Escola Tècnica Superior d'Enginyeria ETSE-UV

MÁSTER UNIVERSITARIO EN CIENCIA DE DATOS

TRABAJO DE FIN DE MÁSTER

UTILIZACIÓN DE MODELOS GENERATIVOS PARA PROVEER DE DATOS ADICIONALES A MODELOS DE APRENDIZAJE SUPERVISADOS.

AUTOR:
EDUARDO DÁVILA GRAU

TUTORES:
VALERO LAPARRA, JORDI MUÑOZ

03/07/2020

Escola Tècnica Superior d'Enginyeria ETSE-UV

Máster Universitario en Ciencia de Datos

Trabajo de Fin de Máster

Utilización de modelos generativos para proveer de datos adicionales a modelos de aprendizaje supervisados.

Autor:
Eduardo Dávila Grau

Tutores:
Valero Laparra, Jordi Muñoz

Tribunal:

Presidente/a:

Vocal 1:

Vocal 2:

fecha de defensa:
calificación:

Resumen:

El campo de la inteligencia artificial ha experimentado un gran crecimiento en los últimos años. Esto se debe, en parte, a la gran cantidad de datos que se han ido almacenando por parte de empresas y otros organismos. Sin embargo, la cantidad de datos etiquetados, que son los susceptibles de ser utilizados en modelos de predicción, es mucho menor.

Por ello, este proyecto analiza el impacto de utilizar muestras sintéticas, creadas por redes neuronales generativas, a la hora de alimentar modelos de predicción, donde el número de muestras etiquetadas disponibles sea bajo.

Para ello, se utilizan dos modelos de red generativa, uno basado en la red generativa adversaria (o GAN), y otro llamado Random Rotation Based Iterative Gaussianization (o RBIG). Con estos modelos se generan datos sintéticos que pretenden asemejarse a los reales.

Los datos utilizados proceden del satélite MetOP de la organización 'European Organisation for the Exploitation of Meteorological Satellites' (EUMETSAT). Este satélite forma grupo con otros dos, cuyo objetivo es realizar órbitas polares alrededor de la Tierra. Sus mediciones son utilizadas para la predicción climática.

Uno de los instrumentos a bordo del satélite, conocido como IASI, se encarga de medir la radiación reflejada por la Tierra. De estas mediciones se obtienen una gran cantidad de datos, divididos en múltiples bandas. La alta dimensionalidad de estos datos requiere una compresión posterior, en este caso llevada a cabo mediante la técnica PCA. Adicionalmente, se ha utilizado un modelo físico de predicción de temperatura, en base a dichos datos, desarrollado por el grupo European Center for Medium-Range Weather Forecast. La combinación de los datos de IASI junto con el modelo del ECMWF componen los datos utilizados en el proyecto.

Tras la generación de muestras con los modelos GAN y RBIG, se alimenta a los modelos de regresión de temperatura con diferentes combinaciones de muestras reales y sintéticas. El análisis de los resultados obtenidos lleva a dos conclusiones principales:

En primer lugar, que la adición de muestras sintéticas a los modelos de regresión resulta muy conveniente siempre que el número de muestras reales disponibles sea relativamente bajo. Por el contrario, cuando el número de las mismas es alto, añadir estos sintéticos no resulta beneficioso. En segundo lugar, se demuestra que el modelo RBIG es más eficiente a la hora de mejorar el error de predicción, siempre teniendo en cuenta la premisa anterior.

Abstract

Artificial Intelligence has experienced major growth in recent years, mainly because the enormous volume of data stored by companies and other organizations. Despite that, the amount of labeled data, which is susceptible to be used in prediction models, is not that big.

Thus, this project analizes the impact of synthetic samples creation and usage so as to feed predition models, especially when the labeled data amount is small.

For that purpose, two generative models have been used. First, a Generative Adversarial Network, or GAN, and second, a model that is based on Random Rotation Iterative Gaussianization or RBIG. These models are used to generate synthetic data, that should look like real data.

These real data come from MetOP, a satellite from the European Organization for the Exploitation of Meteorological Satellites or EUMETSAT. MetOP is part of a group with two additional satellites, orbiting around the Earth, which measurements are very important in climatic predictions. An instrument aboard MetOP, known as IASI, is used to measure the radiancy reflected by Earth. These measurements provide a big amount of useful data, dividided in several bands. Because of that high dimensionality in the data, a technique named PCA is used. On the other hand, the European Center for Medium-Range Wather Forecast provided a physical model in order to get the corresponding atmosfere temperature values. These combined data are used to feed the generative models, RBIG and GAN, in the project.

After the generation of synthetic samples, the regression model is fed with different combinations of real and synthetic data. The analysis of results leads to two conclusions: First, it has been proven that using synthetic data has major benefits only if the amount of real labeled data is low. On the other side, if the amount of real data is big, the usage of synthetic data would no longer be useful. Second, RBIG model has proven to be more effective than GAN, provided that the mentioned above is true.

Índice

Índice de figuras

1. Motivación e introducción

Los últimos años han presenciado un importante aumento de las contribuciones científicas en el campo de la Inteligencia Artificial. Especialmente destacable es el campo del aprendizaje máquina que ha crecido de forma notable, sobre todo a partir de la utilización de técnicas de Deep Learning. En la figura 1 se observa la composición de estos tres campos.

Una gran cantidad de algoritmos y arquitecturas de redes se han desarrollado teórica y prácticamente en ese tiempo, incluidas las redes neuronales convolucionales, redes recurrentes, o aproximaciones generativas como la Red Generativa Adversaria (GAN). Las aplicaciones de estas redes han tenido un gran impacto en campos del Deep Learning como el procesado del lenguaje natural, la visión por computador o el reconocimiento de voz.

Estas investigaciones se han convertido en el estado del arte actual de dichos campos, donde se utilizan datos con estructura espacial y/o temporal en la mayoría de los casos. Así mismo, el campo de investigación sobre la Tierra ha crecido notablemente en la última década, poniendo el foco en problemas de clasificación de imágenes y detección de objetos de imágenes por satélite, entre otros

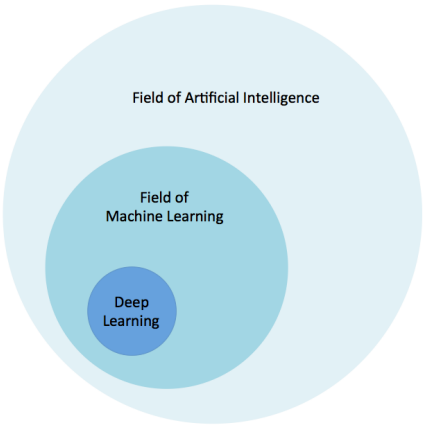

Figura 1: Jerarquía de la inteligencia artificial.

Gran parte de estos modelos de Deep Learning, sin embargo, requieren de una gran cantidad de datos etiquetados para poder realizar de forma correcta el aprendizaje de la red. Este hecho provoca que, a pesar de que hoy en día ya existen métodos muy avanzados de resolución de problemas, ciertos proyectos no pueden llevarse a cabo por la falta de una cantidad suficiente de dichos datos, ya que el proceso de etiquetado requiere en muchas ocasiones del intervención humana directa.

Para lidiar con este tipo de problemáticas, el uso de los enfoques generativos ha resultado de gran ayuda. Estas redes generativas se encargan de generar muestras sintéticas, con suficiente parecido a las originales como para complementarlas correctamente.

Con todo esto, se ha realizado un análisis de la utilidad de utilizar muestras sintéticas para alimentar el entrenamiento de modelos predictivos, donde el número de muestras reales etiquetadas disponibles es bajo.

Para llevar a cabo el análisis se han utilizado dos modelos generativos, uno basado en la competición entre dos redes (GAN), y otro basado en la transformación en Gaussiana de una PDF desconocida,mediante la aplicación de una marginalización sobre cada dimensión de los datos y una rotación sobre el conjunto de los mismos (RBIG).

Estos modelos generan nuevas muestras que complementarán los conjuntos de datos originales. Posteriormente, serán evaluados por una red neuronal que calcula la temperatura de la superfície terrestre. En la figura 2 se observa de forma esquemática la propuesta desarrollada para este problema.

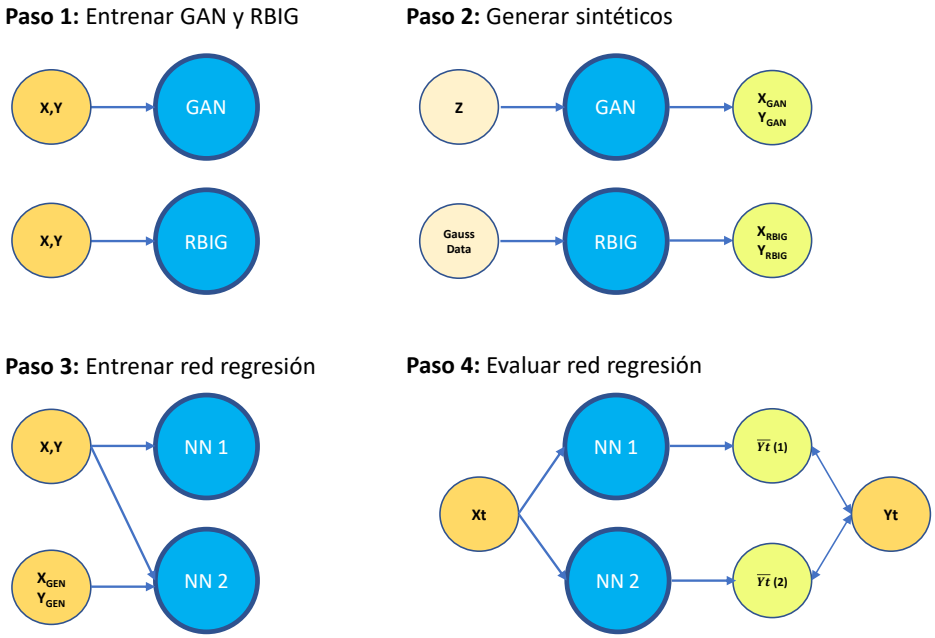

Figura 2: Esquema de la tubería propuesta para este experimento

Durante el desarrollo del proyecto, se han establecido los siguientes objetivos:

- Comprobar la eficacia del modelo de regresión sobre los datos de partida, sin incluir sintéticos. Esto permitirá comprobar la evolución del error de la red neuronal según se le añaden muestras reales.
- Adición de datos sintéticos de forma gradual al modelo anterior, para comprobar si existen variaciones en cuanto a error, tiempo, etc.
- Averiguar el número óptimo de muestras sintéticas que se deben generar para obtener el menor error en la predicción de la temperatura.
- Comparación entre ambos modelos generadores para determinar cual reslta más apropiado para este problema.

2. Estado del arte y procedencia de los datos

El campo relacionado con los modelos generativos ha ganado repercusión en la última década, debido a sus múltiples usos dentro del Deep Learning. Sin embargo, estos modelos se basan en evoluciones de las redes neuronales clásicas, desarrolladas ya por los años 60, con el objetivo de abordar problemas complejos.

Entre los problemas que las redes neuronales resuelven, los relacionados con la predicción han sido los más utilizados durante muchos años. Por ejemplo, en problemas donde se desea obtener un valor continuo como respuesta de la red, como la regresión, el sistema

trata de aprender un predictor en base a unas variables, o lo que es lo mismo, capturar la probabilidad condicional $P(Y \mid X)$.

En el caso del enfoque generativo, se incluye la distribución que siguen los datos, y se intenta predecir la probabilidad de que cierta muestra pertenezca a una clase dada, es decir, capturar la distribución de probabilidad conjunta a todas las variables, $P(X, Y)$, o $P(X)$, si no existen muestras etiquetadas.

A continuación, se muestra información relevante al modelo básico de red neuronal, desarrollado en los años 60, sobre los que evolucionaron gran parte de los modelos posteriores. Seguidamente, se explica el funcionamiento de las dos redes generadoras utilizadas en el proyecto, GAN y RBIG.

2.1. Multilayer Perceptron

El perceptrón multicapa, MLP en adelante, surge como evolución natural del perceptrón individual, desarrollado por Frank Rosenblatt en 1957. Su funcionamiento se basa en la utilización de dichos perceptrones para transmitir información, modificada por diferentes funciones matemáticas aplicadas a la misma. La utilización de grupos de estas unidades forma lo que se conoce como una red, que de organizarse en capas interconectadas, forma el MLP nombrado anteriormente. [1]

En la figura 3 se encuentra una representación del modelo MLP.

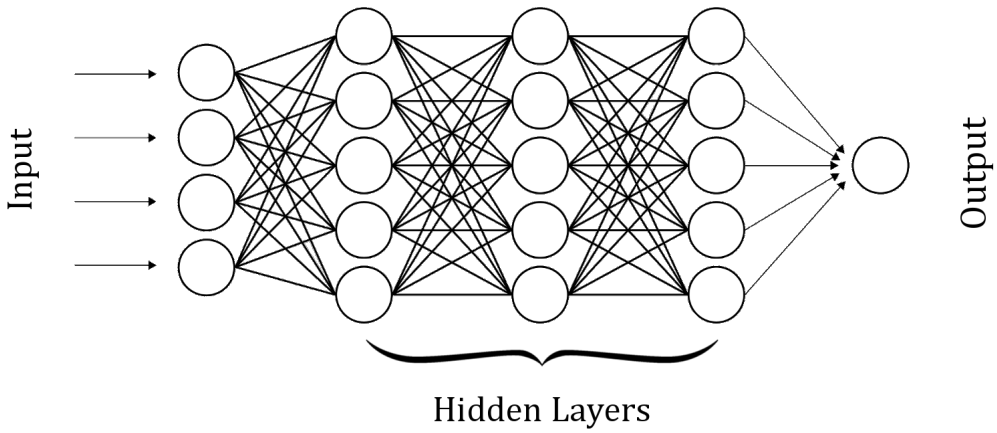

Figura 3: Representación del MLP.

El funcionamiento de estas capas es el siguiente:

- Input layer o capa de entrada: Corresponde a las entradas del modelo, es decir, los datos sobre los que se trabaja.
- Hidden layers o capas ocultas: Corresponde a un conjunto de neuronas interconectadas que van procesando los datos.
- Output layer o capa de salida: Corresponde al resultado final generado por la red.

2.1.1. Funcionamiento de la neurona

El funcionamiento de la red MLP se puede explicar a partir de una unidad neuronal individual, debido a que la red se compone de múltiples de estas unidades interconectadas. Cada una de estas neuronas se compone de los siguientes elementos:

- Entradas de la neurona, (x1,x2...xn).
- Pesos de la neurona, (w1,w2...wn), que representan los coeficientes aplicados a cada entrada.
- Bias, un factor constante aditivo.
- Función de activación, es una función no lineal unidimensional normalmente de un carácter monotónico creciente.

Así pues, cada uno de las entradas debe ser multiplicada por su correspondiente peso, obteniendo así una suma ponderada a la que se le añade el bias, de forma análoga a una función lineal clásica: $y = mx + b$.

El resultado de esta operación genera un resultado lineal, que posteriormente se pasa por una función de activación que permita a la red neuronal captar no linealidades. En la figura 4 se observa el funcionamiento de una de estas neuronas.

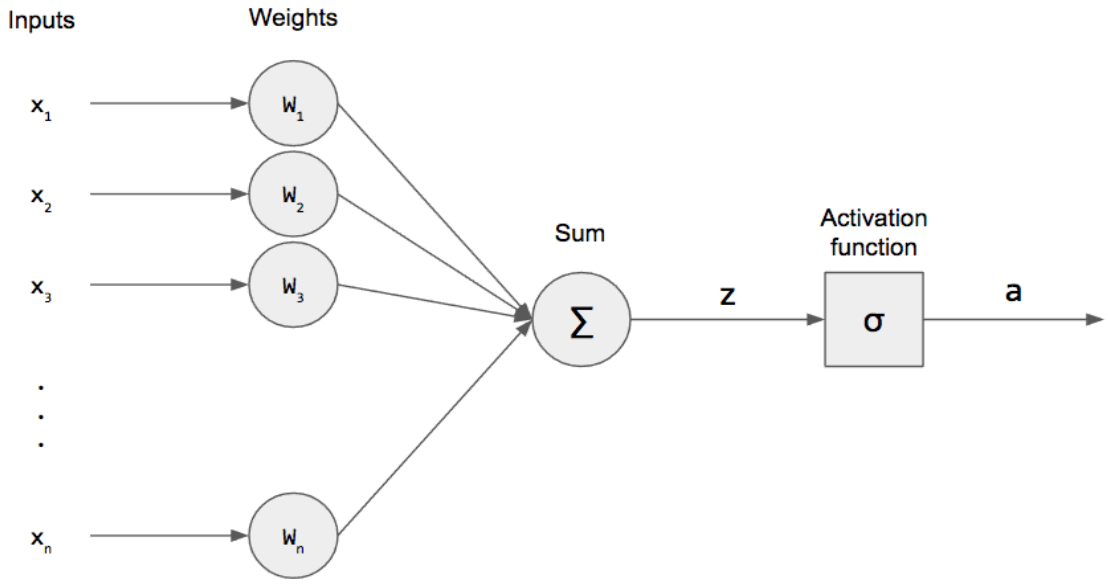

Figura 4: Esquema de un MPL.

2.1.2. Funciones de activación

Las funciones de activación son aquellas que permiten a la red que está siendo entrenada captar no linealidades en los datos. Existen diferentes tipos de estas funciones, pero a continuación se nombrarán las más comunes [2].:

- Sigmoid, que genera valores entre 0 y 1, por lo que es ideal para obtener probabilidades y responde a la fórmula

$$\frac{1}{1 + e^{-x}}$$

.

