Universidad de Valladolid

FACULTAD DE CIENCIAS

TRABAJO FIN DE GRADO

Grado en Matemáticas

Técnicas generativas y de reducción de dimensión basadas en transporte óptimo

Autor: Alejandro Rogel Rodríguez
Tutor: Eustasio del Barrio Tellado
Año 2023

Índice

1. Introducción **3**

2. Conceptos previos **5**
2.1. Aprendizaje supervisado. Clasificación binaria 5
2.2. Regresión logística 7
2.3. Aprendizaje no supervisado 9
2.4. Divergencias y distancias 9
2.5. Descenso del gradiente 15
2.6. Redes neuronales y retropropagación 17

3. Redes Generativas Adeversariales (GANs) **20**
3.1. El problema minimax 20
3.2. Relación con la distancia de Jensen-Shannon 20
3.3. El algoritmo GAN 22

4. Distancia de Wasserstein **24**

5. Wasserstein GANs **31**
5.1. Algoritmo WGAN 34

6. Experimentos **36**

Apéndices **42**

A. Teorema de Radon-Nikodym y Teorema de Rademacher **42**

B. Implementaciones **43**

1. