcanzar valores de detección de anomalías por arriba del 60 %, para valores casi nulos de falsos positivos. En la figura 5.19 se muestra la curva *Recall-Precision*, la cual muestra también un gran mejora con respecto a la curva de la figura 5.16. El valor máximo alcanzado para $F1 = 0{,}78$, es casi el doble que el anterior, y esto se alcanza para un umbral de $u_{error} = 4{,}1$. Este umbral es un valor grande considearando que se está evaluando el error cuadrático y los valores normales de los datos están en el rango $[-1, 1]$. Dado esto y que el valor *Recall*$= 0{,}65$ es el mismo que se obtuvo con el Discriminador da indicios de que se detectan las mismas anomalías en ambos casos. Lo que incrementa el valor de $F1$ con respecto al caso anterior es el valor de *Precision*$= 0{,}98$, el cual se debe a la baja de falsos positivos. Esta baja también se puede ver en la matriz de confusión en 5.20.

Presentados los resultados de detección para ambas redes del método con GAN

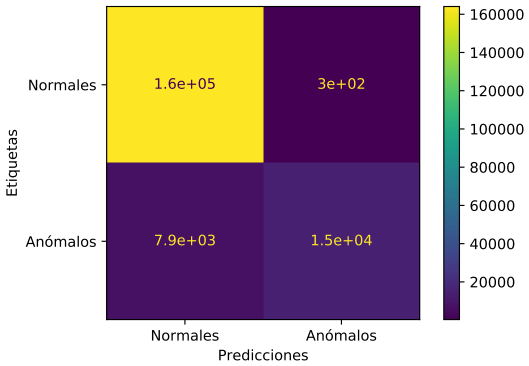


Figura 5.20: Matriz de confusión a partir de la detección por residuo, para un umbral de error cuadrático de $u_{error} = 4{,}1$.

se puede ver que la detección por residuo se desempeña mejor en este caso con respecto a la detección por discriminación. La gran ventaja de este último es que sólo se necesitó de algunos segundos para evaluar todos los datos de evaluación, mientras que la otra detección demoró casi 2 horas, debido a la búsqueda de la muestra generada que más se parezca a la real.

5.3.2. Método con VAE

Manteniendo todo tal cual la prueba anterior para el preprocesamiento de los datos y manteniendo los mismo parámetros que las pruebas anteriores realizadas con este método, se entrenó el modelo con el conjunto SWaT. En la figura 5.21 se puede ver cómo evolucionó la función de pérdida en función de las épocas. En total se actualizaron 431044 parámetros y el entrenamiento duró apenas 36 minutos.

Para la detección, por cada muestra real se muestrearon la misma cantidad de reconstrucciones que el tamaño de los *batches*. Si bien en este caso las reconstrucciones para una misma muestra real varían muy poco entre sí, tiene sentido utilizar más de una para calcular el error de reconstrucción. Este se calculó para todas las reconstrucciones a lo largo de las variables para cada instante de tiempo al igual que la detección con el Generador. Luego de todos los valores se seleccionó el mínimo para cada instante de tiempo. Al igual que el método con GAN se graficaron la curva ROC y *Recall-Precision*, en las figuras 5.22, y 5.23 respectivamente. Como se puede observar el desempeño es prácticamente el mismo que obtenido con el Generador para el método con GAN. Esto también se manifiesta en el valor máximo de *F1*, salvo que el umbral esta vez es de $u_{error} = 3{,}26$. La matriz de confusión 5.24, también muestra similitud con la de 5.20.

Por último, en la figura 5.20 se muestra un ejemplo de reconstrucción, y evaluación. En la figura 5.20.a se puede observar una muestra real con dos anomalías, confirmadas por las etiquetas en la figura 5.20.b. Por otro lado se muestra en la figura 5.20.c, la reconstrucción, que como se puede ver se parecen con la muestra real. En la figura 5.20.d se puede ver el error cuadrático entre la muestra y su

