UNIVERSIDAD DE GRANADA

Sistemas Inteligentes para la Gestión en la Empresa

Trabajo sobre Redes Generativas Adversarias

2019-2020

Máster en Ingeniería Informática

Lidia Sánchez Mérida

lidiasm96@correo.ugr.es

Índice

Introducción

La evolución de la Inteligencia Artificial ha sido muy notable durante los últimos años. Han sido diseñadas y entrenadas para resolver una gran variedad de problemas y simular ciertos comportamientos humanos, como la detección de objetos, animales, personas, en imágenes. Sin embargo, la capacidad de crear contenido nuevo es uno de los ámbitos en los que se continúa investigando. Así surgieron las ***redes generativas adversarias*** (GANs) como un sistema de aprendizaje no supervisado que se encuentra compuesto por dos inteligencias artificiales que compiten entre sí para alcanzar una meta. Un ejemplo representativo de la idea se basa en recrear un paisaje realista de forma sintética. Para ello se utilizan dos redes neuronales en las que, cada una juega un papel diferente. De este modo, la primera red comienza generando imágenes de paisajes mientras que la segunda mide la calidad de las mismas. Al final entre ambas redes, el sistema aprende a generar mejores paisajes sintéticos.

Si bien muchos de los ejemplos más populares de aplicaciones de las GANs pertenecen al mundo de la fotografía, estas redes también son capaces de generar otro tipo de contenido, como texto, audio y vídeo.

Más formalmente, las GANs pertenecen a la familia de **modelos generativos**, que intentan predecir cómo son las características de un elemento dada su clasificación. A diferencia de los modelos discriminativos, que estudian las cualidades de las muestras para etiquetarlas, los generativos intentan extraer la distribución de probabilidad con la que generar un conjunto de muestras a partir del estudio de las relaciones entre los elementos y las clases a las que pertenecen. De este modo, dada una entrada analizan las características que les permiten generar una salida similar [2] [3].

Origen

Este modelo fue descrito por *Ian Goodfellow* de la Universidad de Montreal, quien basándose en estudios anteriores propuso una arquitectura formada por dos redes neuronales que eran entrenadas simultáneamente confrontándolas entre sí. Un modelo generativo trataba de obtener la distribución de los datos con la que generar las muestras, mientras que otro discriminativo las analizaba para decidir si eran reales o sintéticas. De este modo, el objetivo consistía en maximizar la probabilidad de que el segundo modelo se equivocase, lo que indicaría que la muestra sintética es muy similar a la real. Este tipo de entrenamiento basado en la competición entre sendas redes, provoca que ambas ajusten sus parámetros mejorando sus capacidades para alcanzar el objetivo final[4].

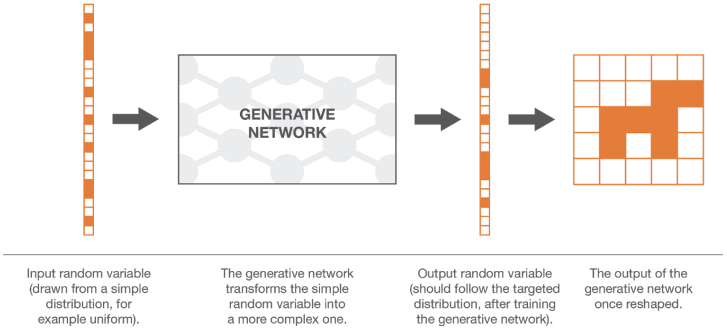
Arquitectura y funcionamiento

El funcionamiento de este modelo, como hemos comentado anteriormente, reside en simular una partida entre dos jugadores. Uno de ellos se conoce como ***generador***, cuyo objetivo consiste en generar muestras sintéticas obteniendo, previamente, la distribución de datos de las muestras de ejemplo proporcionadas. Mientras que el adversario se trata del ***discriminador***, que se encarga de analizar los elementos generados y determinar si pertenecen al grupo de ejemplos o al de los sintéticos.

Ambas redes neuronales son entrenadas de forma diferente puesto que sus objetivos son distintos, y por ende, también disponen de sus propias funciones de coste y parámetros sin posibilidad de influir la una en la otra. Es por ello por lo que *Ian Goodfellow* piensa que este modelo se asemeja más a un juego que a un problema de optimización, ya que en este la solución sería local a sendos espacios de muestras [3].

Generador

El problema de generar una muestra similar a las del conjunto de ejemplo se puede representar como el objetivo de producir un elemento sintético basándose en la distribución de probabilidad de los datos reales. Un caso representativo de este planteamiento reside en generar una imagen sintética de un perro en blanco y negro, aplicando el proceso que podemos visualizar en la siguiente imagen.