- Tangente hiperbólica, que genera valores entre -1 y 1, por lo que es simétrica y es adecuada para las capas ocultas. Responde a la fórmula

$$\frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1$$

.

- ReLU, o Rectified Linear Unit, es una función que genera un 0 si el valor del input es menor que 0, y el propio valor si es mayor. Esta función soluciona problemas de convergencia de las anteriores funciones,y además, es una función no lineal, monótona y requiere una baja carga computacional. Por ello, es una de las mas utilizadas, y se corresponde con la fórmula siguiente:

$$relu(x) = max(0, x)$$

Estas funciones de activación pueden cambiar entre capas, y su aplicación permite la captura de no linealidades que puedan existir en los datos. La figura 5 muestra la forma de las tres funciones mencionadas anteriormente.

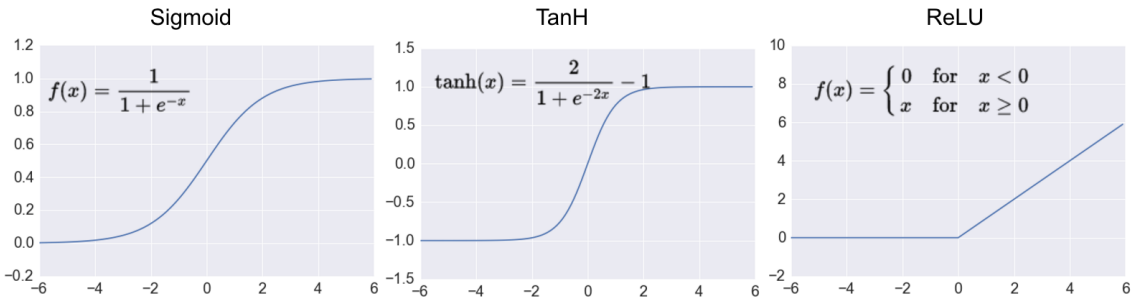

Figura 5: Funciones de activación.

2.1.3. Aprendizaje de la red

En lo referente al aprendizaje de una red neuronal, se realiza un proceso iterativo mediante el cual la red es capaz de aprender a generar valores de salida lo más cercanos posible a los esperados.

Para ello, se suele dividir el conjunto de datos de entrada en fragmentos llamados lotes, lo que evita el problema de cargar en memoria todo el conjunto de datos de forma simultánea. El entrenamiento de la red se realiza pues, para cada uno de dichos lotes. Cuando todos los lotes han sido entrenados se considera que ha sido entrenada una época, y el entrenamiento de la red suele requerir de varias épocas para su correcto funcionamiento.

Este proceso de entrenamiento se puede explicar, de forma esquemática, en los siguientes pasos:

- Inicializar los pesos de la red, ya sea de forma aleatoria o mediante algún análisis previo.

- Forward propagation: Consiste en, a partir de las entradas, ir calculando los valores de salida de cada neurona mediante el proceso comentado en la sección Funcionamiento de la neurona.

- Una vez alcanzadas las neuronas de salida, se deben comparar las predicciones obtenidas con los valores esperados, utilizando para ello una función de coste o error.

- Propagación hacia atrás: En este paso se actualizan los pesos de cada una de las neuronas según su participación en el error calculado anteriormente. De esta forma, en la siguiente iteración la red habrá aprendido de sus errores anteriores, y minimizará dicho error a la hora de calcular futuras predicciones.

2.1.4. Función de coste

Por último, es necesario recalcar el papel de la función de coste en el proceso de entrenamiento. Tras realizar la propagación de pesos hacia adelante en el entrenamiento, se compara el valor generado con el valor esperado mediante una función. Esta función depende del tipo de valor que se haya generado, pero basándonos en los principales tipos de ellas, las funciones de coste más utilizadas son [3]:

- Regresión: Mean Squared Error, o error cuadrático medio.

$$MSE = (\frac{1}{n}) \sum_{i=1}^{n} (y_i - x_i)^2$$

- Clasificación: Cross Entropy. En su forma general:

$$CrossEntropy = -\sum_{i=1}^{n} (y_{o,c} log(p_{o,c}))$$

  Donde N es el número de clases, Y es el indicador binario de si la clase con etiqueta 'c' es una clasificación correcta de la observación 'o', y P representa la probabilidad predicha de que la observación 'o' sea de clase 'c'

Estas funciones medirán la progresión de la red que se está entrenando y, a su vez, servirán como objetivo a minimizar para siguientes iteraciones de la red. De esta forma, a la hora de realizar la propagación hacia atrás, se realizará intentando encontrar la combinación de pesos de las neuronas que minimice dicha función de coste.

Existen distintos métodos disponibles a la hora de realizar el proceso de minimización, pero el descenso por gradiente es el más utilizado.

2.1.5. Descenso por gradiente

El método de descenso por gradiente es uno de los algoritmos de optimización más populares dentro del aprendizaje automático, principalmente por su uso dentro del campo de las redes neuronales. Es un método que trata de minimizar una función 'f'. Para conseguir encontrar el mínimo de una función, existen diferentes opciones:

- Forma analítica: Calculando la derivada de la función y encontrando los puntos donde dicha derivada es 0.

- Forma numérica: Inicializando en un punto de la función y tratando de descender hasta un mínimo utilizando la primera derivada.

- Utilizando métodos aproximados, como BFGS, PSO.

Entre estos métodos, el más interesante es el que hace referencia a una forma numérica, y donde se sitúa el descenso por gradiente.

En este método, se intenta alcanzar un valor mínimo de la función, partiendo desde un punto inicial. Para ello, se hace uso de la primera derivada en el punto específico. Esto permite averiguar el crecimiento de dicha función en ese punto. Así pues, para encontrar un mínimo, será necesario seguir la dirección opuesta a la pendiente.

En la figura 6 se puede observar una representación gráfica del descenso por gradiente. Este hecho hace referencia al apartado relativo al descenso, a lo que se añade la parte correspondiente al gradiente.

El gradiente es una generalización vectorial de la derivada, es decir, un vector de tantas dimensiones como tenga la función, donde cada una de estas contiene la derivada parcial en dicha dimensión, como podemos observar en la siguiente expresión:

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots \frac{\partial f}{\partial x_n}\right]$$

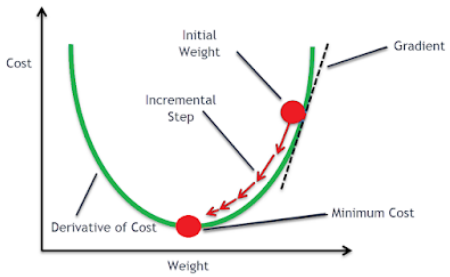

Figura 6: Algoritmo de descenso por gradiente.

Por tanto, el gradiente será el vector que indica cuanto crece la función en un punto concreto, por cada una de las dimensiones de dicha función de forma independiente.

Una vez conocida la dirección en la que se quiere avanzar para descender, es notable destacar cuánto se desea descender. Para controlar esta cantidad, se utiliza un parámetro conocido como learning rate, que permitirá ajustar la cantidad que se avanza en la dirección deseada [4]

Existen tres variantes principales del método del descenso por gradiente, que difieren entre ellas en la cantidad de datos que se utilizan para computar el gradiente de la función objetivo.

- Descenso por gradiente original: Utiliza todo el conjunto de datos para el cálculo del gradiente. Esto puede provocar que el entrenamiento sea muy lento y, además, puede llevar a ser intratable si el tamaño de los datos es lo suficientemente grande como para no caber en la memoria.

  Por otra parte, al actualizar los parámetros de la red en que se utilice el descenso por gradiente, en la dirección opuesta al gradiente, garantiza la convergencia en un mínimo. Este mínimo será global si la superficie de la función es convexa, y local si es no convexa.

- Descenso por gradiente estocástico (SGD): Esta variante se centra en el punto contrario al descenso por gradiente original. En este caso, se utiliza cada muestra individual del banco de datos para la actualización de parámetros de la red.

  Las ventajas de este método se encuentran en su velocidad, que es mucho más rápida en cada iteración. Sin embargo, produce una convergencia menor que en el caso anterior.

- Descenso por gradiente en mini lotes: Esta variante realiza una mezcla entre las dos anteriores. En cada iteración, el algoritmo realiza los cálculos en un conjunto

de datos del banco de datos original. De esta forma, se logra un equilibrio entre llegar a una convergencia aceptable, y un tiempo de entrenamiento no demasiado excesivo.

Por tanto,y como conclusión, el método del descenso por gradiente se ha postulado como una de las opciones mas utilizadas para el entrenamiento de redes neuronales.

En las siguientes subsecciones se comentarán los modelos de GAN y RBIG sobre los que se estructura el presente proyecto.

2.2. Generative Adversarial Networks (GAN)

Entre los modelos generativos, la red generativa adversaria ha tenido un alto impacto en tareas relacionadas con el tratamiento de imágenes. Propuesta por primera vez en 2014, en el artículo llamado Generative Adversarial Networks [5], se basa en la competición entre dos redes neuronales para producir datos del conjunto con el que se le entrena. Estas redes se conocen como el generador y el discriminador. La primera corresponde a una red que se encarga de generar datos del tipo elegido, mientras que la segunda se encarga de distinguir entre datos generados y reales. Estas dos redes poseen sus propias funciones de coste, por lo que a diferencia de otras redes más clásicas, el objetivo es encontrar un punto de equilibrio entre la del generador y la del discriminador.

2.2.1. Estructura del modelo GAN

La estructura de una red GAN está formada por una función diferenciable, como una red neuronal, que actúa como generador de datos, y otra que actúa como discriminador de los mismos. Las funciones principales de estas redes son las siguientes:

- Generador: Encargado de generar un dato a partir de un vector de ruido aleatorio z. De esta forma, la red genera muestras completamente aleatorias al principio, ya que se basa en las calificaciones del discriminador para mejorar sus creaciones.

- Discriminador: Su objetivo consiste en discernir entre datos del banco de datos real y aquellos que han sido creados por la red generadora, devolviendo la probabilidad de que el input sea real, con valores entre 0 y 1. Esta red se entrena con datos reales y datos generados, para poder diferenciar entre ellos.

Esta última red será entrenada, por tanto, para maximizar la probabilidad de asignar etiquetas correctas, tanto a muestras reales como a las creadas por el generador. El generador, por su parte, será entrenado simultáneamente para minimizar $log(1 - D(G(z)))$, es decir, para engañar al discriminador y que este etiquete sus muestras como reales.

En la figura 7 se puede observar un esquema básico de la estructura de una GAN.

Cabe remarcar que la GAN solo será capaz de generar datos similares a aquellos con los que sea entrenada. Es decir, el objetivo del Generador será el de producir muestras sintéticas que capturen la distribución de los datos del banco de datos real [5].

2.2.2. Funciones de coste

En este apartado se comentarán en más detalle las funciones de coste utilizadas por ambas redes de la GAN. Se denomina $J^{(G)}$ a la función de coste del Generador, mientras

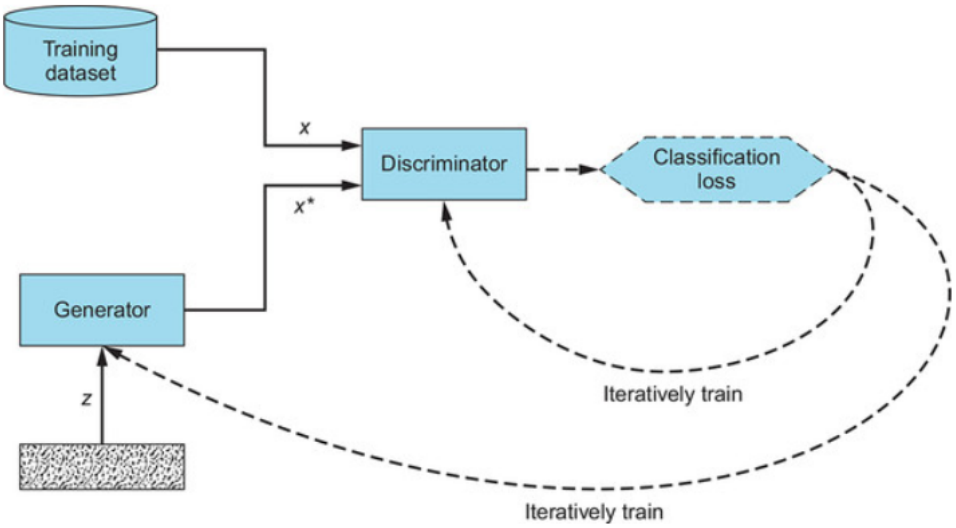

Figura 7: Estructura básica de una GAN. Obtenida de: [6]

que $J^{(D)}$ será la función de coste del Discriminador. De igual manera, $\theta^{(G)}$ y $\theta^{(D)}$ representarán los parámetros de ambas redes, correspondientes a los pesos y bias.

Por tanto, la función de coste de la GAN se plantearía como un juego minimax entre las dos funciones de coste, de las redes G y D, con la siguiente función V(G,D):

$$min_G max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim Pdata(\mathbf{x})}[logD(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim Pz(\mathbf{z})}[log(1 - D(G(\mathbf{z})))]$$

Existen dos aspectos que hacen que las funciones de coste de la GAN sean diferentes a las de una red neuronal convencional. En primer lugar, el entrenamiento de ambas redes se realiza con la función anterior, [6].

En segundo lugar, y relacionado con el punto anterior, cada red solo es capaz de realizar el ajuste de sus propios parámetros. De esta forma, el generador, a pesar de tener pesos y bias tanto propios como del discriminador, solo será capaz de modificar los correspondientes al generador. Así, cada red tiene una parte de control a la hora de determinar la función de coste.

2.2.3. Entrenamiento de la red

El proceso de entrenamiento de la GAN es diferente al que teníamos en el caso de una red neuronal básica. En estos, el entrenamiento se basaba en un problema de optimización, mientras que en este caso ambas redes compiten entre si para minimizar o maximizar la función de coste del discriminador.

El objetivo pues será encontrar un punto de equilibrio "Nash equilibrium", es decir, un punto en el que ninguno de los dos componentes, generador y discriminador, pueda mejorar su situación cambiando de estrategia, o parámetros en este caso. Esto ocurre cuando $J^{(G)}(\theta^{(G)}, \theta^{(D)})$ se minimiza con respecto a los parámetros que se pueden entrenar del generador $J^{(D)}$ y, simultáneamente, $J^{(D)}(\theta^{(G)}, \theta^{(D)})$ está minimizado con respecto a los parámetros del discriminador $J^{(D)}$ [6].

En la figura 8 podemos observar el funcionamiento de un juego de dos jugadores basado en 'zero-sum game' y el proceso de alcanzar el 'Nash equilibrium'. Este tipo de juego es en el que está basado el proceso de entrenamiento de la red GAN. En ella, podemos

observar como el jugador 1, que representaría al discriminador, trata de minimizar la función de coste bajo los parámetros de θ_1, mientras que el jugador 2, que representaría al generador, intenta maximizar dicha función de coste con sus parámetros θ_2. En la figura de la derecha se observa el proceso de alcanzar el punto de equilibro en el espacio en 3D que representan las anteriores figuras.

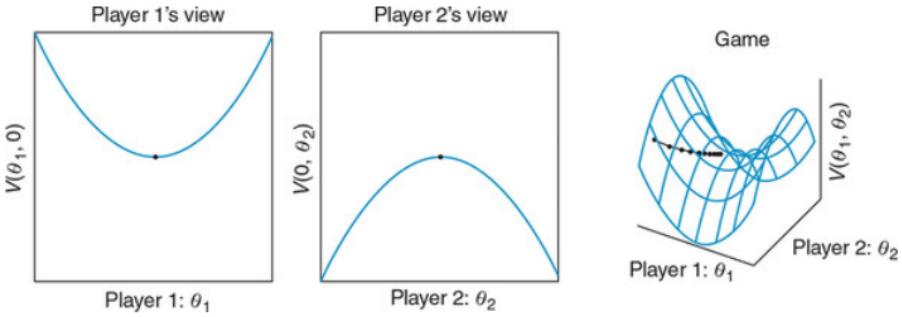

Figura 8: Zero-sum game,
Fuente: Goodfellow, 2019, www.iangoodfellow.com/slides/2019-05-07.pdf

A pesar de esto, la búsqueda del punto de equilibrio entre las dos redes resulta casi imposible en un escenario real, debido a la gran complejidad de los datos que se pueden encontrar. Aún con esto, el objetivo será encontrar una buena aproximación a este punto, donde los datos generados sean capaces de engañar al discriminador un porcentaje de veces, pero que este discriminador, a su vez, sea capaz de distinguir un buen porcentaje de veces entre datos reales y generados.

Por tanto, los pasos que el entrenamiento de la GAN debería seguir son los siguientes [7]:

- Entrenar el discriminador:
  - Elegir un conjunto de ejemplos reales X.
  - Elegir un conjunto de ruido aleatorio z y generar un conjunto de ejemplos falsos con el generador $G(z) = x^*$.
  - Calcular los errores para el discriminador con las muestras reales y las falsas $D(x) y D(X^*)$ y realizar propagación hacia atrás por la red actualizando sus parámetros $\theta^{(D)}$ con el objetivo de minimizar el error de clasificación/regresión.
- Entrenar el generador:
  - Elegir un conjunto de vector de ruido aleatorio z y generar muestras falsas $G(z) = x^*$.
  - Calcular los errores para el discriminador con dichas muestras falsas y actualizar mediante propagación hacia atrás los parámetros del generador $\theta^{(G)}$, con el objetivo de maximizar el error.