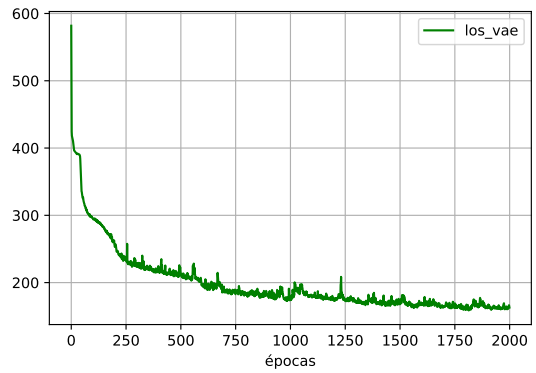


Figura 5.21: Evolución de la función de pérdida a través de las épocas de entrenamiento para el método con VAE. Como se puede apreciar la función converge a un cierto valor, lo que indica que el modelo se aproxima a la distribución real de las muestras.

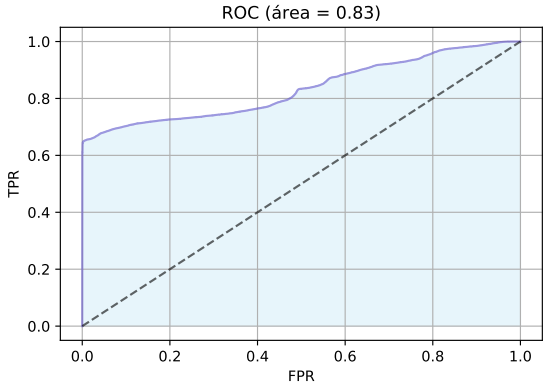


Figura 5.22: Curva ROC a partir de diferentes umbrales para la detección por residuo para el método con VAE. Si se compara con la curva ROC de la detección por residuo con el método con GAN (figura 5.18) se puede apreciar que el desempeño es muy similar.

reconstrucción, donde los valores parecen manifestarse para las anomalías etiquetadas (picos a la derecha), pero también muestra algunos presuntos falsos positivos (picos a la izquierda).

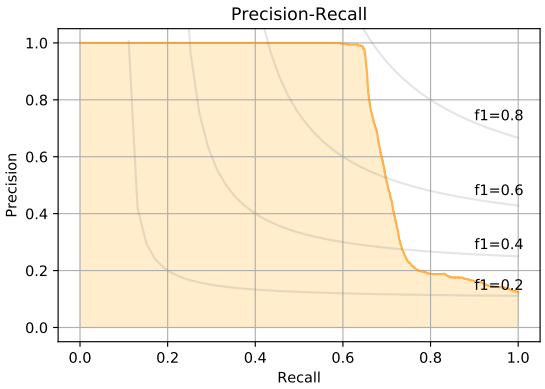


Figura 5.23: Curva Recall-Precision y valores de F1 a partir de diferentes umbrales para la detección por residuo para el método con VAE. Al igual que la ROC, esta curva muestra un desempeño similar al de la curva para la detección por residuo con el método con GAN.

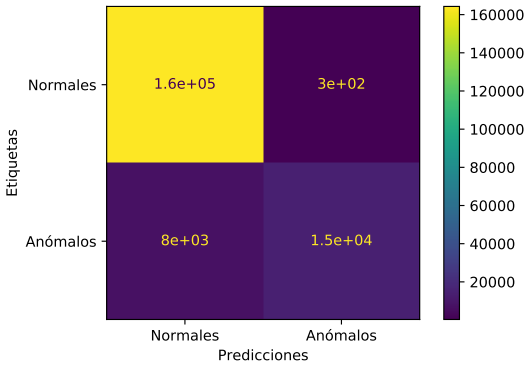
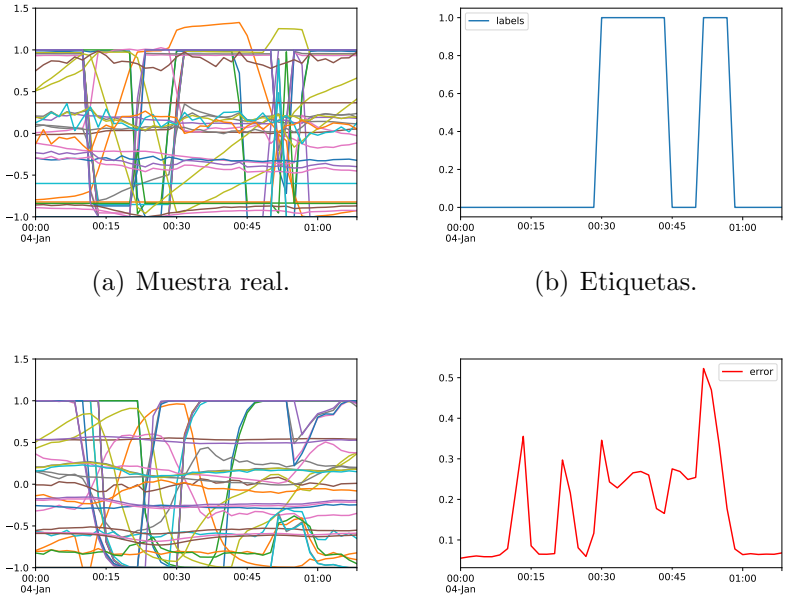


Figura 5.24: Matriz de confusión a partir de los resultado de la detección por residuo, para un umbral $u_{error} = 3{,}26$.



(a) Muestra real.

(b) Etiquetas.

(c) Reconstrucción.

(d) Error cuadrático.

Figura 5.25: Ejemplo de una muestra real (a) junto con sus respectivas etiquetas (b), las cuales muestran dos anomalías. Junto a éstas se puede ver la reconstrucción a partir del modelo VAE (c), y el error cuadrático (d) entre ésta y la muestra real. Si se comparan (b) y (d), se puede ver que en este caso se podría detectar ambas anomalías, pero también algunas falsas alarmas.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Conclusiones y Trabajo a futuro.

En este trabajo se estudiaron, implementaron, y evaluaron dos métodos semi-supervisados para la detección de anomalías en series multivariables, donde el enfoque de ambos es aprender el comportamiento normal de las series, y en base a esto detectar anomalías como todo aquel comportamiento que se aparte de lo aprendido. El primer método basado en el modelo clásico de los VAEs, aprende a generar pequeñas secuencias de datos utilizando una ventana móvil sobre las series temporales. El segundo entrena dos redes recurrentes (RNN) bajo el enfoque adversario de las GANs, de manera que la que actúa como Generador sea capaz de generar series sintéticas que se comporten de manera normal, y la otra como Discriminador sea capaz de diferenciar cuando una serie no se comporta de manera normal.

Ambos métodos fueron evaluados con dos conjuntos de datos reales, de 10 y 51 variables diferentes, donde el primero fue utilizado para observar la capacidad de generación de series sintéticas, y el funcionamiento y alcance de las técnicas de detección, mientras que el segundo se utilizó para evaluar el desempeño de ambos como detectores de anomalías.

En primera instancia, las diferentes pruebas que se hicieron demostraron la buena capacidad de los métodos para generar muestras sintéticas comportándose de manera diferente. Además se pudo observar el potencial de ambos para la detección, como también así los problemas que pueden surgir con el uso de estas herramientas. Se pudo observar cómo es el desbalance entre tipos de comportamientos dentro de las series, donde ambos se ajustaron bien al comportamiento de las series durante los días laborales y no así para el comportamiento en los días de fin de semana, lo que puede ser un problema para el desempeño de los detectores si no se trata estos tipos de datos por separado.

Luego se evaluó el desempeño de la detección de anomalías, donde se puso a prueba ambos métodos ante un problema real de monitoreo de 51 variables a la vez. El método basado en VAEs mostró un buen desempeño detectando el 65 % de las anomalías, prácticamente sin falsas detecciones. Para el método que combina RNNs y GAN, se observó que el Discriminador puede ser una herramienta demasiado sensible para la detección de anomalías, ya que se registraron una proporción grande de falsas detecciones. Por el otro lado con el Generador se logra mejorar

bastante en cuanto a las falsas detecciones, pero esto tiene un costo en tiempo ya que para cada muestra que se quiera evaluar, es necesario hacer una búsqueda que es prácticamente aleatoria. Para ambos métodos se alcanzaron los resultados del estado del arte sin la necesidad de perder generalidad de los datos reduciendo la dimensión.

Ambos métodos fueron implementados con la librería *keras*, que permitió fácilmente trabajar con herramientas de aprendizaje profundo, y resumir en pocas lineas de código la implementación lo que lo hace legibles para los usuarios que quieran continuar con este trabajo. Otra ventaja de la implementación es que luego de entrenados los modelos, el usuario puede guardarlos en un archivo y una vez puestos en producción estos se pueden continuar entrenando en forma paralela sin interrumpir la detección.

En el proceso de implementación también se pudo notar que los métodos presentan algunas desventajas. La primera es la necesidad de normalización de las variables para trabajar con redes neuronales profundas. Abarcar todas las variables de un sistema implica relacionar variables de diferentes tipo, por lo que la normalización debe hacerse para cada una por separado. Esto puede hacer tomar relevancia a variables que no eran relevantes dentro del sistema y hacer que el método desvíe la atención de otras variables que sí eran relevantes. También normalizar implica poner preciso cuidado en que los datos de entrenamiento no presenten valores espurios, un solo valor de este tipo puede perjudicar gravemente, modificando la forma de la variable posterior al normalizado y así afectar el desempeño de la detección. La segunda desventaja es la imposibilidad de agregar una nueva variable una vez puestos los métodos en producción. Agregar una nueva variable cambiaría completamente el dominio del problema, por lo que dado el caso se debería entrenar un nuevo modelo.

Esta tesis también deja la puerta abierta para futuros trabajos. Lo primero es la puesta en producción en una plataforma de análisis de datos masivos y evaluar el desempeño en tiempo real para un problema real. Sobre posibles mejoras de los métodos y nuevos enfoques, queda por mejorar la búsqueda inversa en el métodos basado en GANs de forma que sea más eficiente. Una forma de hacer esto es usar GAN condicionales, en el caso de datos como los de Tel2020 que contiene un comportamiento bien marcado para cada día de la semana, fácilmente se podría extraer de las marcas de tiempo de cada dato el día de la semana al que pertenece y utilizar esta información para condicionar a la GAN. Esto permitiría indicarle al modelo qué día de la semana se está analizando y que este sólo devuelva muestras sintéticas correspondientes al mismo. De manera más compleja existen métodos de GANs, que entrenan junto con un *encoder* que va del espacio de los datos al espacio de latencia, lo cual una vez entrenado, para cada muestra éste devuelve el vector latente que genera la muestra sintética que más se le parece. Además para la detección mediante el error de reconstrucción en ambos métodos se debe fijar un umbral que determina la sensibilidad del método, la cual puede cambiar mucho entre un problema y otro si en vez de generar muestras ambas devolvieran una distribución completa, se podría medir la probabilidad de que la muestra pertenezca a la distribución aprendida lo que lo hace mucho más intuitivo para el

usuario. Estos dos últimos enfoques se pueden encontrar en [5].

Parte del trabajo de esta tesis fue presentado y aceptado en dos conferencias internacionales

- *TMA Conference 2020*, *Network Traffic Measurement and Analysis Conference*, en Berlin, Alemania. Bajo el título: *Net-GAN: Recurrent Generative Adversarial Networks for Network Anomaly Detection in Multivariate Time-Series*
- *SIGCOMM '20 posters and demos*, *ACM Special Interest Group on Data Communication*, en Nueva York, EEUU. Bajo el título: *Network Anomaly Detection with Net-GAN, a Generative Adversarial Network for Analysis of Multivariate Time-Series*

Esta página ha sido intencionalmente dejada en blanco.

Apéndice A

"Trucos" para el entrenamiento de las GANs

Dada la cantidad de parámetros y las formas de entrenar una GAN, existen algunas publicaciones como [10], [27], donde se presentan algunos "trucos" y técnicas para mejorar el rendimiento de este tipo de redes. Según ellos mismo los "trucos" son prácticas que valen la pena probar, pero no deben ser tomadas como la mejor práctica.

Etiquetas suavizadas

Las GANs funcionan bien cuando el discriminador es capaz de estimar una relación entre dos densidades, pero las redes neuronales profundas tienden a devolver probabilidades extremas. Para apoyar a que el discriminador devuelva probabilidades más suaves se puede usar una técnica llamada *one-sided label smoothing* [27]. Esta técnica es simplemente relajar las etiqueta correspondiente a los datos reales cuando se entrena el discriminador, como por ejemplo $\alpha * \mathbf{1}$ con $\alpha = 0{,}9$. También se puede usar una variable uniforme entre dos valores de manera de obtener para cada *batche* un valor distinto que multiplique a las etiquetas $\mathbf{1}$. Los parámetros *lim_max_0* y *lim_max_1* que se muestran en la tablas de parámetros A.1, son para fijar estos dos valores. En cuanto al valor óptimo del discriminador ($D*$), que es con el que se supone que se trabaja, esto es simplemente una reducción por el valor utilizado. En cambio es importante no suavizar las etiquetas de las muestras falsas ($\beta + \mathbf{0}$), porque esto produciría un cambio en la forma del discriminador óptimo.

Equilibrar G y D

Puede ser intuitivo pensar en equilibrar a los jugadores G y D para evitar que uno domine sobre el otro. Una forma es cambiar la forma de las redes. En este caso en que ambas son redes recurrentes donde la cantidad de celdas queda determinada por el largo de secuencia, la forma de cambiar el tamaño de las redes con respecto una de la otra es desbalancear la cantidad de unidades del vector de estados ocultos h. Los valores *hidden_units_g* y *hidden_units_d* son los que fijan

estos tamaño.

Otra forma de equilibrar es entrenar k-veces una red por cada vez que se entrena la otra. Esto es, para cada vez que una red actualiza sus parámetros con un *batche* de muestras, la otra actualiza k-veces sus parámetros para k *batches* distintos. Estas cantidades se controlan con los parámetros *G_round*, y *D_round* para el generador y el discriminador respectivamente.

Nombre	Valor
aggregate	1200
seq_length	72
seq_step	1
num_generate_features	1
latent_dim	1
hidden_units_g	100
hidden_units_d	100
batch_size	72
lim_max_1	1
lim_min_1	1
D_rounds	1
G_rounds	3
learning_rate	1e-5
decay_step	30000
decay_rate	0.98
num_epochs	1000

Tabla A.1: Tabla de parámetros donde a modo de ejemplo se muestran los valores para las primeras pruebas con el método con GAN.

Apéndice B

Herramientas para el aprendizaje profundo

B.1. Funciones de activación

Las funciones de activación son las que aparecen generalmente a la salida de cada neurona y se debe elegir una por cada capa de la red. Estas determinan si la neurona está activa o no en el caso binario o dan el valor de la potencia de activación en otros casos. Generalmente estás funciones son no lineales, porque de lo contrario la salida será una combinación lineal de la entrada. A continuación se describen aquellas funciones que son utilizadas en este documento:

Tangente hiperbólica

Es una función continua, que para un vector de entrada devuelve un vector del mismo tamaño.

$$\tanh(\mathbf{x}) = \frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}}$$

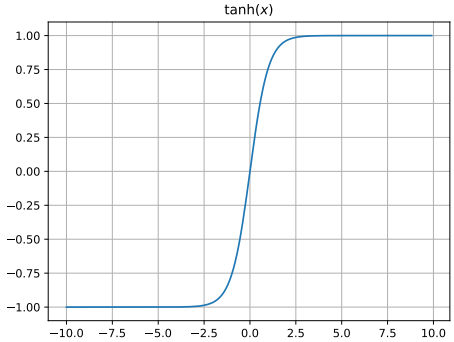


Figura B.1: Tangente hiperbólica de $\boldsymbol{x}_i$.

Softmax

Esta función está asociada a capas de salida en problema de predicción ya que devuelve solamente valores entre $[0, 1]$ y la suma de todos los elementos de la salida suman 1.

$$softmax(\mathbf{x}) = \frac{e^{\mathbf{x}_i}}{\sum_j \mathbf{x}_j}$$

Referencias

[1] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1):1–18, 2015.

[2] José Camacho, José Manuel García-Giménez, Noemí Marta Fuentes-García, and Gabriel Maciá-Fernández. Multivariate big data analysis for intrusion detection: 5 steps from the haystack to the needle. *Computers & Security*, 87:101603, 2019.

[3] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *arXiv preprint arXiv:1901.03407*, 2019.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[5] Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelfi. A survey on gans for anomaly detection. *arXiv preprint arXiv:1906.11632*, 2019.