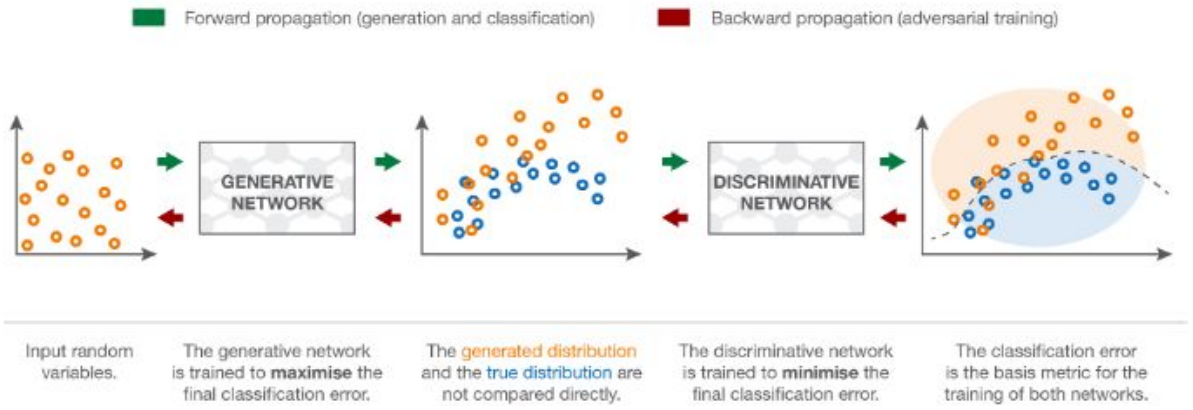
En este caso se representa la imagen como un conjunto de vectores cuyos elementos disponen del rango de valores: blanco o negro. Como primer paso generamos una imagen al azar y se la proporcionamos a la red neuronal. Esta dispone de una ***función de transformación*** en la que aplica una determinada distribución de probabilidad para generar una imagen en cada iteración. Conforme avance el entrenamiento, la distribución generada se asemejará más a la distribución real provocando una mejoría en las imágenes generadas hasta conseguir obtener un conjunto de vectores que representen la imagen de un perro. De este modo el problema se plantea como la generación de una muestra aleatoria siguiendo una determinada distribución de probabilidad [5].

Entrenamiento

Para obtener el mencionado método de transformación con el que el generador vaya mejorando la calidad de las imágenes generadas, se aplica un ***entrenamiento indirecto***, que consiste en forzar que la distribución obtenida sea cada vez más similar a la real. De este modo, la función de transformación del **generador** dispone de una serie de parámetros, que al comienzo se inicializan de forma aleatoria, por lo que las primeras muestras que genere no serán de buena calidad. Para ajustarlos aplica el método del *Gradiente Descendente*, con el que podrá adaptarlos para que la distribución generada sea muy cercana a la real para intentar confundir al discriminador. Por lo tanto, su objetivo es **maximizar el error de clasificación del discriminador**, lo que supondría que la muestra producida es tan similar a las reales que son indiferenciables para esta última red.

De igual modo, el **discriminador** también dispone de una función diferente con su conjunto particular de parámetros que también pueden ser ajustados mediante Gradiente Descendente. Sin embargo su objetivo es diferente y más sencillo, puesto que se trata de identificar si una muestra pertenece al conjunto de entrenamiento real o ha sido generada. Por lo tanto, al contrario que el generador, esta red neuronal debe **minimizar el error de clasificación**. Como se muestra en la siguiente imagen, ambas redes pueden ser

entrenadas conjuntamente utilizando ***backpropagation*** ya que es uno de los métodos de entrenamiento más populares y eficaces para aplicar en redes neuronales y así mejorar los pesos de las mismas propagando el error obtenido hacia las capas anteriores. Asimismo, a través de este método ambas redes, pese a disponer de parámetros independientes, pueden acceder al error de su contrincante con el objetivo de mejorar personalmente e intentar alcanzar su objetivo venciendo a su oponente.



Gracias a este esquema, se puede aplicar una función de coste representada por el algoritmo ***Minimax***, que refleja la situación en la que dos jugadores intentan mejorar a costa de que su adversario pierda, es decir, en el caso de las GANs el coste de la red generadora sería el coste negativo de la discriminadora. Esta es la versión más sencilla de dicho algoritmo en el que la suma de sendos costes resulta en 0. A continuación podemos apreciar la fórmula que representa la función de coste del discriminador (*J(D)*) y la del generador (*J(G)*). El primer factor de la fórmula refleja la información que le proporcionan el conjunto de entrenamiento real y el objetivo de maximizar la probabilidad de clasificarlas como reales. Mientras que el segundo término hace referencia a las muestras sintéticas generadas y a su meta de maximizar la probabilidad de etiquetarlas como falsas. Este objetivo es justo el contrario del generador, el cual pretende minimizar la probabilidad de acierto del discriminador, lo que supondría una calidad de la muestra suficiente como para no ser diferenciable con respecto a una real.

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$
$$J^{(G)} = -J^{(D)}$$

El **problema del método Minimax** es que en la práctica no es aplicable al entrenamiento de las GANs ya que se pueden originar situaciones en las que el discriminador se encuentre tan bien entrenado que clasifique todas las muestras generadas como falsas con una gran seguridad, lo que provocaría que el gradiente del generador desapareciera. La **solución** a este inconveniente consiste en modificar la fórmula del coste del generador *J(G)* a la que visualizamos a continuación. En este caso, en lugar de simplemente cambiar el signo del valor obtenido en la función del coste del discriminador *J(D)*, cambiamos el orden de los términos para generar una nueva función de coste para el generador. Esta, en lugar de minimizar la probabilidad de acierto del discriminador, plantea el objetivo de **maximizar la probabilidad de fallo**. De este modo, aseguramos que ambos jugadores pueden continuar aplicando el método del Gradiente Descendente para continuar mejorando sus parámetros tanto para el ganador como para el perdedor. Al

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D(G(\boldsymbol{z}))$$

realizar estas modificaciones, ya no se puede alcanzar una suma 0 al sumar los costes de ambas funciones [4] [5].