2.3. Modelo RBIG

El segundo modelo sobre el que se han realizado las pruebas es el conocido como RBIG (Rotation-based Iterative Gaussianization) [8]. Este modelo surge como una solución al problema de estimar funciones de densidad de probabilidad multidimensionales.

La problemática de los métodos clásicos de estima de densidades reside en que requieren una cantidad de muestras que escapa a las capacidades de que se dispone actualmente, debido principalmente al problema conocido como maldición de la dimensionalidad [9].

En la figura 9 se puede observar la evolución del número de muestras requeridas para la estimación de una PDF desconocida, según el número de dimensiones.

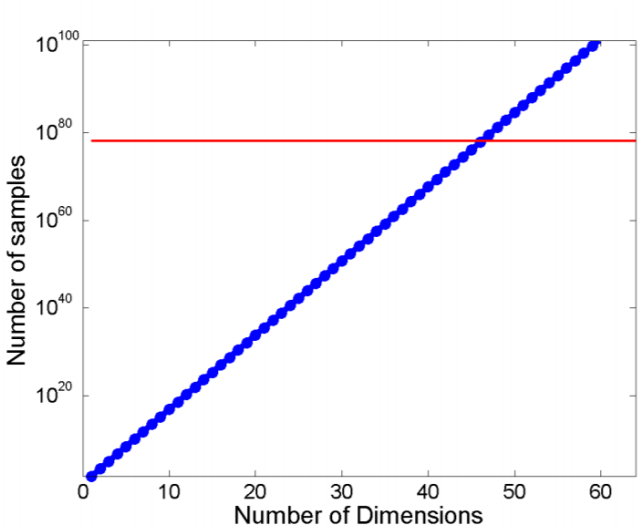

Figura 9: Muestras vs dimensiones.

Así, para estimar una PDF que tuviera alrededor de 50 dimensiones, algo muy común hoy en día, se requerirían alrededor de 10^{80} muestras, que equivale al número de átomos existentes en el universo, por lo que un nuevo enfoque se torna imprescindible para abordar esta problemática.

Como posible solución al problema,el modelo RBIG propone transformar la PDF desconocida en una que si resulte conocida, como puede ser el caso de la distribución Gaussiana.

Para ello, el método RBIG propone realizar una serie de transformaciones sencillas que vayan modificando la distribución para parecerse mas a una distribución Gaussiana con media cero y matriz de covarianza identidad.

El proceso iterativo se compone de dos elementos. En primer lugar, se realiza una gaussianización marginal de cada de las dimensiones, de forma independiente, y a continuación, se le aplica una rotación al conjunto.

La rotación óptima es aquella que intente buscar las direcciones menos Gaussianas del conjunto. De esta forma, la siguiente iteración intentará Gaussianizarlas. Sin embargo se demuestra que cualquier rotación hace converger al método [8]. Posibles rotaciones a usar son las siguientes:

- ICA, o Independent Component Analysis, que encuentra aquellas direcciones menos Gaussianas. Tiene como inconveniente su gran coste computacional, y por tanto, su lentitud.
- Con rotaciones aleatorias, que se demuestra la convergencia, con menor coste computacional pero en un número de iteraciones mucho más alto.
- PCA, o Principal Component Analysis, que se establece como una solución intermedia entre las dos anteriores, y es la que mejores resultados ofrece en la mayoría de casos.

Partiendo, por tanto, de una variable aleatoria multidimensional x^0, que sigue una PDF desconocida, en cada iteración k se realiza el siguiente proceso:

$$G : x^{k+1} = R_k \Psi_k(x^k)$$

Donde G es la transformación resultante, Ψ_k representa la Gaussianización marginal de cada dimensión de x^k, y R_k es una matriz de rotación.

Cabe destacar que en este proceso iterativo el peso más grande recae sobre la Gaussianización marginal, ya que la rotación tras cada iteración se puede realizar de forma aleatoria o con un PCA, y en cualquier caso, no tiene una influencia excesiva. Esta Gaussianización marginal, como ya se ha comentado, transforma la distribución de cada cada dimensión en Gaussiana, y para ello se realiza una ecualización de la distribución original para convertirla en uniforme, y la inversa de la PDF Gaussiana.

RBIG cuenta con una funcionalidad adicional, cada etapa añadida iterativamente al modelo es invertible. Este hecho es el que provoca que puedan generarse nuevas muestras similiares a las originales. Para ello, el modelo sigue los siguientes pasos:

- Se obtiene una transformación G mediante el aprendizaje de la red, comentada anteriormente, sobre los datos originales. Con ella tenemos la PDF desconocida transformada en Gaussiana.
- Se generan muestras aleatorias que sigan una distribución Gaussiana.
- Se aplica la transformación inversa G^{-1} a las muestra Gaussianas obteniendo muestra que siguen la distribución de los datos originales.

En la figura 10 se ilustra este procedimiento. Se puede observar como 3 grupos de datos, con PDF desconocidas, son transformadas en PDF Gaussianas. Una vez realizado esto, se reconstruyen los datos invirtiendo el proceso y obteniendo muestras de dicha PDF [8].

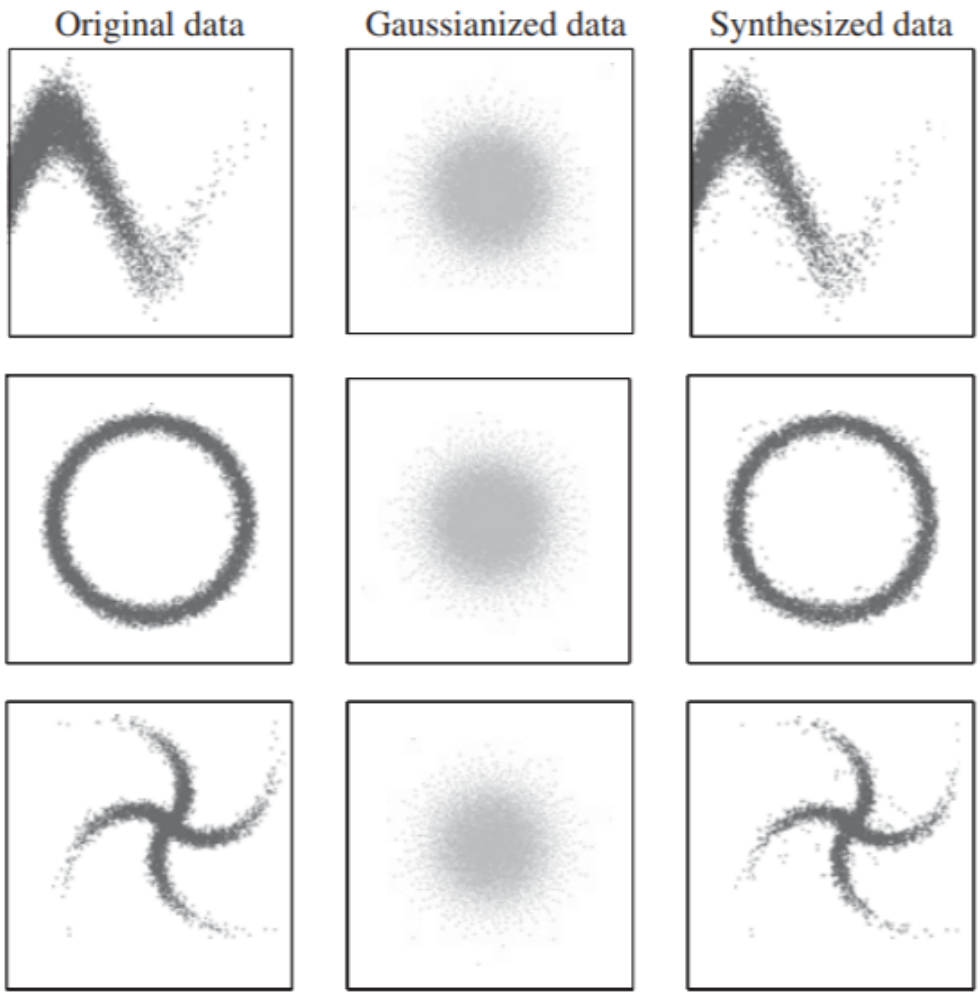

Figura 10: Generación de muestras con RBIG.

RBIG se postula como un método capaz de realizar la tarea de generación de datos, por lo que se ha escogido como candidato para este proyecto.

3. Descripción de los modelos y datos utilizados

3.1. Procedencia de los datos

El objetivo de este proyecto es comprobar si la generación de muestras sintéticas con los modelos GAN y RBIG resulta adecuada para un modelo de regresión de temperaturas.

Los datos utilizados para tal fin provienen de dos fuentes distintas. En primer lugar, el conjunto de datos originales se obtiene de un satélite de la Agencia Espacial de Meteorología (EUMETSAT). En segundo lugar, el European Center for Medium-Range Weather Forecasts proporciona un modelo físico de predicción de temperaturas en base a dichos datos. La combinación de estos dos elementos da lugar al banco de datos utilizado en el proyecto.

3.1.1. EUMETSAT, MetOP e IASI

Los datos de entrada originales provienen de un satélite de la Agencia Espacial de Meteorología (EUMETSAT en adelante). EUMETSAT es una organización intergubernamental cuyo objetivo es proveer de datos por satélite relacionados con condiciones climáticas. Estos datos son de gran importancia a la hora de realizar multitud de tareas diferentes, como por ejemplo la monitorización climática, avisos de posibles desastres climáticos, etc.

Para ello, EUMETSAT ha lanzado a la órbita terrestre multitud de satélites, entre los que podemos destacar, entre otros:

- METEOSAT: Provee de imágenes del disco de la tierra completo y es utilizado normalmente para pronósticos climáticos.
- MetOP: Un conjunto de 3 satélites en órbitas polares bajas, y proveen de información detallada sobre la atmósfera, océanos y continentes.

De estos, MetOP es el más interesante para este proyecto, y es el satélite que provee los datos objeto de este proyecto. MetOP cuenta con un total de tres satélites, que están ubicados en la órbita baja de la Tierra, a una altitud de 817 km, y que han sido lanzados de forma secuencial desde el año 2006, en el caso de Metop-A hasta 2018 en el caso de Metop-C. El objetivo de los mismos es obtener observaciones globales de la atmósfera, océanos y continentes. Cada nuevo satélite añadido al conjunto de MetOP ayuda a prevenir efectos climáticos adversos que podrían influir en las mediciones, como la existencia de nubes. [10].

Cada uno de estos satélites cuenta con una serie de instrumentos encargados de tomar diferentes medidas. Entre ellos encontramos el nombrado como Infrared Atmospheric Sounding Interferometer (IASI en adelante), de cuyos datos se nutre este proyecto.

IASI es un instrumento que mide la radiación reflejada por la Tierra, realizando barridos verticales de la misma. Para ello, IASI mide la parte infrarroja del espectro electromagnético emitido por la Tierra, con una resolución horizontal de 12 km y una anchura de 2200 km. Realiza un total de 14 órbitas y sus observaciones globales pueden ser obtenidas dos veces al día [11].

En la figura 11 se puede observar una representación de su funcionamiento.

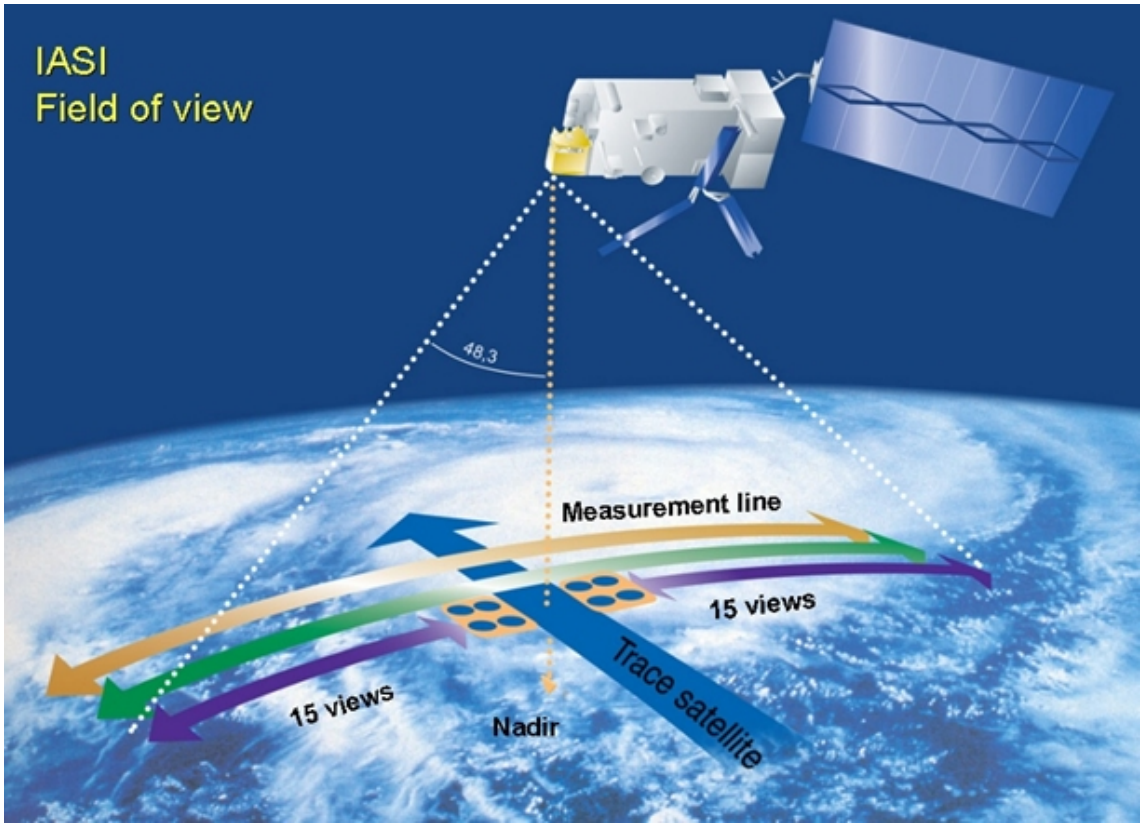

Figura 11: Instrumento IASI a bordo del satélite MetOP

3.1.2. PCA

Los datos de partida que se han utilizado en este proyecto provienen del instrumento IASI, a bordo del satélite MetOP, que mide el espectro de radiofrecuencias emitidas por la Tierra.

Los datos de MetOP tienen una gran cantidad de dimensiones, debido principalmente a que IASI es capaz de producir más de 6000 bandas frecuenciales por cada muestra.

Este alto número de dimensiones hace necesaria la aplicación de un tratamiento que reduzca dicha cantidad. Para ello, se utiliza una técnica conocida como PCA, o Principal Component Analysis.

El PCA es una técnica utilizada para describir un conjunto de datos con unas nuevas variables, que no estarán correlacionadas. Además, la técnica construye una rotación en la que se escoge un nuevo sistema de coordenadas para el conjunto de datos que se vaya a reducir. Esta rotación trata de que el nuevo sistema de coordenadas tenga la mayor varianza posible entre sus dimensiones. De esta forma, las primeras componentes principales describirán la mayor parte de la varianza de los datos, mientras que el resto, en la mayoría de casos, será posible descartarlas. [12].

Por tanto, valiéndose de esta técnica, se ha reducido el tamaño de los datos IASI de 8461 dimensiones a un total de 50 dimensiones, es decir, las 50 primeras componentes principales resultantes de aplicar un PCA a los datos.

3.1.3. ECMWF

Los datos de salida del proyecto se corresponden con la temperatura en la superficie de la Tierra, correspondiente a la posición de las radiancias medidas por IASI.

Para obtener esta medida, se recurrió a un modelo físico, desarrollado por el grupo 'European Center for Medium-Range Weather Forecasts' (ECMWF en adelante), que provee perfiles de temperatura y humedad para los datos en cuestión.

3.2. Descripción del banco de datos utilizado

En el apartado 3.1 se comenta la procedencia de los datos y los tratamientos que fueron realizados sobre dichos datos. Tras la aplicación de modelo físico del ECMWF y el PCA, el conjunto de datos resultante presentaba los siguientes elementos:

- 'X': Que corresponde a las primeras 50 componentes principales tras aplicar el PCA al banco de datos original.
- 'Y': Que corresponde a la etiqueta o valor a predecir, en este caso la temperatura en la superficie de la Tierra, obtenida del ECMWF.
- 'lat': Que corresponde a la latitud del punto medido.
- 'lon': Que corresponde a la longitud del punto medido.
- 'transformer': Que corresponde al tipo de transformación a la que se ha sometido al dato, PCA en este caso.

Con estos datos se decidió crear un banco de datos para el proyecto, con un total de 10.000 muestras en la fase inicial, para realizar los primeros entrenamientos, y posteriormente se amplió a un total de 100.000 muestras. Este banco de datos se formó de las componentes 'X' e 'y', nombradas anteriormente, de modo que cada registro estaba formado por un vector de 51 elementos, que consiste en los 50 primeros componentes de la muestra original y, a continuación, el valor de su temperatura.

3.3. Módulo GAN

El modelo GAN consta de dos redes neuronales, el generador y el discriminador, que compiten entre si para generar nuevas muestras de una calidad aceptable.

A continuación se detalla el generador desarrollado para el proyecto.

3.3.1. Generador

En el proceso de creación de muestras sintéticas, el generador comienza su tarea a partir de un vector de ruido aleatorio, normalmente asociado a la letra z. Normalmente, estos valores aleatorios se obtienen de una distribución normal, y el tamaño del vector 100.