[6] Georgios Douzas and Fernando Bacao. Effective data generation for imbalanced learning using conditional generative adversarial networks. *Expert Systems with applications*, 91:464–471, 2018.

[7] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer, 2002.

[8] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[9] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. A dataset to support research in the design of secure water treatment systems. In *International Conference on Critical Information Infrastructures Security*, pages 88–99. Springer, 2016.

[10] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

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

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[13] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *Advances in neural information processing systems*, pages 513–520, 2007.

[14] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Mahesh V Joshi, Ramesh C Agarwal, and Vipin Kumar. Mining needle in a haystack: classifying rare classes via two-phase rule induction. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 91–102, 2001.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[19] Kingsly Leung and Christopher Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, pages 333–342, 2005.

[20] Dan Li, Dacheng Chen, Jonathan Goh, and See-Kiong Ng. Anomaly detection with generative adversarial networks for multivariate time series. *arXiv preprint arXiv:1809.04758*, 2018.

[21] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. Ugr '16: A new dataset for the evaluation of cyclostationarity-based network idss. *Computers & Security*, 73:411–424, 2018.

[22] 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.

[23] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89, pages 89–94. Presses universitaires de Louvain, 2015.

[24] Sergio Martínez Tagliafico, Gastón García González, Alicia Fernández, Gabriel Gómez, and José Acuña. Real time anomaly detection in network traffic time series. In *Proceedings ITISE 2018: International conference on Time Series and Forecasting, Granada, Spain, 19-21 sep*, pages 1417–1428, 2018.

[25] A. P. Mathur and N. O. Tippenhauer. Swat: a water treatment testbed for research and training on ics security. In *2016 International Workshop on Cyber-physical Systems for Smart Water Networks (CySWater)*, pages 31–36, 2016.

[26] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed. Deepant: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access*, 7:1991–2005, 2019.

[27] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[28] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *ICISSP*, pages 108–116, 2018.

[29] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, et al. Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications. In *Proceedings of the 2018 World Wide Web Conference*, pages 187–196, 2018.

[30] Sultan Zavrak and Murat İskefiyeli. Anomaly-based intrusion detection from network flow features using variational autoencoder. *IEEE Access*, 8:108346–108358, 2020.

Esta página ha sido intencionalmente dejada en blanco.

Índice de tablas

4.1. Variables que componen el conjunto Tel2020. 32

5.1. Parámetros para la creación de las muestras correspondiente al conjunto Tel2020. 37
5.2. Cantidad de parámetros a entrenar para cada modelo. 37
5.3. Cantidad de parámetros a entrenar para cada modelo, cuando se utilizan todas la variables del conjunto Tel2020. 45
5.4. Parámetros del preprocesamiento aplicado a los datos del conjunto SWaT. 51
5.5. Cantidad de datos para la evaluación. 52

A.1. Tabla de parámetros donde a modo de ejemplo se muestran los valores para las primeras pruebas con el método con GAN. 66

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

2.1. Ejemplos de una serie univariable (a) y otra multivariable (b) extraídos del conjunto Tel2020 (Sección 4.4.2). 4

4.1. Arquitectura del modelo GAN, donde se indica: el tipo, la entrada y la salida para cada capa del Generador (a) y el Discriminador (b). Ambos tiene la misma estructura, lo único que cambia son las dimensiones de las entradas y salidas. 28

4.2. Arquitectura del modelo VAE, donde se indica: el tipo, la entrada y la salida de cada capa del *Encoder* (a) y el *Decoder* (b). También se puede apreciar las conexiones entre capas, y la perfundida a la que se encuentra cada una. 29

5.1. La figura (a) permite observar el comportamiento de la variable *callCost* para aproximadamente 20 días donde se puede apreciar una diferencia en la forma entre los días laborales y los no laborales. Las figuras (b) y (c) muestran un ejemplo de ambas formas. 36

5.2. La figura (a) permite observar la evolución de las funciones de pérdida a través de las épocas para el Generador y el Discriminador. Donde se puede apreciar la competencia entre ambas redes, ya que cuando una crece la otra decrece y viceversa. En la figura (b) se puede apreciar la evolución del valor de MMD a través de las épocas. En este caso, esta gráfica permite observar como rápidamente el modelo es capaz de generar muestras que se parezcan a las reales, ya que la gráfica muestra una caída abrupta hasta la época 200, para luego estabilizarse. 38