Aplicaciones

Debido a la diversidad de contenido que se le puede proporcionar a estos modelos, el número de ámbitos a los que se pueden aplicar es sumamente amplio.

Ampliar conjuntos de datos

En múltiples problemas, el conjunto de datos no dispone de suficientes muestras como para entrenar modelos competentes para resolverlo, o bien se encuentran muy desbalanceados. Generalmente, una de las mejores soluciones reside en generar un mayor número de muestras. Sin embargo, esto también conlleva el problema asociado de cómo generarlas, ya que dependiendo de lo sofisticado que sea el proceso la calidad será mejor o peor, lo cual tiene un impacto directo en el entrenamiento de los modelos. Por lo tanto, podemos utilizar las GANs para generar muestras sintéticas, como se ha realizado, por ejemplo, sobre el conjunto de dígitos del *MNIST*, el de caras humanas de *Toronto* así como sobre las fotografías del *CIFAR-10*.

Generar nuevos personajes

Otra de las aplicaciones más comunes reside en, a partir de personajes de dibujos animados, producir nuevos caracteres. Con este objetivo se desarrollaron dos proyectos para la serie *Pokémon*, en los que realmente aplicaron dos variantes de las GANs: *WGAN* (*The Wasserstein GAN*) y *DCGAN* (*Deep Convolutional GAN*), que serán explicadas en la sección de variantes.

Traducción de imágenes

En este ámbito el objetivo es proporcionar una descripción de la imagen o el objeto que deseamos generar para que la GAN sea capaz de producir una imagen sintética. Para ello se pueden aportar imágenes cartográficas, imágenes para convertirlas de día a de noche, fotos en blanco y negro para transformarlas a color e incluso dibujos a lápiz para obtener una imagen del objeto real a color.
Sin embargo, también se pueden proporcionar **descripciones textuales** acerca de los objetos, plantas, personas o animales que deseamos que las GANs representen en una fotografía sintética. Asimismo, también existe un determinado modelo denominado ***conditional GANs*** con las que se pueden generar imágenes realistas que represente el mismo significado aportando ima ǵ enes reales o simplemente bocetos sencillos.

Seguridad

Orientado, de nuevo, al ámbito de las imágenes las GANs pueden ser capaces de generar fotografías de frente solo proporcionando imágenes de una persona desde diferentes ángulos. Este mecanismo puede facilitar el **reconocimiento facial** sin disponer de una imagen frontal de la persona [6].

Otra de las técnicas que se pueden aplicar es la **esteganografía** para detectar anomalías en las imágenes y así obtener el contenido que ocultan. Para ello han utilizado un modelo

específico denominado ***SSGAN*** (*Secure Steganography Based On Generative Adversarial Network*) [7].

Edición de imágenes

En primer lugar destacamos la posibilidad de editar propiedades de una imagen, por ejemplo del retrato de una persona, tales como su color del cabello, estilo, expresión facial e incluso la posibilidad de modificar su aspecto acorde a una determinada edad. Asimismo, un modelo particular denominado ***SRGAN***, es capaz de mejorar y aumentar la resolución de las imágenes.

Una segunda técnica consiste en aplicar el modelo GANs para bien completar una parte perdida de la imagen o bien integrar una zona concreta de una fotografía en otra generando una nueva imagen en la que dicha parte se encuentre totalmente integrada. En el primer caso se trata de reconstruir una fotografía a partir de sus características, completando los huecos que pudiese haber, mientras que en el segundo caso se trata de modificar una zona de la imagen añadiendo elementos que no estaban desde un principio [6].

Predicción de vídeos

En este caso, se publicó un artículo en 2016 en el que se utilizaba una variante de las GANs añadiendole una capa convolutiva capaz de diferenciar entre la parte frontal y el fondo de la imagen, con el objetivo de generar pequeños vídeos de hasta un segundo en los que se predice la siguiente escena al *frame* anterior. En los diversos experimentos que realizaron pudieron descubrir que el modelo podía aprender ciertas características del vídeo que le ayudaban a reconocer las acciones que se desenvolvían en el mismo. Esto supondría un gran avance en la compresión de las distintas acciones que se pueden realizar para mejorar las simulaciones [8].

Generación de objetos 3D

Hasta ahora hemos detallado la capacidad de las GANs para generar imágenes en 2D a partir de diverso tipo de contenido. Sin embargo, se ha demostrado que también son capaces de modelar objetos en 3D [6]. Para ello existen dos enfoques:

- En este primero combinan las **GANs con redes convolutivas volumétricas** para, en primer lugar, forzar que el generador obtenga la estructura de los objetos de baja dimensión para no tener que proporcionarle imágenes o modelos de alta calidad para generar el objeto en 3D. Mientras que por otro lado, el discriminador dispone de un descriptor 3D capaz de reconocer las diferentes partes del objeto en 3D aplicando técnicas de aprendizaje no supervisado [9].
- En el segundo modelo utilizan una variante denominada ***PrGANs*** (*Projective GANs*), capaz de entrenar redes neuronales profundas con un conjunto de objetos 3D cuyas proyecciones coincidan posteriormente con las distribuciones de probabilidad de las imágenes en 2D proporcionadas como entrada. Integrando esta proyección no es necesario incluir información 3D acerca del objeto, puesto que solo aportando diferentes puntos de vista de su geometría 2D se pueden generar nuevas imágenes 3D sin aplicar técnicas supervisadas [10].