En la figura 12 se puede comprobar la estructura de uno de los generadores creados para este proyecto:

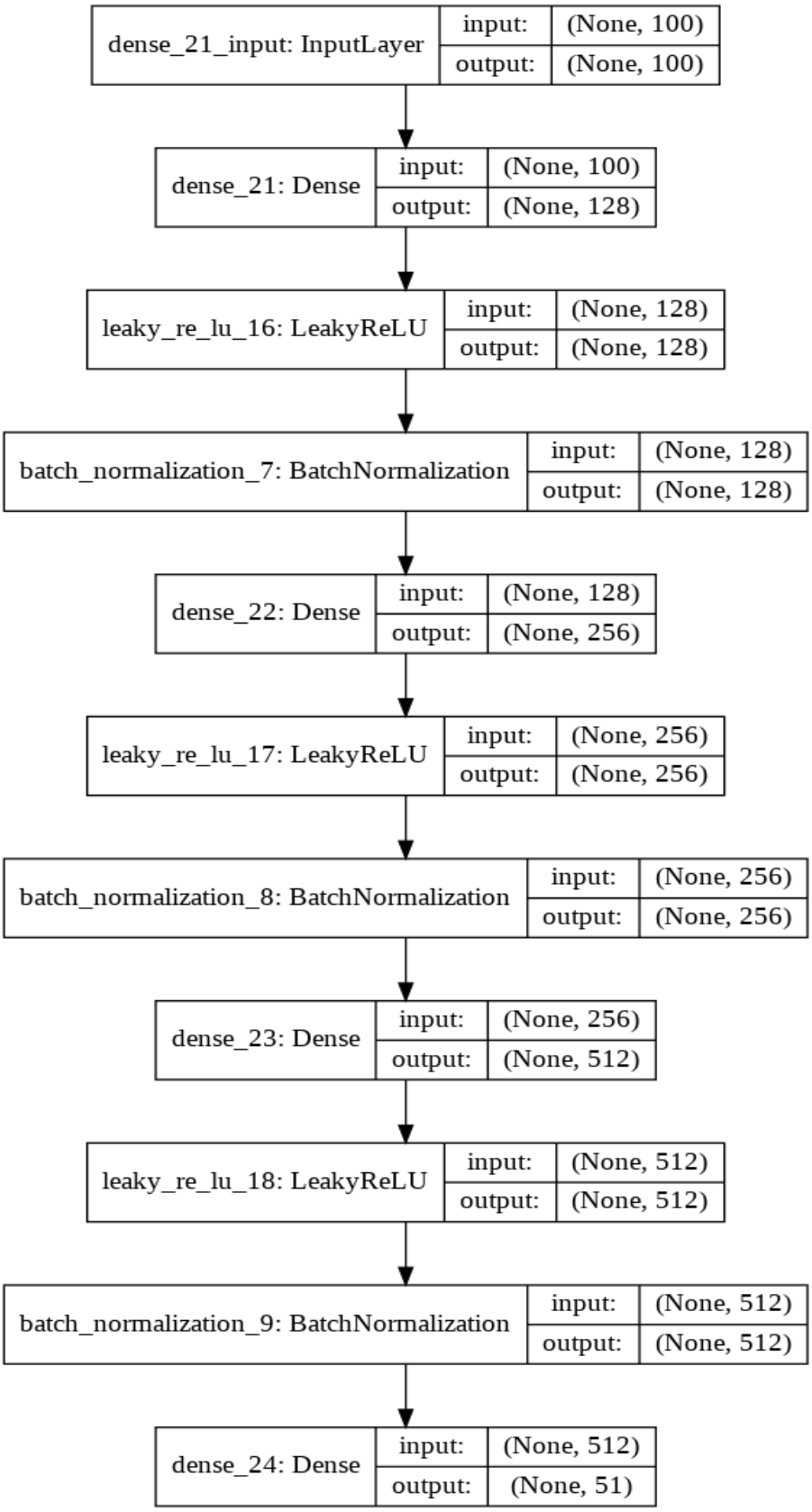

Figura 12: Generador de la GAN.

Durante el proyecto se entrenan diferentes modelos generadores. Cada uno de estos modelos corresponde a un generador entrenado con un número concreto de muestras. Sin

embargo, el funcionamiento de todos ellos es el mismo, por lo que a continuación se detalla la estructura de uno de estos generadores, que puede ser extrapolable al resto.

El modelo generador está formado por los siguientes elementos:

- Modelo secuencial basado en la librería de deep learning Keras [13].
- Capa de entrada: Con 100 nodos, correspondientes al vector de ruido Z.
- Primera capa densa, de 128 nodos, seguido de una función de activación LeakyReLU y un BatchNormalization.
- Segunda capa densa, de 256 nodos, seguido de LeakyReLU y BatchNormalization.
- Tercera capa densa, con 512 nodos, con LeakyReLU y BatchNormalization.
- Capa de salida, con 51 nodos, correspondientes a las 51 dimensiones que deseamos.

Cabe recalcar la terminología empleada en la lista anterior:

- Secuencial: Se refiere a la forma de construcción del modelo. En su forma secuencial, el modelo se construye capa a capa, siguiendo el orden de creación.
- Capa densa: Capa de la red neuronal donde todas las neuronas de la capa anterior están conectadas a las de la capa siguiente.
- Leaky ReLU: Un tipo de función de activación, que transforma la señal de entrada en otra distinta para la siguiente capa. Fue explicado en el apartado 2.1.2
- BatchNormalization: Mecanismo que nos permite un entrenamiento más estable, estandarizando los datos para tener media 0 y desviación estándar 1.

Por tanto, el generador hace pasar el vector de ruido inicial por tres capas densas, que incorporan las activaciones LeakyRelu y el paso de BatchNormalization. En el anexo 'GAN-Generador' se incluye un ejemplo simplificado de Generador de GAN muy similar al implementado en el proyecto. Para ello, se ha utilizado el lenguaje de programación Python, y la librería Keras de deep learning.

En los anexos de este documento se pueden encontrar las referencias a todos los archivos de implementación desarrollados durante este proyecto.

3.3.2. Discriminador

La segunda red que forma la GAN es el discriminador. Este se encarga de discernir entre muestras reales y muestras generadas por el generador, devolviendo un único valor, que corresponde a la puntuación de realismo de la muestra suministrada.

Tal como ocurre con los generadores, diferentes discriminadores han sido entrenados durante el proyecto, con diferentes números de muestras reales, pero se explicará uno, dado que todos ellos son iguales.

En la figura 13 se puede comprobar la arquitectura del discriminador:

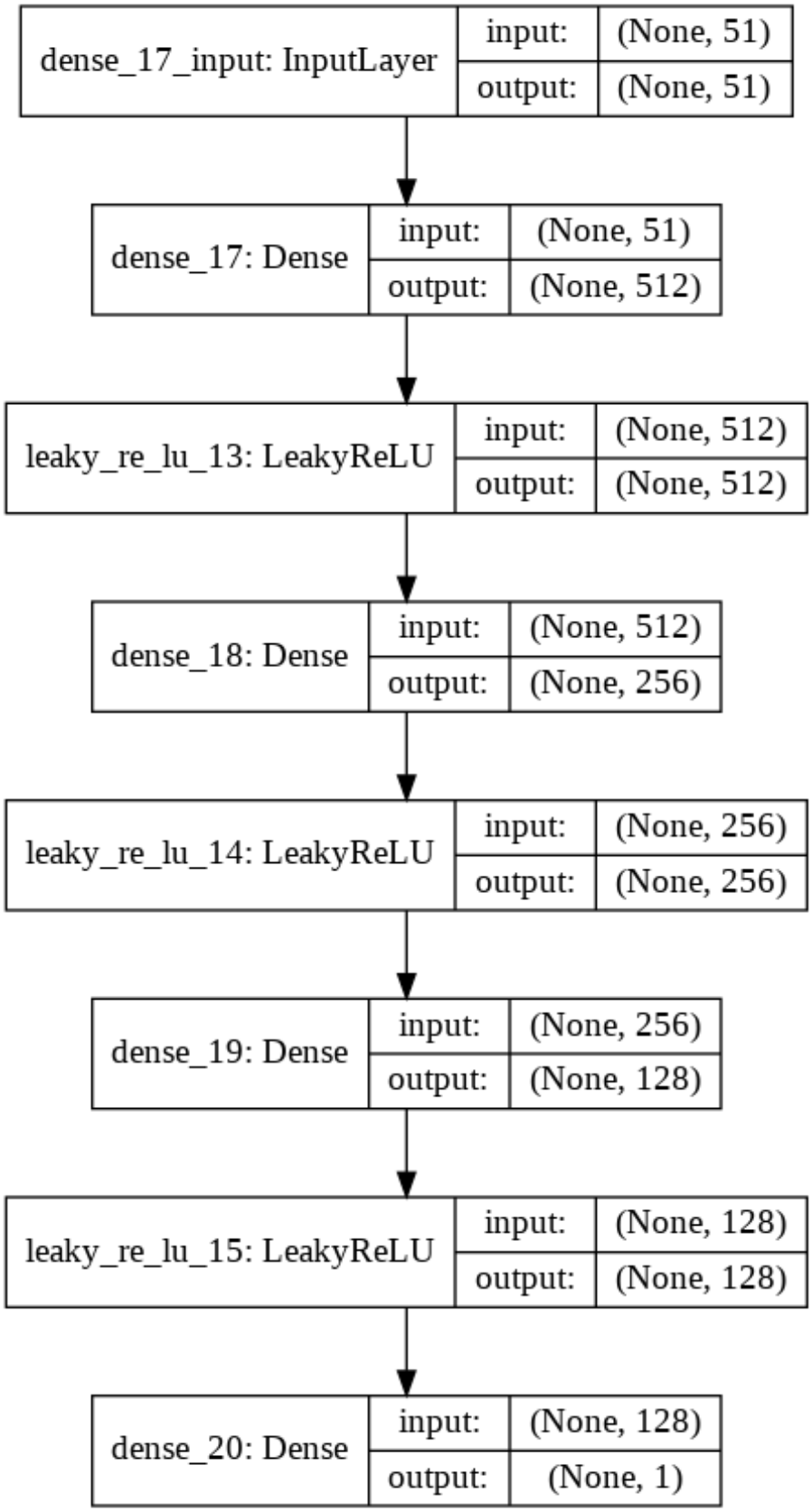

Figura 13: Discriminador de la GAN,

- Modelo secuencial basado en la librería de deep learning Keras.

- Capa de entrada: Con 51 nodos, a los datos originales, o a las salidas de la red generadora.
- Primera capa densa, de 512 nodos, seguido de una función de activación LeakyReLU .
- Segunda capa densa, de 256 nodos, seguido de LeakyReLU .
- Tercera capa densa, con 128 nodos, con LeakyReLU .
- Capa de salida, con 1 solo nodo, correspondiente a la probabilidad de que el dato que se le ha suministrado sea real o no, con valores entre 0 y 1. Se le aplica la función de activación Sigmoide a dicha capa, para que devuelva los valores entre 0 y 1.

Cabe destacar que, a diferencia del generador, en esta segunda red no se ha aplicado la técnica BatchNormalization a las capas. Esto es debido a que dicha técnica, aplicada en todas las capas suele llevar a inestabilidades en el entrenamiento [14], y además, se comprobó que los resultados del entrenamiento de la GAN eran mejores sin aplicar dicha técnica..

En el anexo 'GAN-Discriminador' se puede observar un ejemplo de implementación de discriminador de GAN desarrollado en el proyecto, así como sus hiperparámetros y valores más relevantes. En los ficheros adjuntos, se puede encontrar el discriminador integrado en la estructura GAN desarrollada para este proyecto.

3.3.3. Función de coste y entrenamiento de la GAN

En el entrenamiento de la GAN, la función de coste desempeña un papel fundamental. Como se comenta en el apartado 2.2.3, el objetivo de ambas redes es encontrar un punto de equilibrio en el que las muestras del generador sean lo suficientemente parecidas a las reales.

Para aproximarse a este equilibrio, la GAN utiliza una función de coste compartida entre ambas redes. Sin embargo, aunque esta función cuente con parámetros de las dos redes, cada una solo será capaz de modificar los suyos propios, aunque influirán en el entrenamiento del otro.

A continuación se muestran los resultados de entrenamiento de una GAN. Por simplicidad, se ha elegido el número de muestras reales 7000, valor en el que los resultados son aceptables. Sin embargo, se han desarrollado diferentes combinaciones de muestras reales y modelos GAN para explorar las diferentes posibilidades en cuanto a resultados.

Hay que destacar que, aunque la red GAN se entrena de forma conjunta, para el proceso de creación de muestras sintéticas se extrae únicamente el módulo generador. De este modo, una vez la GAN completa ha sido entrenada, se utiliza el generador para crear el número de muestras artificiales deseado.

En la figura 14 se muestra una representación de la primera dimensión de los datos (Eje X) frente a la dimensión 51 (Eje Y), que corresponde al valor objetivo de temperatura. Se superponen, además, los valores reales frente a los generados por la GAN, para poder ver sus diferencias.

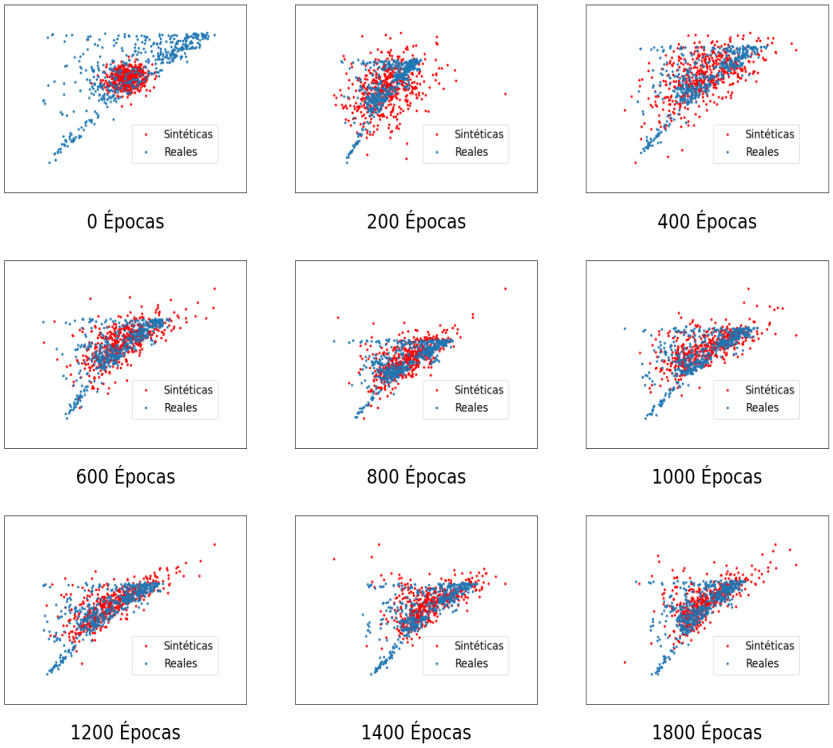

Figura 14: Entrenamiento de la GAN.

Se puede observar como, en épocas tempranas, los datos generados se parecen a los propios de una distribución Gaussiana, ya que proceden de ruido aleatorio. Con el paso de las épocas, la GAN aprende la estructura de los datos objetivo, y genera formas más similares a las reales. En épocas avanzadas del entrenamiento, la GAN muestra que el núcleo de las muestras está en el mismo lugar que en el de las reales, y además, su forma general se asemeja a las mismas.

En la siguiente subsección se comentará el modelo RBIG desarrollado.

3.4. Módulo RBIG

Para el desarrollo del modelo RBIG, se obtuvo una versión de los desarrolladores del mismo. El objetivo de añadir este modelo al proyecto recae en comprobar su eficacia en la generación de datos y, además, comparar los resultados obtenidos con los de la GAN.

El proceso para generar muestras con RBIG es el siguiente:

- Entrenar el modelo RBIG con el conjunto de datos reales.
- Tras el entrenamiento, la PDF de los datos habrá sido transformado a una Gaussiana.
- Generar muestras Gaussianas aleatorias. El número debe ser el de muestras sintéticas deseadas.
- Aplicar la transformación inversa a la que se le aplica a los datos en el entrenamiento del modelo. De esta forma, se convierten los datos Gaussianos en datos de la PDF original desconocida.

La figura 15 muestra los datos originales frente a los sintéticos generados, eligiendo siempre la primera dimensión frente a la última, que corresponde a la temperatura. Se ha elegido el modelo entrenado con 7000 muestras reales.

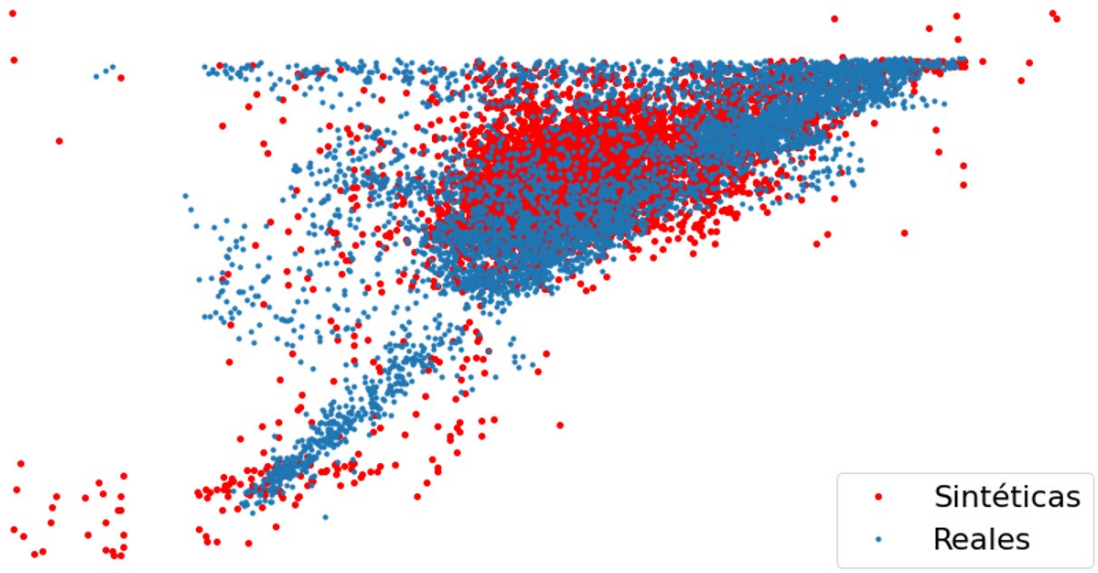

Figura 15: Comparación de muestras reales y sintéticas con RBIG

Se observa como, de forma análoga a las GAN, los datos generados parecen captar la forma de los reales, lo que implica un parecido con estas muestras aceptable.