5.4. En la figura (a) se muestras tres series univariable, una es real y corresponde a datos utilizados en el entrenamiento de los modelos VAE y GAN, y las otras dos se construyeron a partir de muestras reconstruidas y generadas por estos modelos respectivamente. La figura (b) muestra el error cuadrático para cada instante entre la serie real y cada una de las series sintéticas. 41

5.5. Acercamiento del ejemplo de la figura 5.4 donde se puede apreciar mejor las diferencias entre las series sintéticas y la real. 42

5.6. En este ejemplo también se compara una serie real con las series sintéticas creadas a partir de las muestras de ambos modelos, con la diferencia con respecto al ejemplo de la figura 5.4 que en este caso la serie real no fue utilizada en el entrenamiento de los modelos. En la figura (b) se puede apreciar que el error es mayor en comparación con el ejemplo anterior. 43

5.7. En esta comparación también se utilizó una serie nunca antes vista por los modelos, pero estos fueron entrenados con un conjunto de datos mucho mayor en comparación con los ejemplos anteriores. Como se puede ver, se redujo el error con respecto a los días laborales si se compara con la figura 5.6, pero el error en los días no laborales es mayor. 44

5.8. En la figura (a) se puede ver un ejemplo de una serie multivariable, que corresponde a utilzar todas la variables del conjunto Tel2020. Las figuras (b) y (c) muestran como queda esta misma serie según los modelos de GAN y VAE respectivamente. Comparando ambas series sintéticas se puede apreciar como la construida a partir de muestras generadas por el modelo GAN, captura muy bien la varianza de todas las variables. Donde por otro lado la generada a partir de muestras reconstruidas, se muestra mucho más "suave". 46

5.9. En este ejemplo se busca mostrar de forma gráfica, cómo es la detección a partir del residuo tanto para el método con VAE como para el método con GAN. En la figura (a) se compara una muestra real y una generada por uno de los dos modelos. En la figura (b) se ve el error cuadrático entre ambas para cada instante, junto con los parámetros calculados en la fase de validación: el error medio μ_{error} y diferentes umbrales a partir de sumar $K\sigma$. 48

5.10. A partir de una muestra normal (a), se puede apreciar la salida del Discriminador (b). Como se puede ver la salida es cercana a 1 para todos los instantes lo que era de esperar ya que la muestra no presenta anomalías. 49

5.11. Forzando una anomalía colectiva en todas las variables a la vez sobre la muestra del ejemplo anterior como se muestra en (a), se puede apreciar en (b) como responde el Discriminador a la misma. Se puede ver que durante el intervalo en el que se produce la anomalía, la predicción devuelve valores bajos al rededor de 0.4 para todas las variables. 49

5.12. Esta vez se forzó a que la muestra presente una anomalía puntual y de contexto en todas las variables a la vez como se ve en (a). Se puede ver en (b) que la salida del Discriminador responde con un valor bajo en el instante que se produce la anomalía, con respecto a los demás valores que componen la salida. 50

5.13. En este ejemplo se buscó observar la respuesta del Discriminador a anomalía que solo se presentan en alguna de las variables de la series. Forzando tres anomalías puntuales distintas en las variables: azul, naranja, y verde, se puede ver en (b) que el discriminador reacciona para una sola de ellas, y en este caso el valor no es tan pequeño como en las anomalías de los ejemplos anteriores. 50
5.14. Evolución de las funciones de pérdida (a) y de la función MMD (b) a través de las épocas del entrenamiento del método con GAN con los datos del conjunto SWaT. Sobre todo en (b) se puede ver que el modelo logra ajustarse a la distribución real de los muestras observadas. 52
5.25. Ejemplo de una muestra real (a) junto con sus respectivas etiquetas (b), las cuales muestran dos anomalías. Junto a éstas se puede ver la reconstrucción a partir del modelo VAE (c), y el error cuadrático (d) entre ésta y la muestra real. Si se comparan (b) y (d), se puede ver que en este caso se podría detectar ambas anomalías, pero también algunas falsas alarmas. 59

Esta es la última página.
Compilado el viernes 25 septiembre, 2020.
http://iie.fing.edu.uy/