Ventajas y dificultades

Entre las diferentes ventajas que supone este modelo, destacamos las siguientes:

- Se trata de un método de **aprendizaje no supervisado**, por lo que se ahorra bastante tiempo en clasificar las muestra para poder entrenar las redes neuronales.
- El proceso de **entrenamiento es complementario** para ambos componentes por lo que se pueden entrenar las dos redes conjuntamente aunque dispongan de objetivos diferentes [11].
- Asimismo, el entrenamiento de sendas redes **no requiere de métodos complejos** como las cadenas de Markov, sino que es suficiente con aplicar el método *backpropagation*, que es popularmente conocido y estudiado desde hace tiempo [12].
- De igual modo, su entrenamiento **no necesita de una realimentación constante de datos** sino que, en el caso del generador, se guía en función del Gradiente Descendente que consigue el discriminador, lo que le permite ser más competente estadísticamente [4].
- A diferencia de la gran mayoría de redes neuronales habituales, las GANs son capaces de **generar nuevo contenido** similar a las muestras de ejemplo. Por lo tanto, el número de ámbitos en los que se pueden aplicar son mucho más amplios.
- Disponen de la capacidad de **obtener y aprender la distribución de probabilidad** de los datos para generar muestras sintéticas. Mientras que generalmente el proceso consiste en que dada la distribución de probabilidad, clasificar los elementos en las diferentes categorías existentes, lo que supone un gran avance en el ámbito de la Inteligencia Artificial.
- El discriminador al ser un **clasificador** orientado a un problema, se puede reutilizar de forma independiente del generador [11].

Sin embargo, como toda tecnología también dispone de una serie de limitaciones y dificultades como los que destacamos a continuación:

- Es imprescindible **sincronizar el entrenamiento del generador y el discriminador** para que el modelo obtenido se comporte de forma correcta. Este proceso es bastante complicado y requiere un diseño muy estricto para evitar que ninguna de las redes neuronales avance demasiado y comience a perjudicar el aprendizaje de la otra.
- Relacionado con el punto anterior, es posible que ocurra una situación en la que el **generador comience a producir las mismas muestras** de modo que no sea capaz de mejorar sus creaciones y su aprendizaje se quede paralizado [12]. Este problema se denomina ***mode collapse***. Un ejemplo representativo de este problema surgió al aplicarlo sobre el conjunto de dígitos del MNIST, cuando el generador fue incapaz de generar muestras de los 10 posibles dígitos produciendo solo la del número seis [13].
- Debido a que el entrenamiento se asemeja a un juego en el que si una parte gana, la otra pierde, estas oscilaciones pueden ser indefinidas provocando la **no convergencia** de alguno de los dos algoritmos o de ambos [14].

Variantes

DCGANs

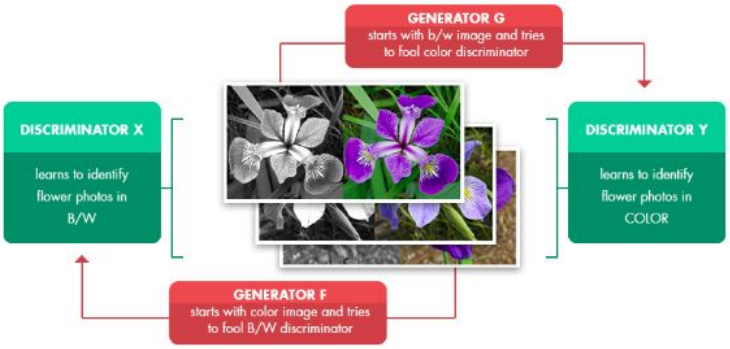
Las *Deep Convolution GANs* son una extensión del modelo original, cuyas capas mayormente son convolutivas. El generador se encuentra compuesto por capas **convolutivas traspuestas** para generar un mayor número de muestras de las que recibe, mientras que el discriminador hace uso de capas convolutivas tradicionales para realizar su entrenamiento orientado a la clasificación. Ambas redes neuronales son entrenadas utilizando minilotes y Gradiente Descendente Estocástico. Asimismo, en sendas se aplica la denominada **normalización *batch*** exceptuando la capa de salida del generador y la de entrada del discriminador [15] [16]. Esta técnica consistente en normalizar las entradas de una capa de modo que tengan una media 0 y desviación típica 1. Entre sus diferentes ventajas se encuentra un entrenamiento más rápido, ya que gracias a la normalización de las entradas la variación de los pesos es menor y por lo tanto se pueden utilizar tasas de aprendizaje más elevadas. De este modo conseguimos aumentar la tasa de aciertos de sendas redes, lo que provocará una convergencia más rápida si se encuentran equilibradas [17].

Una diferencia adicional acerca de las funciones de activación que utilizan ambas redes neuronales reside en que, por un lado el generador utiliza funciones ***ReLU*** en todas las capas exceptuando la de salida en la que utiliza la **tangente hiperbólica**. Mientras que el discriminador utiliza ***LeakyReLU*** [15], con la cual evita un fenómeno denominado *dead ReLU* o *dying ReLU* en el que el proceso de entrenamiento debido a que solo devuelve como resultado cero y por tanto su derivada se sitúa en valores negativos [18].

CycleGANs

Este tipo de redes son ampliamente utilizadas para generar una imagen similar a la de entrada pero modificando su aspecto. Este ámbito se conoce como ***la traducción de imágenes***. De este modo podemos transformar una foto de día a una imagen de noche, paisajes soleados a nublados, fotos en blanco y negro a color, etc.