En la siguiente subsección se comentará el modelo de regresión utilizado para llevar a cabo los experimentos.

3.5. Modelo de regresión

Para la evaluación de las muestras generadas se utilizará un modelo de regresión basado en una red neuronal sencilla. El objetivo de esta será predecir la temperatura de la atmósfera proporcionada por el modelo físico del ECMWF, a partir de los datos que se le suministren.

La red contará con los siguientes elementos:

- Inputs: Vector con 50 elementos, correspondientes a los primeros valores del PCA del banco de datos del instrumento IASI, y otro vector con las temperaturas de cada registro.
- Outputs: Un único valor correspondiente a la temperatura de la atmósfera predicha por el modelo.
- Una capa densa, con un número de neuronas igual a 64.

- Una capa de dropout, que intenta que la red evite el overfitting.
- Una capa densa de 32 neuronas.
- Una capa de output.

En la figura 16 se puede observar la arquitectura de la red.

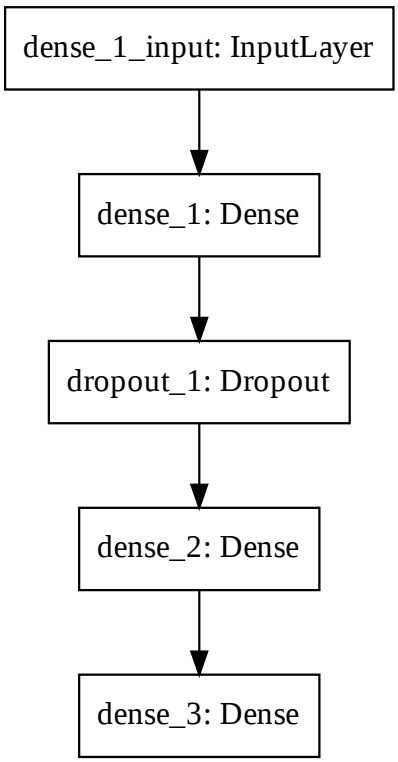

Figura 16: Modelo de regresión.

4. Experimentos realizados y resultados obtenidos

Esta sección pretende comentar los experimentos realizados durante el proyecto, así como los resultados obtenidos de los mismos.

4.1. Experimento propuesto

La propuesta del experimento es la comprobación de la eficacia de añadir muestras generadas a un modelo de predicción de temperaturas. Se pretende averiguar dicha eficacia para números de muestras disponibles bajos, así como para valores más elevados.

Con esta premisa, el experimento plantea los siguientes pasos a seguir:

- Comprobar, en primer lugar, la eficacia del modelo regresor utilizando únicamente los datos reales disponibles (no sintéticos).

- Generar diferentes números de muestras sintéticas con los modelos GAN y RBIG entrenados previamente.
- Mezclar los conjuntos de datos sintéticos y reales y comprobar la calidad del modelo de regresión.
- Comprobar si la adición de sintéticos mejora el error, y hasta que número resulta eficaz dicha adición.
- Comprobar el número de muestras reales en la que añadir sintéticos no resulta eficaz.
- Comprobar si la adición excesiva de sintéticos penaliza el resultado.
- Comprobar cual de los dos modelos generativos analizados obtiene mejores muestras sintéticas para este problema.

Para el experimento se han supuesto cinco escenarios en lo referente a número de muestras reales disponibles. Estos números de reales son: 500, 1000, 3000, 7000 y 20000. Por tanto, se han evaluado todos los pasos anteriores para cada uno de estos cinco supuestos.

Para asegurar un buen funcionamiento del experimento, se ha utilizado siempre la misma división entre conjunto de entrenamiento, validación y test. Concretamente, se ha utilizado el primer 70 % de las muestras disponibles como entrenamiento, el siguiente 20 % como validación, y el último 10 % como grupo de test.

En el desarrollo del experimento, el primer paso realizado es el entrenamiento de los modelos de regresión con diferentes combinaciones de muestras reales, sin la generación de sintéticos. Para evitar posibles efectos debido al azar, se han repetido un total de tres veces todos los entrenamientos, logrando una solución de compromiso entre robustez de los resultados y tiempo de entrenamiento. Para el cálculo de los resultados finales, se ha realizado la media de dichas tres mediciones.

El segundo paso a realizar es el entrenamiento de las redes generadoras, RBIG y GAN. Como ocurre en el caso anterior, se ha creado una red de cada tipo para cada uno de los números de muestras reales propuestos. Para 500 muestras reales disponibles, por tanto, se dispone una GAN entrenada únicamente con dichas 500 muestras, de igual forma que se dispone de un modelo de regresión que se entrena con las mismas.

Tras el entrenamiento de las redes generadoras, la creación de los sintéticos es el siguiente paso. Se ha optado por crear una cantidad de muestras sintéticas igual a las combinaciones de reales, por simplicidad. Así, las opciones de sintéticas disponibles son: 500, 1000, 3000, 7000 y 20000.

Por tanto, se tiene un total de 25 combinaciones de reales y sintéticos para cada modelo generador, que corresponden a cinco combinaciones de reales por cinco combinaciones de sintéticos.

Por último, se hace uso del regresor correspondiente a cada combinación para obtener las predicciones y las medidas de error en test. Cabe destacar que, todos los entrenamientos han sido realizados 3 veces y los resultados han sido obtenidos aplicando la media, tal y como ocurría anteriormente.

4.2. Consideraciones del entrenamiento de las redes

En las secciones 2.2, 3.3, 2.3 y 3.4 se comentan los aspectos generales de creación y funcionamiento de los modelos GAN y RBIG, respectivamente. Sin embargo, durante el desarrollo del proyecto se han tomado ciertas acciones concretas para ajustarse al mismo.

En esta sección se comentarán los elementos más relevantes de cada uno de los modelos, en lo que se refiere a su entrenamiento.

4.2.1. GAN

El modelo GAN presenta una gran complejidad a la hora de plantear su entrenamiento. Esto se debe, principalmente, a que existen una buena cantidad de parámetros dentro de la misma, y su ajuste requiere del método de prueba y error. Además, cada problema requiere una configuración de GAN distinta, por lo que no existen configuraciones estándar que funcionen en general.

La GAN presenta, además, algunos problemas por el hecho de su competición entre el generador y discriminador, lo que implica que dos redes neuronales busquen un punto de equilibrio.

En términos de redes neuronales, el principal problema técnico que presenta la competición entre dos redes a la vez es que pueden llegar a fallos de convergencia.

Los fallos de convergencia y problemas más comunes son los siguientes:

- En lugar de encontrar un punto de equilibrio, el generador oscila generando ejemplos específicos del dominio. Esto implica que la GAN estaría oscilando entre generar dos tipos de muestras concretas, sin llegar a alcanzar el equilibrio. [15]
- "Mode collapse", que implica que el generador crea la misma salida para múltiples entradas del vector 'z'.
- No existe una buena métrica para evaluar el funcionamiento de la GAN durante el entrenamiento. La mejor solución por el momento es, durante el mismo, visualizar las muestras generadas. [16]

A día de hoy, no existe una fundamentación teórica sobre como diseñar y entrenar modelos GAN, por lo que para lidiar con los problemas anteriores, entre otros, se requiere de experimentar con la misma. Sin embargo, existen algunas prácticas que suelen facilitar el buen funcionamiento, a pesar de lo anterior.

En la investigación realizada en el artículo "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks"[14], múltiples arquitecturas y composiciones de parámetros fueron realizadas por el método de prueba y error, y sus conclusiones establecen un punto de inicio sobre el que construir las GAN.

En el artículo, podemos encontrar sugerencias sobre el entrenamiento, como la utilización de Batch Normalization, utilización de funciones de activación concretas, o el uso de ciertos optimizadores.

En el proyecto se han utilizado las siguientes de sus recomendaciones, para mejorar el proceso de entrenamiento:

- Utilización de Batch Normalization. Esta técnica estandariza la entrada que proviene de la capa anterior, para que tenga media cero y varianza unidad. Esta acción tiene el efecto de estabilizar el proceso de entrenamiento. Como punto de partida,

se recomienda aplicar este método tanto el en generador como en el discriminador, salvo en las capas de salida.

- Utilizar ReLU, Leaky ReLU y Tanh. Las funciones de activación ReLU, como se comenta en el apartado 2.1.2, permiten abordar el problema del desvanecimiento del gradiente. El artículo recomienda utilizar funciones ReLu para el generador, y Leaky ReLu para el discriminador. Adicionalmente, se utiliza la función tangente hiperbólica en la capa de salida del generador. De igual forma las entradas se escalan al rango [-1,1] para su correcto funcionamiento con la función de activación anterior.

- Utilización del optimizador Adam. La utilización de este optimizador frente a otros resulta beneficioso en muchos casos, por lo que se suele utilizar de forma predeterminada. Además, el artículo proporciona unos parámetros que suelen funcionar de manera correcta, y que se han utilizado en el proyecto, tanto para el generador como para el discriminador. Estos corresponden a una tasa de aprendizaje de 0.001 y un momentum (beta1) de 0.5.

Con estas técnicas, obtenidas del artículo anterior, el comienzo de entrenamiento de la GAN resultó más sencillo.

A pesar de ello, fueron necesarias diferentes entrenamientos, basados en prueba y visualización de resultados, para encontrar las combinaciones de parámetros adecuados para este proyecto. Finalmente, los parámetros elegidos y técnicas aplicadas son las siguientes:

- Optimizador Adam, tanto para el generador como el discriminador, con una tasa de aprendizaje de 0.001 y momentum de 0.5.

- Batch size, o tamaño del lote de 128.

- Número de épocas de entrenamiento: 2000, valor suficiente en el que el entrenamiento de la GAN ya genera datos adecuados para todos los supuestos de cantidad de muestras reales.

- Número de capas y unidades dentro de cada capa, comentado en el apartado 3.3.

- Tamaño del vector de ruido aleatorio 'z', utilizado por el generador: 100.

- Estandarización de los datos de entrada, previamente al entrenamiento. Se comprobó que realizar este paso previo mejoraba el entrenamiento de la GAN, por lo que fue incluido.

- Utilización de Batch Normalization en el generador. Tras varias pruebas, se comprobó que la adición del mismo en el discriminador no aportaba beneficios.

- Utilización de Leaky ReLU tanto en el generador como en el discriminador, en todas sus capas, salvo la última.

Tras establecer todo lo anterior, se comprobó como seguía apareciendo un problema recurrente en la GAN, la velocidad de entrenamiento de generador y discriminador no es la misma. Esto conlleva que una de las dos redes aprendera a realizar su tarea con mayor rapidez, cuando lo buscado es que lo realicen de forma similar, para encontrar un equilibrio entre ambas.

Para lidiar con este problema, se introdujo una modificación en la selección de muestras de entrenamiento de cada red. Esta modificación consiste en, tras cada iteración, comprobar cual de las dos redes ha mejorado en mayor medida. Tras esto, a dicha red se le alimenta con menos muestras para la siguiente iteración, mientras que a la segunda red se le da un número mayor de muestras. Con esto se consigue que ambas redes se estabilicen en el ritmo de aprendizaje, lo que llevó al resultado final, mostrado en la figura 14

Por último, cabe destacar lo referente al tiempo de entrenamiento requerido por la GAN. Esta cantidad de tiempo, en contra de lo que cabría esperar, no aumenta conforme aumenta el número de muestras con la que se le entrena, sino que se mantiene estable en valores cercanos a 5 minutos, 15 segundos.

En la tabla 1 se puede observar los tiempos de entrenamiento de la GAN según el número de muestras que se han utilizado, siempre para un número de épocas igual a 2000 y utilizando el entorno de Google Colab, acelerado por una GPU.

Número de muestras				
500 muestras	1000 muestras	3000 muestras	7000 muestras	20000 muestras
5min,32seg	5min,23seg	5min,01seg	5min,45seg	5min,12seg

Cuadro 1: Tiempo de entrenamiento de la GAN según número de muestras.

4.2.2. RBIG

En el caso del modelo generador RBIG, el proceso de entrenamiento resulta menos complejo, ya que no dispone de tantos parámetros a configurar manualmente. En su lugar, el artículo original establece unos valores por defecto para dichos parámetros, que son los utilizados en este proyecto.[8].

Sin embargo, cabe destacar algunos comportamientos llevados a cabo durante el entrenamiento del modelo RBIG que se deben tener en cuenta. Como se comentó en los apartados 2.3 y 3.4, el objetivo de RBIG es transformar unos datos, con una PDF desconocida, en unos con PDF Gaussiana. Para ello, el modelo se organiza en una serie de denominadas 'capas', formadas por una Gaussianización marginal de una dimensión de los datos, y la posterior rotación del conjunto. Por tanto, por cada iteración, se añade una nueva capa a la red. En el artículo original se demuestra que, tras añadir suficientes capas, el resultado será una PDF Gaussiana, ya que la aplicación de una Gaussianización marginal y una rotación siempre genera una Gaussianización.

Otro punto a tener en cuenta es el criterio de parada del entrenamiento. A diferencia de la GAN comentada anteriormente, al modelo RBIG no se le indica un número exacto de épocas a entrenar, sino que este continúa añadiendo capas hasta que se cumpla un criterio determinado. El modelo RBIG dejará de añadir capas, y por ende, de seguir entrenando, cuando se compruebe que la adición de una capa ya no Gaussianiza los datos. En este punto, por tanto , los datos ya habrán alcanzado la forma deseada, ya que la única manera de que una capa no Gaussianice es que el conjunto ya sea Gaussiano.

En el artículo original también se demuestra que estas transformaciones son invertibles, por lo que, una vez alcanzado el punto de parada, podrán generarse nuevas muestras generando datos Gaussianos aleatorios y aplicándoles las transformaciones inversas desarrolladas por el modelo RBIG.

En lo referente a los tiempos de entrenamiento, RBIG muestra una tremenda mejora si se compara con el modelo GAN, ya que presenta unos tiempos de entrenamiento de la

red que oscilan entre los 10 y los 15 segundos, según el número de muestras. En la tabla 2 se puede observar los tiempos de entrenamiento de la GAN según el número de muestras que se han utilizado, de nuevo en el entorno de Google Colab, con aceleración de GPU.

Número de muestras				
500 muestras	1000 muestras	3000 muestras	7000 muestras	20000 muestras
9segundos	9,2segundos	9.3segundos	10segundos	15segundos

Cuadro 2: Tiempo de entrenamiento de RBIG según número de muestras.

4.3. Análisis de resultados

Esta sección engloba los resultados obtenidos durante el proyecto. Adicionalmente a este documento, se ha desarrollado un entorno Shiny que agrupa todas las visualizaciones obtenidas, de forma fácil y accesible. En el anexo 7.3 se puede encontrar toda la información al respecto. [1]

4.3.1. Error en test frente al número de reales con la red GAN

Las primeras conclusiones se obtienen tras analizar los errores producidos por el regresor sobre el conjunto de test, una vez entrenados.

En la figura 17 se muestra la evolución del error sobre el conjunto de test. Cada una de las gráficas mostradas se corresponde con uno de los números de muestras reales, comentados en el apartado 4.1. Dentro de estas, se observa el error del modelo de regresión calculado únicamente con los datos disponibles en rojo, y a continuación, los errores de test obtenidos para cada combinación de sintéticos añadidos.

De este conjunto de figuras se extraen los siguientes resultados referentes al entrenamiento con GAN:

- Se puede observar como, para valores de muestras reales bajos, como 500, 1000 e incluso 3000, la adición de sintéticos resulta beneficiosa, ya que la reducción del error respecto al modelo original sin sintéticos es muy grande. Se observa como valores de error de 60, 59 y 13 grados en el número de muestras anteriormente nombrado, se reduce a valores entre 6,5 y 5,5 grados en los modelos con sintéticos añadidos.

- Se observa, de igual forma, que en modelos con una cantidad de muestras reales alta el error es menor, de alrededor de 4 grados, hecho que tiene sentido ya que los datos reales siempre van a ser mejores que las generadas.

- En estos tres valores, 500, 1000 y 3000 muestras reales, se observa como el número óptimo de sintéticos a añadir sería 3000, ya que a partir de este punto la adición de más sintéticos ya no reporta disminuciones del error.

- Para el valor de 7000 reales, comprobamos como de nuevo, el valor de 3000 sintéticos añadidos reporta la mejor disminución del error, aunque las escalas son mucho menores que en los casos anteriores.