Para ello la arquitectura consiste en disponer de de un equipo formado por un generador y un discriminador expertos en reconocer los detalles de un determinado aspecto, mientras que existe un segundo grupo con otro generador y discriminador especializados en otra cualidad diferente. En el ejemplo que se visualiza en la figura podemos observar la existencia de un generador que produce imágenes en blanco y negro, mientras que un segundo las genera a color. Del mismo modo, se incluye un discriminador especialista en identificar fotografías en blanco y negro mientras que el otro aprende a reconocer los colores. Por un lado, la primera parte del entrenamiento consiste en que el generador de color comienza a generar imágenes en blanco y negro para luego colorearlas e intentar engañar al discriminador especialista en fotografías a color. Así, este generador aprende a transformar imágenes en blanco y negro a color.



Mientras que en el segundo equipo, el generador comienza a producir fotografías en color para intentar transformarlas a una escala de grises e intentar que el discriminador experto en imágenes en blanco y negro la clasifique como real. De este modo, los generadores comienza a producir imágenes contrarias a la especialización de los discriminadores a los que se encuentra conectados para aprender a transformarlas hacia el tipo contrario, que en este caso se traduce de blanco y negro a color, y viceversa.

De esta idea surgió ***PIX2PIX***, una red neuronal adversaria condicional o ***CGAN*** que es capaz de transformar las características de una muestra para generar un nuevo elemento equivalente pero de aspecto diferente. Este modelo toma como entrada la imagen a traducir y una fotografía de ejemplo para realizar la transformación. El problema reside en que ambas deben tener un perfil muy similar para realizar la correspondencia, lo cual a veces es imposible o requiere un tiempo desorbitado para completar el proceso. Es por ello por lo que plantearon *CycleGAN* como una arquitectura basada en PIX2PIX pero con un entrenamiento basado en dos colecciones de imágenes diferentes, una perteneciente a un tipo, por ejemplo en blanco y negro, y otra a color. Así, el modelo es capaz de traducir una foto en sendos sentidos sin la necesidad de analizar las coincidencias con una imagen objetivo.

Para ello introducen la idea de un ***entrenamiento completo cíclico***, en el cual se transforma la entrada a la salida deseada para luego volver a transformar la muestra generada con el objetivo de comprobar el grado de similaridad con la original. Mediante este proceso se mide la calidad de la traducción para mejorar el modelo [19].

WGANs

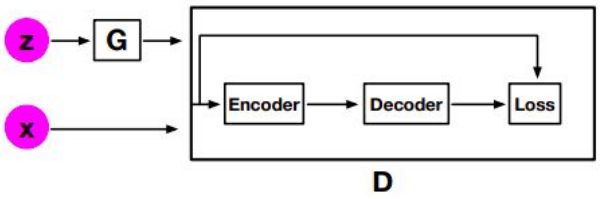
En este modelo se introduce como función de coste una variación de la **distancia de *Wasserstein***, con la que se mide la distancia de dos distribuciones de probabilidad. Para calcularla se reemplaza el discriminador por un componente similar llamado ***crítico***, cuyo objetivo reside en generar una puntuación, en lugar de una probabilidad como el discriminador, que determine el grado de realismo de las muestras proporcionadas como entrada [20] [21].

Asimismo, otra diferencia con respecto al modelo original reside en la función de coste consistente en limitar el valor de los pesos de esta red neuronal a un determinado rango para cumplir la restricción ***Lipschitz***. Esta se basa en añadir a la función de coste un parámetro *K* mayor o igual que cero para resolver un problema denominado la *desinformación del gradiente*, en el cual el clasificador no dispone de información sobre la distribución de probabilidad de las muestras [22].

Una de las ventajas que proporciona esta variante reside en que el uso de la distancia de Wasserstein para determinar el realismo de una muestra, es que impide que el algoritmo oscile fuertemente entre distintos valores, lo cual proporciona una mayor estabilidad al proceso de aprendizaje mediante Gradiente Descendiente [20].

BEGANs

En esta variante denominada *Boundary Equilibrium GAN*, el discriminador dispone de una arquitectura ***autoencoder*** [23], que representa una red neuronal capaz de aprender cómo comprimir y codificar la representación de las muestras para posteriormente realizar los



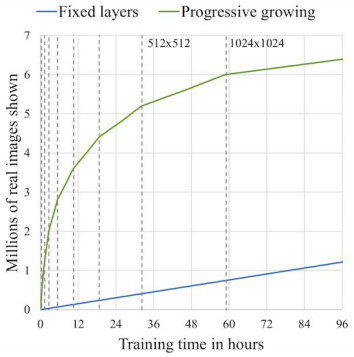
procesos contrarios y construir una muy similar. Para ello utiliza técnicas de aprendizaje no supervisado y suelen ser usadas para reducir la dimensión de un conjunto de datos por su independencia al ruido en los datos [24]. En la siguiente figura podemos observar la arquitectura de este modelo, en la que *x* es la muestra real mientras que *z* es la muestra aleatoria generada al comienzo del entrenamiento del generador.