[1] Shiny link: https://eduardodavilatfm.shinyapps.io/Shiny/

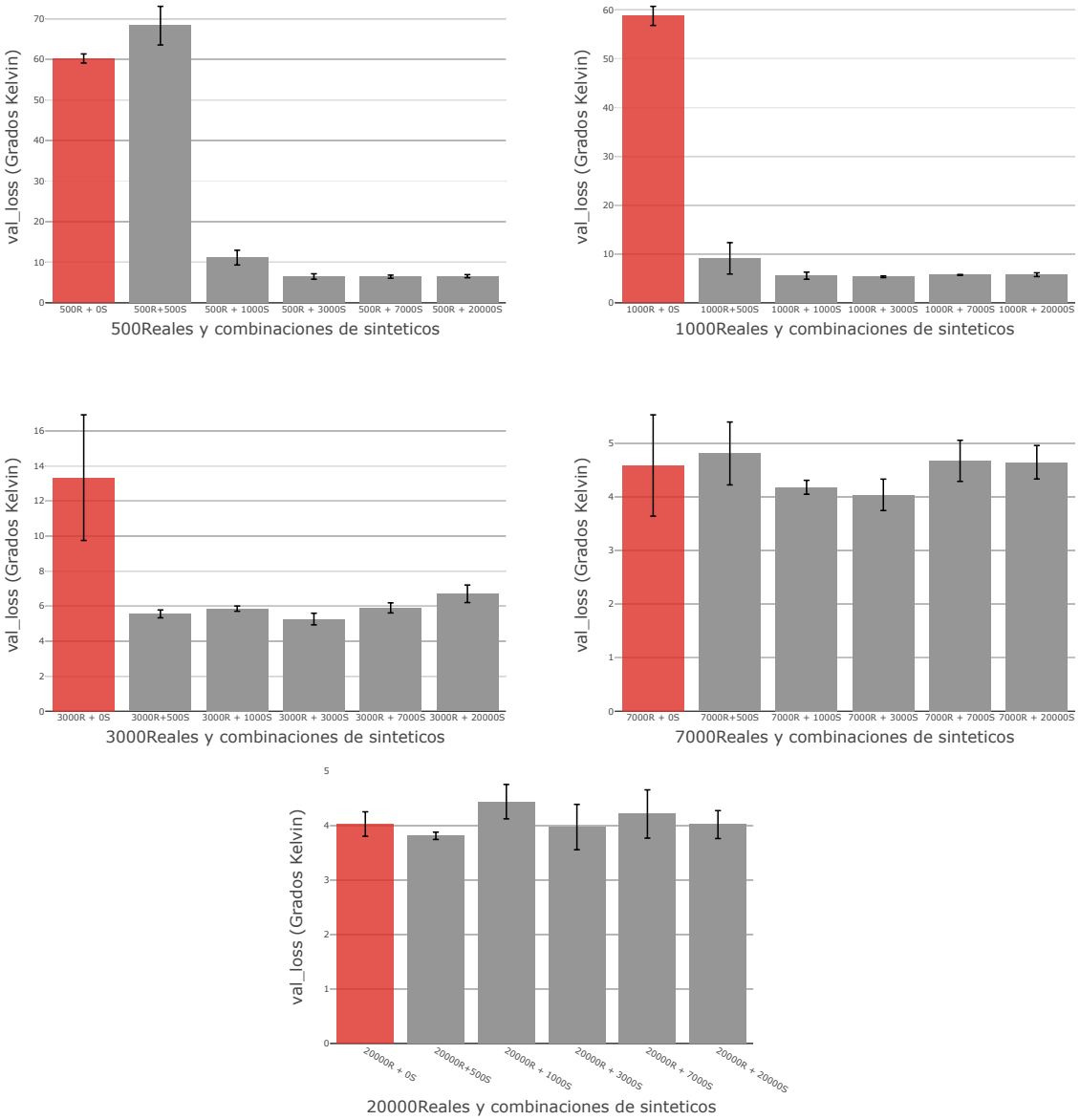

Figura 17: Gráficas de error en test de la GAN.

- En cambio, para modelos con muchas muestras reales, 20000 en este caso, se observa como no resulta eficaz añadir sintéticos, ya que el propio modelo es capaz de predecir correctamente la temperatura por si solo, y los datos sintéticos aportan mejoras mínimas, o empeoran el resultado.

Por tanto, se puede afirmar que, en este problema, resulta muy beneficioso utilizar una GAN como modelo generativo si se dispone de un número de muestras bajo, de hasta 3000 muestras, ya que a pesar de que la generación de sintéticos conlleva errores, son menores en comparación con la mejora que se obtiene respecto al modelo sin sintéticos.

4.3.2. Error en test frente al número de reales con la red RBIG

Tal y como ocurre en la sección anterior, los primeros resultados sobre el modelo RBIG se pueden encontrar en la figura 18.

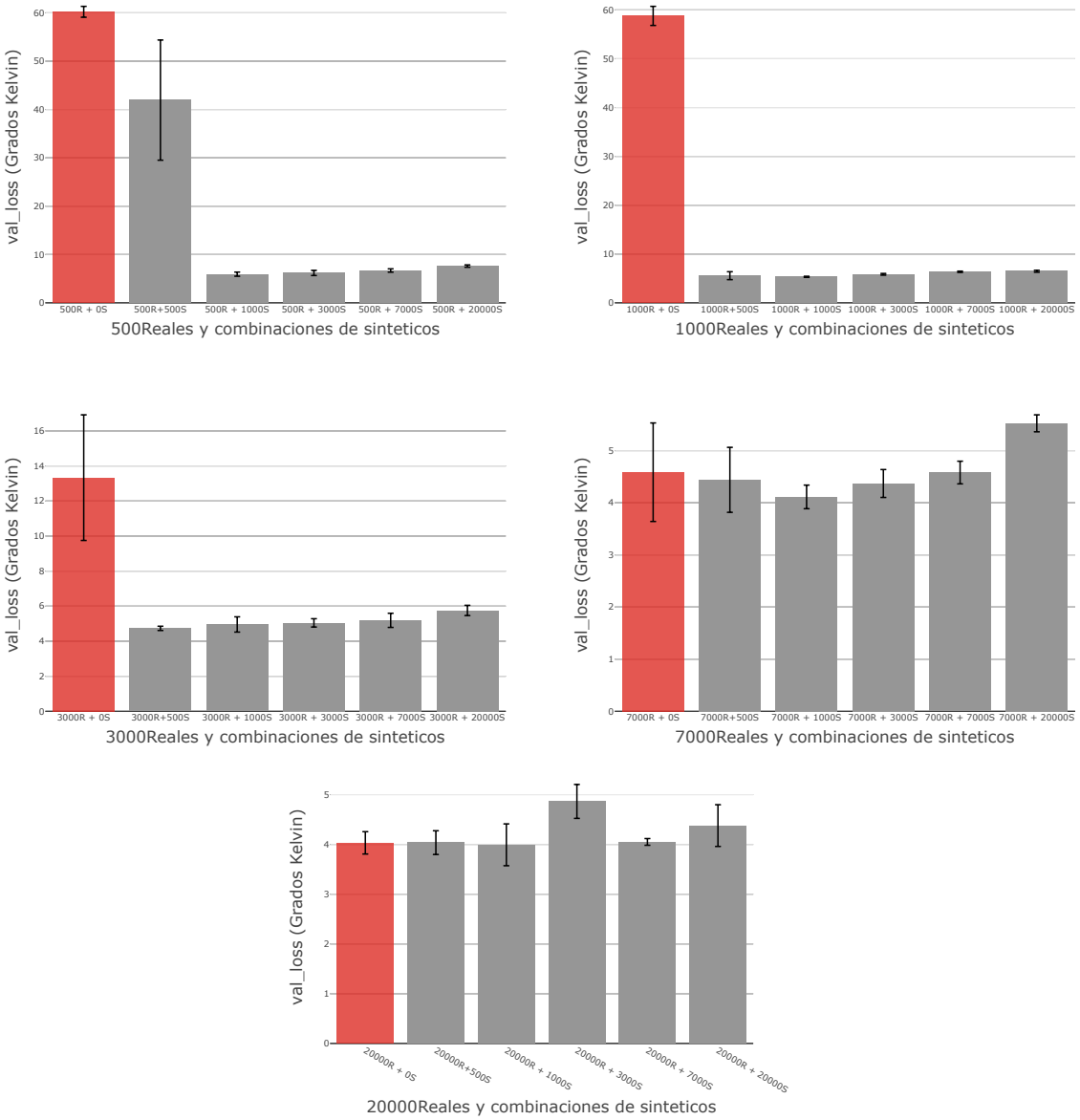

Figura 18: Gráficas de error en test de RBIG.

De este conjunto de figuras se extraen los siguientes resultados referentes al entrenamiento con RBIG:

- Para valores de 500 y 1000 reales, podemos observar como, claramente, añadir sintéticos resulta beneficioso, reduciendo drásticamente el error. En el caso de 500, se observa que, partiendo de un error base de alrededor de 60 grados, con la adición de 1000 sintéticos ya obtenemos el valor de error mínimo en todas las pruebas realizadas, alrededor de 5 grados. La adición posterior de sintéticos va aumentando el error, debido probablemente a que el modelo RBIG, al haber sido entrenado únicamente con dichas 500 muestras, no genera datos del todo precisos. Para el caso de las 1000 muestras reales, en cambio, vemos como con la suma de 1000 reales y 500 sintéticos ya alcanzamos un mínimo, pero el error ya no aumenta apenas, por lo que es de suponer que RBIG ha aprendido la estructura de los datos lo suficiente como para generar datos que tengan sentido.

- En los 3000 datos reales, que parten de un error de 14 grados, la adición de sintéticos reduce el error, como en los casos anteriores, aunque aumentar este número de sintéticos va aumentando el error. Esto suponemos que es porque el propio modelo base con 3000 muestras ya ha aprendido mejor como son los datos, reduciendo el error de 60 a 14 grados, y añadir sintéticos al principio mejora, por tener más muestras, pero al final estos sintéticos son peores que los reales y acaban lastrando el aprendizaje.

- En el caso de 7000 sintéticos, se observa como añadir hasta 1000 sintéticos reporta una pequeña mejora en el error, mientras que añadir una cantidad mayor empeora notablemente el resultado.

- En el caso de 20000 muestras, podemos observar claramente como no resulta conveniente crear datos sintéticos, ya que todos los resultados se mantienen al nivel del modelo sin sintéticos (alrededor de 4), o lo empeoran.

4.3.3. Error frente al número de sintéticos en la red GAN

Una vez comprobados los errores sobre test, se comprueba ahora la evolución que presenta el error conforme se añaden sintéticos, siempre partiendo del modelo sin ninguno. Para ello, se utilizan los historiales de entrenamiento de cada regresor, que han sido guardados previamente. Estos historiales cuentan con 2000 valores de error cada uno, correspondientes a cada una de las 2000 épocas entrenadas. Para unificar los datos, se han escogido para cada uno de dichos entrenamientos, la media de los últimos 100 valores. De esta forma, se tiene de un valor medio al que ha llegado dicho modelo, comprobado en este caso con el error de validación.

En la figura 19 se muestra como evoluciona el error de los conjuntos de 500 muestras reales y 1000 muestras reales, si se les va añadiendo sintéticos.

De los datos de esta figura, se observa como el primer valor corresponde a la media de los últimos 100 valores del entrenamiento del regresor con 500 reales y 0 sintéticos. El siguiente indicaría la media para el caso de 500 reales y 500 sintéticas, y así sucesivamente.

Cabe destacar que estos valores no son los mismos que los de la sección anterior, ya que estos provienen de la evaluación sobre test, y los actuales de la evaluación sobre validación y el historial de entrenamiento.

En la figura 19, se observa como el error disminuye con la adición de sintéticos, aunque para valores altos, en 500 reales comienza a aumentar. Sin embargo, en estos supuesto siempre resultaría beneficioso añadir los sintéticos, ya que el error de base es muy alto.

En la figura 20 se puede comprobar como, para un número de muestras reales 7000, la adición de sintéticos respecto al punto inicial resulta beneficiosa, pero a partir de los 3000 sintéticos, el error comienza a aumentar, por lo que un valor de 3000 o inferior sería una buena elección de sintéticos a generar.

Por último, en la figura 21, para muchas muestras reales, se comprueba como añadir nuevos sintéticos no aporta a penas beneficios, o empeora el resultado, en el caso de 20000 reales, por lo que no resultaría conveniente su generación.

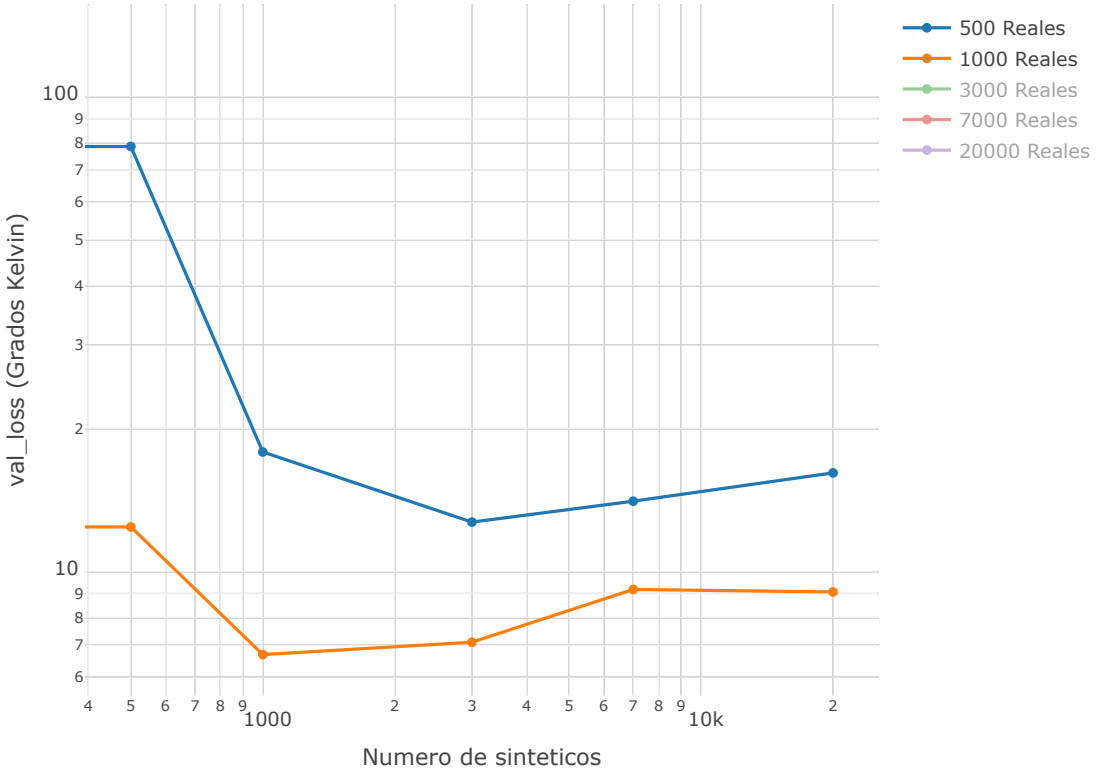

Figura 19: Error según sintéticos con valores bajos de reales.

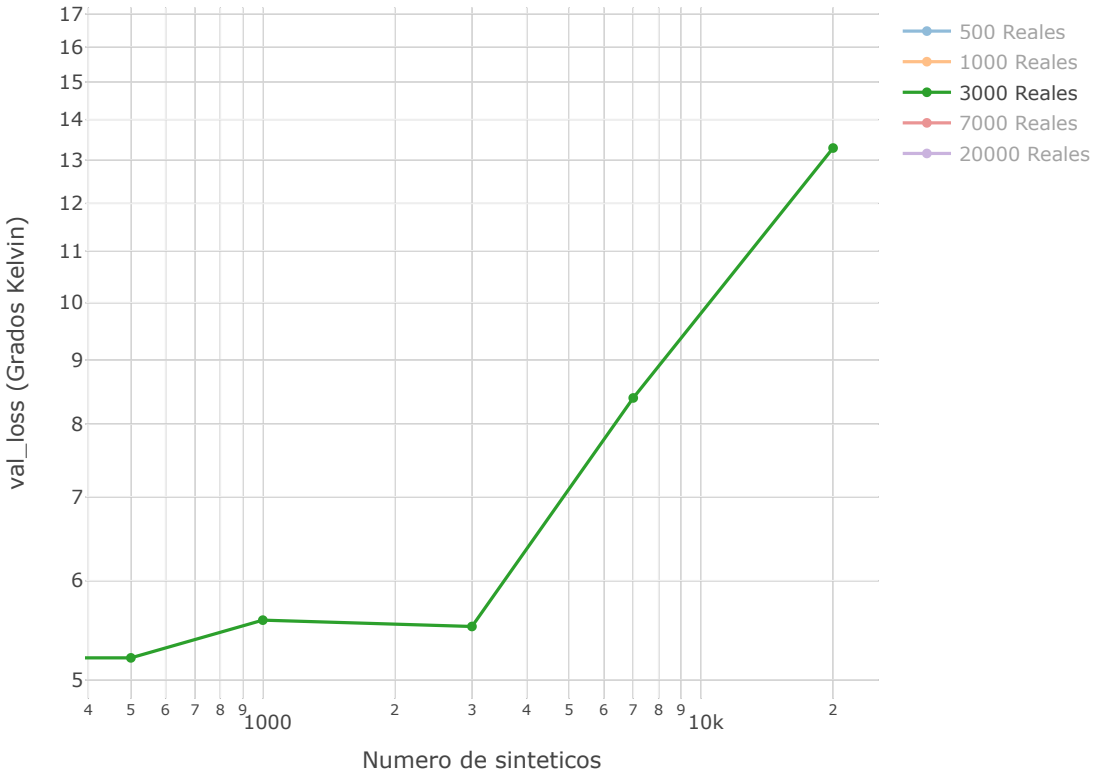

Figura 20: Error según sintéticos, 3000 reales

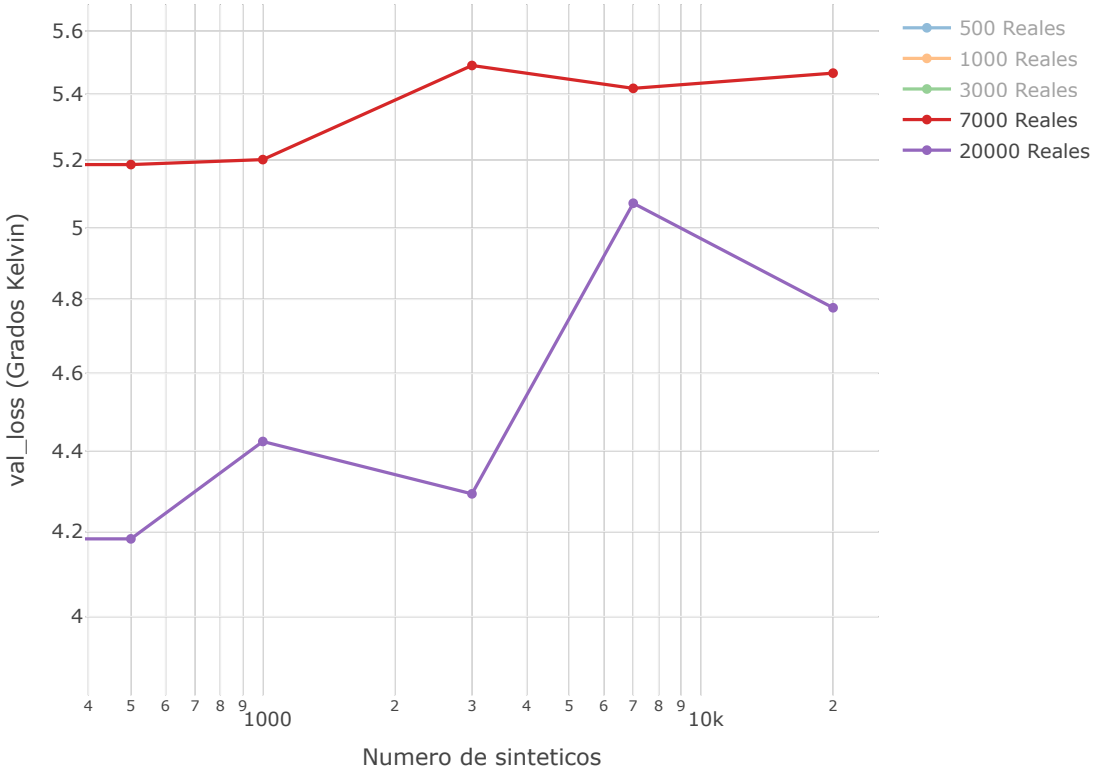

Figura 21: Error según sintéticos, valores altos de reales.

4.3.4. Error frente al número de sintéticos en la red RBIG

Se comentan a continuación los mismos resultados de la sección anterior para el modelo RBIG.

En la figura 22 se observa como, para 500 muestras reales, el error decae al añadir un mínimo de 1000 sintéticos. El mínimo error se alcanza añadiendo 3000, y después este comienza a subir. En el caso de 1000 reales, con 1000 sintéticos extra se logra el mínimo, momento en el cual el error aumenta de nuevo.

En la figura 23, que corresponde a 3000 reales, se observa una disminución del error notoria con 500 reales, aumentando el error tras esta cifra. Esto puede deberse a que con 3000 muestras reales, el propio modelo está cerca de poder aprender correctamente como son los datos, y con una pequeña adición de sintéticos lo consigue.

Por último, en la figura 24, que corresponde a 7000 y 20000 reales, se observa que, para 7000, añadir hasta 1000 sintéticos resultaría beneficioso, pero a partir de dicho momento el error aumenta notablemente. Para 20000 reales, no hay ninguna ocasión en la que generar sintéticos sea beneficioso, ya que siempre aumenta el error.

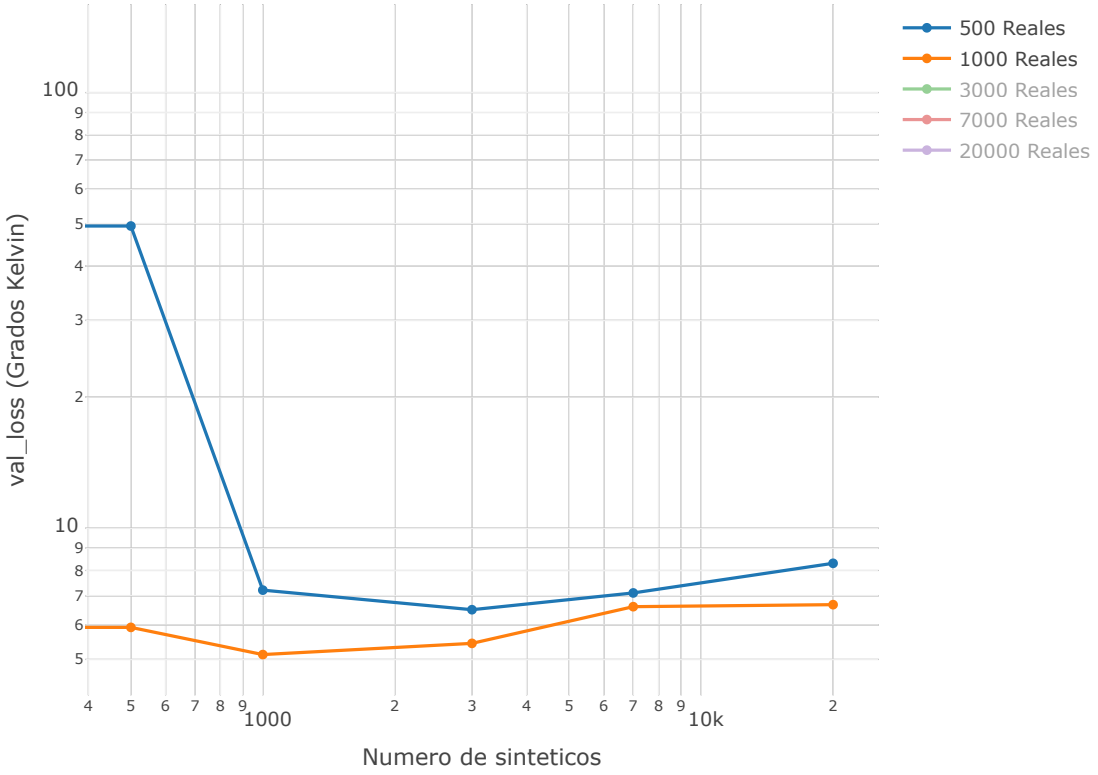

Figura 22: Error según sintéticos con valores bajos de reales.

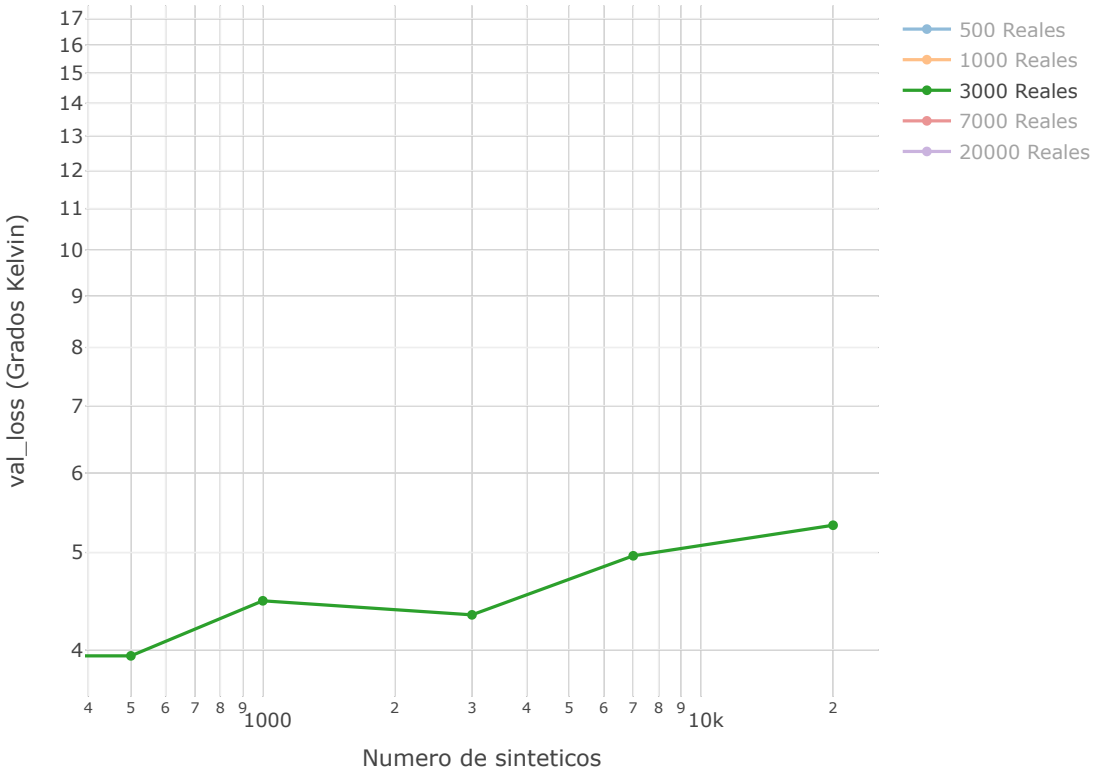

Figura 23: Error según sintéticos, 3000 reales.

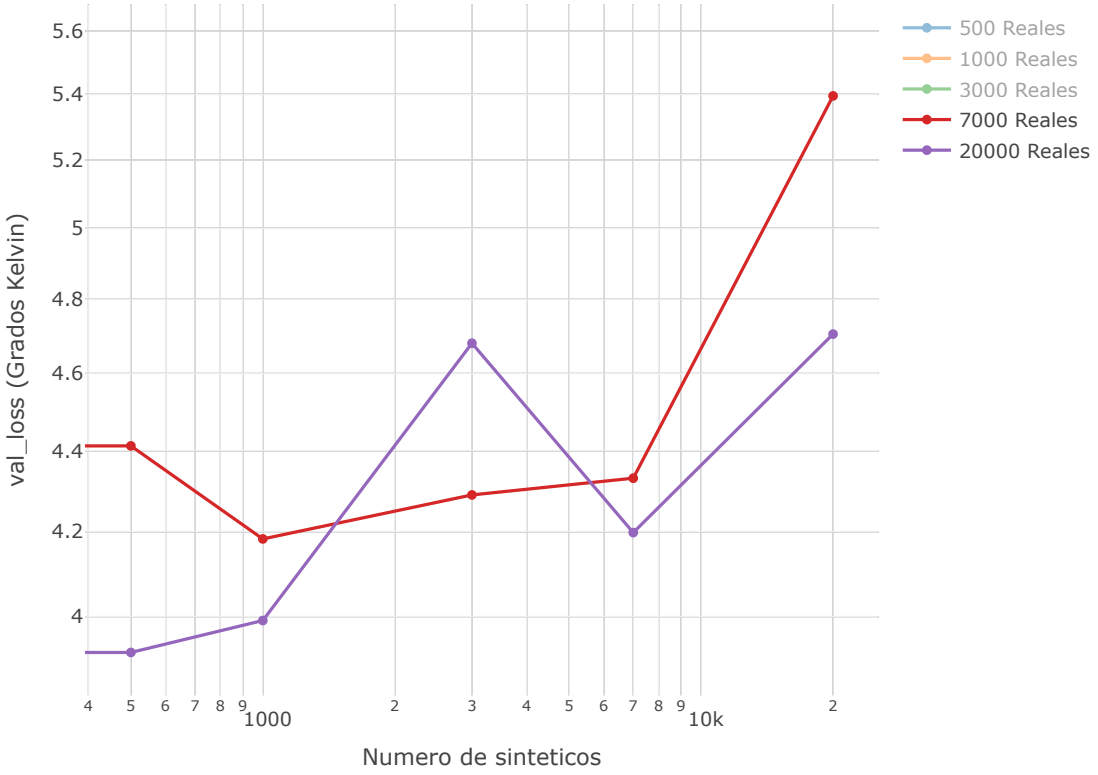

Figura 24: Error según sintéticos, valores altos de reales

4.3.5. Comparación de los modelos RBIG y GAN

En esta sección se va a comentar la comparación realizada sobre los modelos de RBIG y GAN, con el objetivo de determinar que modelo sería más eficiente a la hora de generar muestras, en el problema actual.

De la gráfica 25 se puede extraer, pues, lo siguiente:

- Para valores de reales bajos, como 500, se observa como RBIG representa una mejor opción que la GAN en casi todos los casos, siendo especialmente relevante en un abajo número de datos sintéticos (como 500 o 1000) donde el error que reporta RBIG es notablemente menor que el de la GAN. Una vez se aumenta dicho número, ambos se mantienen aproximadamente igual.

- En el caso de 1000 muestras reales, se observa de nuevo como, en menor diferencia, RBIG genera mejores resultados que la GAN para números de sintéticos bajos. Si se aumentan estos, en cambio, la GAN parece mantenerse más estable, mientras que RBIG empeora ligeramente.

- Para 3000 muestras reales podemos afirmar, en base a los resultados, que RBIG es una mejor opción, ya que su error es en la mayoría de las situaciones menor que el de la GAN.

- A partir de los 7000 datos reales,se observa que resulta poco eficaz añadir sintéticos, y aún así, para un número bajo de estos, RBIG parece presentar una reducción del error más alta que la GAN, aunque baja si se compara con las gráficas anteriores.

- Por último, con 20000 muestras reales, observamos como ninguno de los dos modelos resulta eficaz a la hora de mejorar el error. Una posibilidad es que el modelo ya tiene suficientes datos para aprender usando únicamente los reales, y añadir sintéticos no aporta más que ruido a su aprendizaje, con su correspondiente aumento del error.

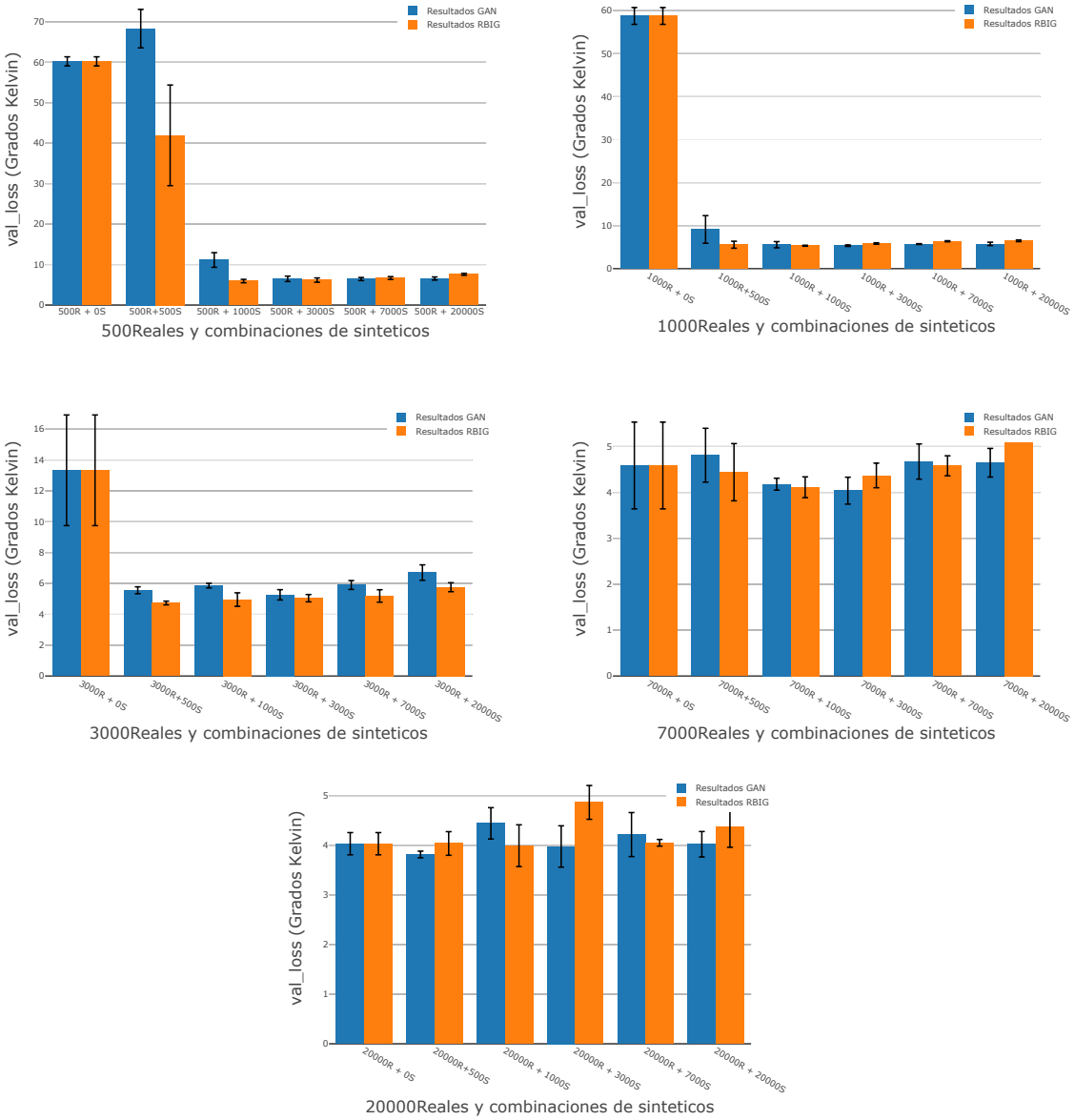

Figura 25: Comparación entre GAN y RBIG sobre test.