Asimismo, como se ha mencionado anteriormente, es el discriminador el que presenta una nueva arquitectura así como una función de coste definida por la **distancia de *Wasserstein*** de modo que, en lugar de decidir si una imagen es real o no, sea más flexible indicando el grado de realidad. Estas modificaciones le brinda más tiempo al generador para mejorar, a la vez que limitan el progreso del discriminador para que no se encuentre tan preparado que rechace todas las muestras sintéticas provocando, como hemos explicado anteriormente, el fracaso del entrenamiento del modelo [23].

ProGANs

*Progressive Growing GAN*s es uno de los modelos utilizados por *NVIDIA* para producir imágenes de alta resolución [23]. Para ello aplica un **entrenamiento progresivo** en el que comienza con imágenes de poca resolución para posteriormente añadir más capas a las redes con el fin de incrementar la calidad de las imágenes. De este modo al comienzo del proceso, tanto el generador como el discriminador disponen de muy pocas capas y por lo tanto su entrenamiento es más rápido en las primeras fases, a costa de la baja resolución de las imágenes sintéticas. Conforme avanza el entrenamiento de sendos modelos, se les añaden más capas para duplicar la resolución de la imagen y mejorar su calidad. Para ello se añaden de forma progresiva de modo que la arquitectura se vaya adaptando a ellas, mientras que las capas ya entrenadas pueden seguir ajustando sus pesos. Este proceso continua hasta alcanzar el nivel de resolución especificado.

Entre las diferentes ventajas que proporciona este modelo reside la capacidad de dividir el análisis de una imagen en diferentes iteraciones, de modo que, en las primeras estudia e identifica su estructura mientras que en las últimas continúa captando los detalles para mejorar la resolución. Este tipo de entrenamiento ayuda a mejorar la calidad de las imágenes, puesto que es mucho más estable que el tradicional, y por ello ayuda a evitar que alguna de las dos redes llegue al fin de su entrenamiento, desequilibrando el modelo.



Asimismo, también es más eficiente puesto que los primeros entrenamientos son mucho más veloces debido a que las arquitecturas son más sencillas. Tal es así el impacto sobre la inversión temporal, que en uno de los experimentos de esta fuente, un modelo *ProGAN* resulta de entre 2 a 6 veces más rápido de entrenar que un modelo GAN tradicional. La evolución del entrenamiento en función del tiempo y el número de imágenes de este experimento se puede visualizar en la captura de al lado, en la que destaca la gran

capacidad de las ProGAN para analizar más fotografías que un modelo GAN tradicional [25].

Caso práctico: GAN dígitos MNIST

Como hemos podido observar, existen diversos ámbitos en los que se puede aplicar el modelo GAN o alguna de sus variantes, sin embargo, el más extendido consiste en generar imágenes. Por lo tanto, mi experimento reside en entrenar una red neuronal adversaria que sea capaz de generar dígitos del 0 al 9. Tanto para generar ambas redes como para acceder a la fuente de datos de imágenes, que en mi caso se corresponde con el ***dataset* de dígitos MNIST**, voy a utilizar la librería ***keras***. Como el objetivo es generar imágenes de los dígitos además de entrenar al discriminador para que diferencie entre una imagen del conjunto MNIST y una sintética, solo vamos a hacer uso del conjunto de entrenamiento, que dispone de 60.000 muestras con sus correspondientes etiquetas.

Entre las distintas implementaciones que he probado, la que mejor relación calidad~tiempo tiene se encuentra en esta fuente [26]. Tanto el generador como el discriminador disponen de una capa de entrada de 784 neuronas que recibe los 784 píxeles que componen cada imagen, una oculta de 128 neuronas con una función de activación ***ReLU*** y una de salida con función ***sigmoide*** para calcular la probabilidad de que sea una imagen real o sintética. Los resultados podemos visualizarlos en las siguientes capturas. La primera de la izquierda se corresponde con la primera imagen que el generador ha producido. Como podemos apreciar, no representa ningún dígito puesto que está creada aleatoriamente. Sin embargo, 35.000 iteraciones después podemos comenzar a intuir algunos dígitos en la primera figura de la derecha.

En la segunda fila de imágenes, la primera es el resultado tras 65.000 iteraciones. En este caso los dígitos sí son fácilmente reconocibles, lo que nos indica que el algoritmo se encuentra cerca de su convergencia. Por último, la imagen a la derecha representa el resultado final de la fotografía tras 100.000 iteraciones. A diferencia de la imagen anterior, en este caso los números se encuentran más definidos y con mayor grosor pero en sendas imágenes son reconocibles.

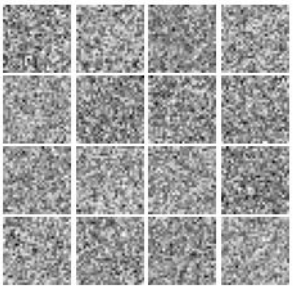


Figura 1. Primera imagen generada.

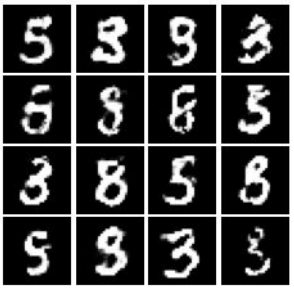
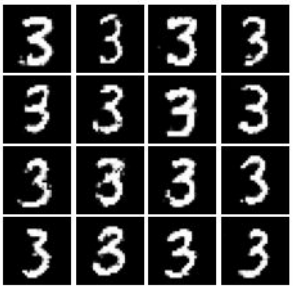


Figura 2. Segunda imagen generada.