Por último, en la siguiente sección se comentarán las conclusiones del trabajo.

5. Conclusiones

Este proyecto trata sobre la utilización de muestras generadas de forma sintética para alimentar modelos de predicción. Para comprobar la eficacia de las mismas, se ha planteado un experimento en el que una red neuronal de regresión cuenta con diferentes números de muestras disponibles. En base a esta premisa, se ha comprobado como, con pocos datos, esta red no es capaz de predecir la salida esperada correctamente, mientras que si se le proporciona un número suficiente de los mismos, logra su objetivo.

Para suplir la falta de datos en los supuestos pertinentes, se han utilizado dos modelos de redes neuronales Generativas, GAN y RBIG, que crean muestras sintéticas similares a las originales. Con estas muestras adicionales, se han vuelto a entrenar las redes regresoras, comprobando como, con sintéticos añadidos, las predicciones resultaban notablemente mejores, sobre todo en casos donde la cantidad de datos reales disponibles es baja.

Se ha comprobado además, que los tiempos de entrenamiento de RBIG son claramente superiores a los de la GAN. Por ello, el modelo RBIG se postula como la mejor opción para abordar este problema.

Se concluye, por tanto, que una generación de datos sintéticos mediante el método RBIG puede ayudar en procesos de regresión, siempre y cuando el número de datos reales sea bajo.

6. Trabajo futuro

Este proyecto ha abordado la problemática de generar muestras sintéticas mediante los modelos de redes generativas GAN y RBIG. Sin embargo, en la actualidad existen una gran cantidad de modelos alternativos que podrían realizar un trabajo similar, o incluso mejorar el mismo. Por ello, sería conveniente explorar otros modelos generativos para la generación de dichas muestras.

En el proyecto, además de los modelos nombrados, se planteó la utilización de un modelo adicional, conocido como Variational Autoencoder (VAE, en adelante). Se propuso dicho modelo como el tercero para los experimentos, pero tras la realización de todos los entrenamientos y pruebas, similares al de los otros dos modelos, fue descartado, ya que no se consiguió obtener la combinación de parámetros y configuraciones oportuna para una buena generación de muestras. Sin embargo, de encontrarse dicha combinación, el modelo VAE podría resultar en una buena incorporación al proyecto.

Por otro lado, en los modelos GAN y RBIG desarrollados sería posible obtener mejoras si se intentaran combinaciones de parámetros diferentes. Por ejemplo, la adición de nuevas capas densas, capas de dropout o de Batch Normalization en diferentes combinaciones podría aportar nuevos resultados.

Cabe destacar que los modelos desarrollados en este proyecto están siendo entrenados siempre con los mismos datos, por lo que es probable que no sean capaces de generalizar a otros datos diferentes. Un punto a comprobar en el futuro sería el funcionamiento de los mismos, obteniendo nuevos conjuntos de datos de IASI, para asegurar su correcto funcionamiento y evitar posibles efectos debidos a la casualidad.

Referencias

[1] Frank Rosenblatt. «The Perceptron - a perceiving and recognizing automation». En: *Report 85–460–1, Cornell Aeronautical Laboratory* (1957).

[2] *Todo lo que Necesitas Saber sobre el Descenso del Gradiente Aplicado a Redes Neuronales*. Sep. de 2019. URL: https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78.

[3] Giuseppe Bonaccorso. *Machine Learning Algorithms: A Reference Guide to Popular Algorithms for Data Science and Machine Learning*. Packt Publishing, 2017. ISBN: 1785889621.

[4] Sebastian Ruder. «An overview of gradient descent optimization algorithms». En: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747.

[5] Ian J. Goodfellow y col. *Generative Adversarial Networks*. 2014. eprint: arXiv:1406.2661.

[6] Jakub Langr. *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning Publications, oct. de 2019. ISBN: 1617295566. URL: https://www.xarg.org/ref/a/1617295566/.

[7] David Foster. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media, jul. de 2019. ISBN: 1492041890.

[8] V Laparra, G Camps-Valls y J Malo. «Iterative Gaussianization: From ICA to Random Rotations». En: *IEEE Transactions on Neural Networks* 22.4 (abr. de 2011), págs. 537-549. DOI: 10.1109/tnn.2011.2106511. URL: https://doi.org/10.1109/tnn.2011.2106511.

[9] Naveen Venkat. *The Curse of Dimensionality: Inside Out*. Sep. de 2018. DOI: 10.13140/RG.2.2.29631.36006.

[10] *METOP satellite in EUMETSAT webpage*. https://www.eumetsat.int/website/home/Satellites/CurrentSatellites/Metop/index.html. Accessed: 2020-05-26.

[11] *IASI instrument in EUMETSAT webpage*. https://www.eumetsat.int/website/home/Satellites/CurrentSatellites/Metop/MetopDesign/IASI/index.htmll. Accessed: 2020-05-26.

[12] Jonathon Shlens. «A Tutorial on Principal Component Analysis». En: *CoRR* abs/1404.1100 (2014). arXiv: 1404.1100. URL: http://arxiv.org/abs/1404.1100.

[13] François Chollet y col. *Keras*. https://keras.io. 2015.

[14] Alec Radford, Luke Metz y Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. eprint: arXiv:1511.06434.

[15] Ian J. Goodfellow. «NIPS 2016 Tutorial: Generative Adversarial Networks». En: *CoRR* abs/1701.00160 (2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[16] Tim Salimans y col. «Improved Techniques for Training GANs». En: *CoRR* abs/1606.03498 (2016). arXiv: 1606.03498. URL: http://arxiv.org/abs/1606.03498.

7. Anexos

En esta sección se mostrarán los anexos que han sido indicados durante el proyecto y que muestran, en general, fragmentos de código referentes a las distintas partes del proyecto.

7.1. GAN-Generador

```
from keras.layers import Input, Dense, Dropout
from keras.layers import BatchNormalization, Activation
from keras.layers.advanced_activations import LeakyReLU
from keras.models import Sequential, Model

def Generator(self):
    model = Sequential()
    model.add(Dense(128,input_dim = self.latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(256))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(self.X_dim))
    model.summary()
    noise = Input(shape=(self.latent_dim,))
    img = model(noise)
    return Model(noise, img)
```

En este ejemplo simplificado, obtenido de la implementación del proyecto, se puede observar como la función Generador crea un modelo secuencial, al que posteriormente se le añaden las capas densas y las funciones de activación y BatchNormalization.

En estas, encontramos dos variables interesantes, alpha y momentum. La primera es un hiperparámetro que se utiliza para controlar el valor al que la función satura las entradas negativas de la red. La segunda se refiere a una técnica utilizada en redes neuronales para mejorar la velocidad de entrenamiento y mejorar la precisión.

7.2. GAN-Discriminador

```
def Discriminator(self):
    model = Sequential()
    model.add(Dense(512,input_shape=(self.X_dim,)))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(256))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(128))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.summary()
    img = Input(shape=(self.X_dim,))
    validity = model(img)
    return Model(img, validity)
```

En este ejemplo simplificado, obtenido de la implementación del proyecto, se puede observar como la función Discriminador crea un modelo secuencial, al que posteriormente se le añaden las capas densas y las funciones de activación.

La entrada de esta red es 'X_dim' que corresponde a las dimensiones de los datos, tanto reales como creados, 51. Las siguientes capas van reduciendo su número de nodos hasta llegar a la capa de salida, que tiene únicamente un nodo, que corresponde a la probabilidad de que el dato sea real o no, entre 0 y 1.

7.3. Complemento Shiny

En este apartado se comentará el complemento Shiny desarrollado para la visualización de los resultados del proyecto. El motivo de este complemento se debe a que Shiny proporciona un entorno interactivo de visualización, por lo que resulta muy eficiente a la hora de mostrar resultados.

Para acceder al recurso, se puede hacer clic en este link, o introduciéndolo en el navegador: https://eduardodavilatfm.shinyapps.io/Shiny/

Una vez dentro, se podrá observar una página de introducción, que cuenta con un menú superior, como se muestra en la imagen siguiente:

Figura 26: Interfaz de Shiny para el proyecto.

En el menú superior, se puede acceder a todas las gráficas mostradas en este documento, de forma unificada e interactiva.

En las opciones se podemos elegir entre dos secciones, gráficas individuales y gráficas de comparación. En la primera podemos encontrar cada una de las gráficas desarrolladas para este proyecto, de forma aislada, incluyendo todos los resultados presentados en este documento.

En la figura 27, a continuación, se puede ver un ejemplo de las gráficas individuales, mientras que en la figura 28 se puede observar un ejemplo de las gráficas comparativas.

En el interior de cada sección aparece siempre, en la parte superior, las gráficas solicitadas y en la parte inferior, un panel donde se pueden indicar las características deseadas, como el modelo a mostrar, las muestras elegidas etc. Adicionalmente, se han añadido instrucciones para su visualización siempre que ha sido necesario, en la parte derecha.

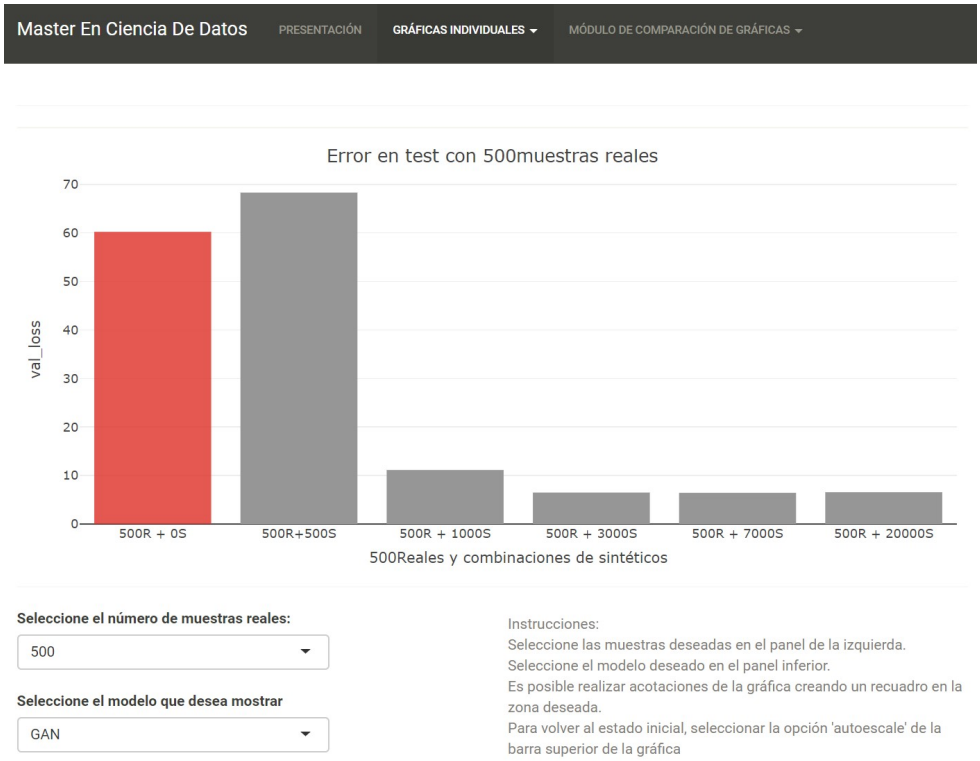

Figura 27: Gráficas individuales en el complemento Shiny

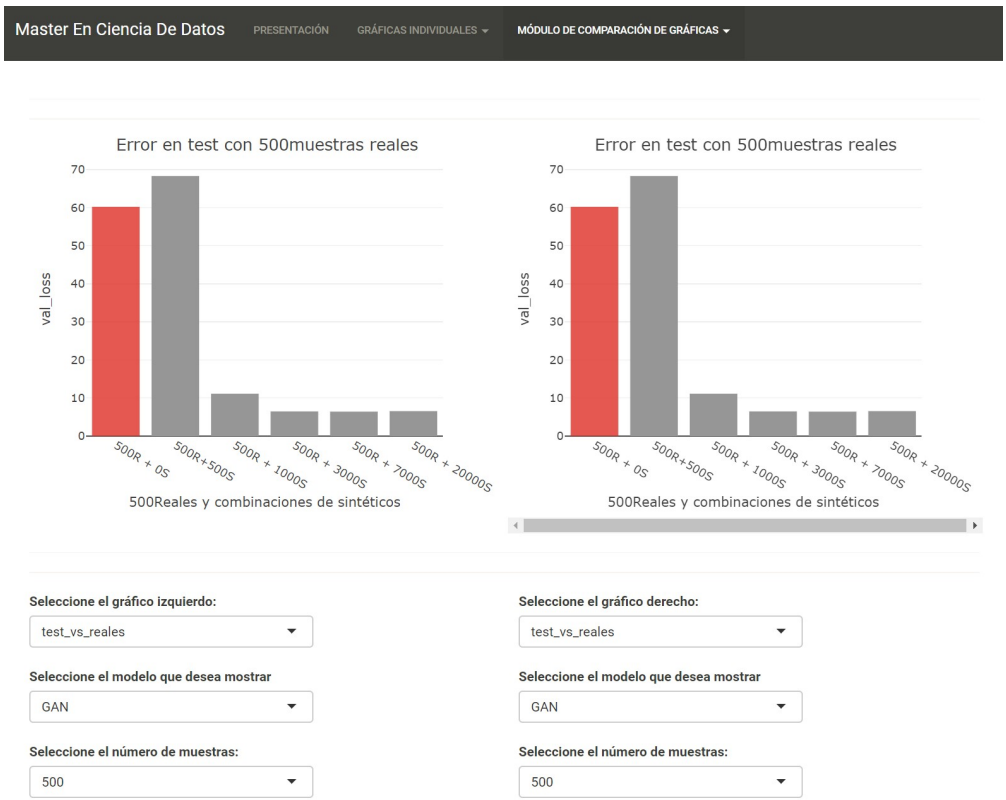

Figura 28: Gráficas comparativas en el complemento Shiny

7.4. Código en Google Colab

Durante el desarrollo del proyecto, parte de la implementación fue realizada en la herramienta online Google Colab. Esta herramienta permite la utilización de un entorno de Python operativo, con acceso a una GPU gratuita, con lo que algunos procesos de entrenamiento de las redes resultan mas rápidos. A continuación, se comparten los diferentes links a las secciones del proyecto desarrolladas en Colab, y además, se comentará brevemente que se puede encontrar en cada una de dichas secciones. Cabe destacar que todos los cuadernos de Python de Google Colab están comentados, para una mejor comprensión, pero no pueden ser ejecutados, ya que requieren de la conexión a la cuenta personal de Google Drive del creador.

- Cuaderno **BigtrainIASI**: En este cuaderno se encuentra todo el desarrollo del análisis que ha llevado a los resultados expuestos durante este documento, tanto para el caso de RBIG como para el de GAN. Todo el código dentro de este cuaderno se encuentra comentado y ordenado, siguiendo una estructura similar para los dos modelos de red. Puede acceder al cuaderno haciendo clic Aquí, o utilizando el link al pie de página en su navegador, abriendo el archivo con Google Colaboratory: [2]

- Cuaderno **EntrenamientoGAN**: En este cuaderno se han realizado los principales entrenamientos de las redes GAN, importando el módulo GAN correspondiente. Puede acceder al cuaderno haciendo clic Aquí, o utilizando el link al pie de página en su navegador, abriendo el archivo con Google Colaboratory: [3]

- Cuaderno **Entrenamiento RBIG**: En este cuaderno, de igual forma al anterior, se han realizado los entrenamientos de la red RBIG, importanto el módulo RBIG directamente desde el GitHub original. Puede acceder al cuaderno haciendo clic Aquí, o utilizando el link al pie de página en su navegador, abriendo el archivo con Google Colaboratory: [4]

[2] BigTrainIASI:**https://colab.research.google.com/drive/1V0w9jUMguagLmaV_vQrQCHvilRdmBB6G?usp=sharing**

[3] EntrenamientoGAN:**https://drive.google.com/file/d/1gjhgha_aHdFl-J6aX0x0s9nOzV4o7qc_/view?usp=sharing**

[4] Entrenamiento RBIG:**https://colab.research.google.com/drive/1QBPa9DpRwY-m-K3M4gIcBqD-xIZ4CFgS?usp=sharing**

7.5. Código adjunto

Además del código desarrollado en Google Colab, se ha trabajado en modo local en tareas que no requerían del uso de la GPU. Estas secciones de la implementación han sido extraídas y se encuentran adjuntas al presente documento. A continuación se comentará los archivos disponibles y que se puede encontrar en cada uno de ellos:

- Código base de la GAN: En el fichero **"Gan_general.py"** se encuentra toda la implementación relacionada con la GAN. Se ha realizado en formato módulo de Python, de modo que para acceder a esta clase, basta con realizar la importación correspondiente desde cualquier otro fichero.
- Código base de RBIG: En el caso de RBIG, la implementación básica se descarga directamente desde el repositorio original: **https://github.com/jejjohnson/rbig.git**
- Código del Shiny desarrollado: En este adjunto se encuentra todo el código utilizado para generar el Shiny, nombrado en el anexo 7.3.