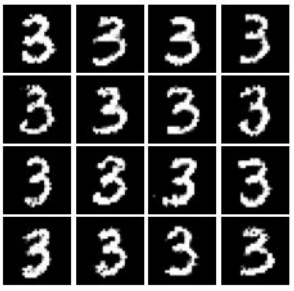


Figura 3. Tercera imagen generada.

Figura 4. Imagen final generada.

Como podemos observar, si bien ha conseguido definir correctamente la imagen, solo aparece el número 3. Esto se debe a uno de los problemas comentados anteriormente denominado *mode collapse,* en el que el generador se convierte en un experto produciendo este dígito que ignora al resto de las clases y no dibuja los otros números.

Caso práctico: DCGAN dígitos MNIST

Una solución a la anterior deficiencia reside en utilizar redes neuronales profundas como el modelo *DCGAN*. Por lo tanto, he querido probarlo para comprobar si los resultados mejoran. Para ello, he encontrado esta implementación [27] consistente en crear una primera red neuronal convolutiva, que se corresponde con el **discriminador**, con una primera capa de entrada con 784 neuronas para recibir los píxeles de la imagen, cuatro capas convolutivas estriadas para realizar *downsampling* y dividir la imagen en varios trozos. Estas capas disponen de una función de activación ***LeakyReLU*** en combinación con *dropout*. Por último se encuentra la capa de salida con una función de activación ***sigmoidal*** para calcular la probabilidad de que la imagen sea real o no.

Por otro lado tenemos el **generador**, que es un modelo DCGAN con capas convolutivas traspuestas para realizar *upsampling* entre las tres primeras capas para aumentar la calidad de las imágenes generadas. Entre las siguientes capas, como hemos explicado anteriormente, se aplica un proceso de **normalización *batch*** con el que estabilizar el entrenamiento. Mientras que las últimas capas disponen de una función de activación ***sigmoidal*** junto con ***dropout*** para poder generar la imagen sintética.

El inconveniente de este modelo es que es **sumamente costoso computacionalmente** llegando hasta invertir dos horas en 500 iteraciones. Por lo tanto, solo lo he podido ejecutar durante 1.000 iteraciones. Sin embargo, como podemos observar en la siguiente comparativa, con 500 y 1.000 iteraciones proporciona mejores resultados que el modelo GAN. Mientras que en este más sencillo aún no podemos distinguir ni siquiera el perfil de un número, puesto que las imágenes están compuestas mayoritariamente de ruido, con el modelo DCGAN sí que se comienzan a apreciar algunas formas.
Asimismo, gracias a su mayor calidad y resolución, podemos incluso distinguir algunos de los números generados como el 0 y el 7. Por lo que podemos determinar que con un modelo más complejo, se consiguen mejores resultados, resolviendo el problema *mode collapse* pero a costa de un mayor tiempo y sobrecarga computacional.

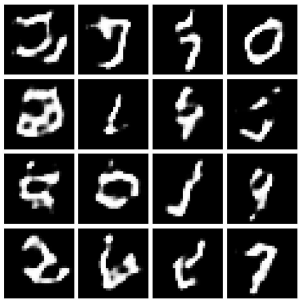


Figura 5. DCGAN 500 iteraciones.

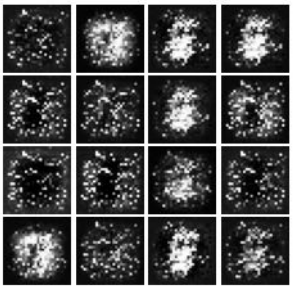


Figura 6. GAN 500 iteraciones.

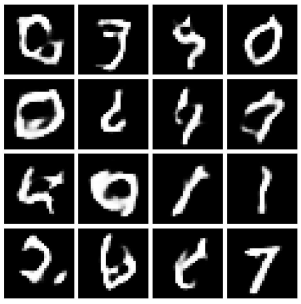


Figura 7. DCGAN 1.000 iteraciones.

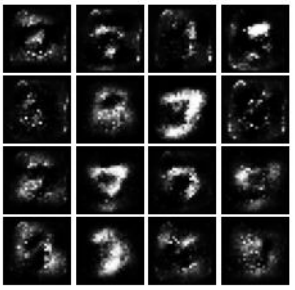


Figura 8. GAN 1.000 iteraciones.

Conclusiones

Como hemos podido observar a lo largo de este trabajo, los modelos GANs son una de los algoritmos más revolucionarios dentro de la Inteligencia Artificial. Han conseguido mejorar la generación de contenido de todo tipo (imágenes, audio, vídeo, etc.) en base a un entrenamiento complementario utilizando dos redes neuronales multicapa. Asimismo, ambas son bastante sencillas tanto en la topología como en las técnicas de entrenamiento, siendo *backpropagation* la más utilizada.

La gran cantidad de aplicaciones que tiene este modelo ha facilitado el desarrollo de múltiples variantes, orientadas a mejorar los diversos objetivos de cada una de ellas como la traducción de imágenes y/o texto, reconocimiento facial para mejorar las identificaciones de personas e incluso predicciones de vídeo.

Sin embargo, como hemos podido leer existen algunas dificultades asociadas al entrenamiento ya que resulta complicado encontrar un equilibrio de modo que ninguna de las dos redes neuronales supere a la otra y el modelo resultante no sea bueno. Este hecho, además, lo he experimentado de forma personal en el primer caso práctico en el que el generador se ha especializado en un tipo de dígito y en la imagen solo aparece el mismo. Con modelos más complejos, este inconveniente y algunos otros se pueden solucionar con arquitecturas más sofisticadas, como las redes convolutivas. Sin embargo, su complejidad y la gran cantidad de datos de entrenamiento pueden provocar que el modelo sea casi imposible de ejecutar con tecnología tradicional, como ha ocurrido en el segundo caso práctico con la DCGAN. No obstante, con ordenadores más potentes se pueden alcanzar resultados casi perfectos, como hemos podido apreciar en este documento, siendo capaces los modelos de generar contenido tan similar al real que hasta una persona podría pensar que realmente lo es.

Bibliografía

1. Lenovo, Marcos Martínez, *Redes generativas adversarias: la IA está aprendiendo de forma exponencial,* 2019, https://www.bloglenovo.es/redes-generativas-adversarias-la-ia-esta-aprendiendo-de-forma-exponencial/
2. Chris Nicholson, *A Beginner's Guide to Generative Adversarial Networks (GANs)*, https://pathmind.com/wiki/generative-adversarial-network-gan

3. Kiran Sudhir, *Generative Adversarial Networks- History and Overview*, 2017, https://towardsdatascience.com/generative-adversarial-networks-history-and-overview-7effbb713545
4. Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, *Generative Adversarial Networks*, 2014, https://arxiv.org/pdf/1406.2661.pdf
5. Joseph Rocca, *Understanding Generative Adversarial Networks (GANs)*, 2019, https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29
6. Jason Brownlee, *18 Impressive Applications of Generative Adversarial Networks (GANs)*, 2019, https://machinelearningmastery.com/impressive-applications-of-generative-adversarial-networks/
7. Faizan Shaikh, *Top 5 Interesting Applications of GANs for Every Machine Learning Enthusiast!,* 2019, https://www.analyticsvidhya.com/blog/2019/04/top-5-interesting-applications-gans-deep-learning/
8. Carl Vondrick, Hamed Pirsiavash, Antonio Torralba*, Generating Videos with Scene Dynamics*, 2016, https://arxiv.org/pdf/1609.02612.pdf
9. Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T. Freeman, Joshua B. Tenenbaum, *Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling*, 2016, https://arxiv.org/pdf/1610.07584.pdf
10. Matheus Gadelha, Subhransu Maji, Rui Wang, *3D Shape Induction from 2D Views of Multiple Objects*, 2016, https://arxiv.org/pdf/1612.05872.pdf
11. *Advantages of GANs*, https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789136678/1/ch01lvl1sec14/advantages-of-gans
12. Easyai, *Generate a confrontation network – Generative Adversarial Networks | GAN*, 2019, https://easyai.tech/en/ai-definition/gan/#yqd
13. Jonathan Hui, *GAN — Why it is so hard to train Generative Adversarial Networks!*, 2018, https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b
14. Hayk Hakobyan, *How GANs can turn AI into a massive force*, 2018, https://www.techinasia.com/talk/gan-turn-ai-into-massive-force
15. PyTorch, DCGAN Tutorial, https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html
16. Jonathan Hui, *GAN — DCGAN (Deep convolutional generative adversarial networks)*, 2018, https://medium.com/@jonathan_hui/gan-dcgan-deep-convolutional-generative-adversarial-networks-df855c438f
17. Álvaro Durán Tovar, *Everything you wish to know about BatchNorm*, 2019, https://medium.com/deeplearningmadeeasy/everything-you-wish-to-know-about-batchnorm-6055e07fdce2#:~:text=What%20is%20BatchNorm%3F,can%20use%20higher%20learning%20rates.
18. Danqing Liu, *A Practical Guide to ReLU*, 2017, https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7
19. Olga Liakhovich, *Learning Image to Image Translation with Generative Adversarial Networks*, 2017,

https://devblogs.microsoft.com/cse/2017/06/12/learning-image-image-translation-cyclegans/
20. Lilian Weng, *From GAN to WGAN,* 2017, https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html#wasserstein-gan-wgan
21. Jonathan Hui, *GAN — Wasserstein GAN & WGAN-GP*, 2018, https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490
22. Universidad de Shanghai, Peking y Standford, Zhiming Zhou, Jiadong Liang, Lantao Yu, Yong Yu, Zhihua Zhang, *Lipschitz Generative Adversarial Nets*, 2019, https://icml.cc/media/Slides/icml/2019/halla(11-14-00)-11-15-10-4628-lipschitz_gener.pdf
23. ZHENGWEI WANG, QI SHE, TOMÁS E. WARD, *Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy*, https://arxiv.org/pdf/1906.01529.pdf
24. Will Badr, *Auto-Encoder: What Is It? And What Is It Used For? (Part 1)*, 2019, https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726#:~:text=Autoencoder%20is%20an%20unsupervised%20artificial,the%20original%20input%20as%20possible.
25. Sarah Wolf, *ProGAN: How NVIDIA Generated Images of Unprecedented Quality*, 2018, https://towardsdatascience.com/progan-how-nvidia-generated-images-of-unprecedented-quality-51c98ec2cbd2
26. Daniele Paliotta, *Introduction to GANs with Python and TensorFlow*, https://stackabuse.com/introduction-to-gans-with-python-and-tensorflow/
27. Rowel Atienza, *GAN by Example using Keras on Tensorflow Backend*, 2017, https://towardsdatascience.com/gan-by-example-using-keras-on-tensorflow-backend-1a6d515a60d0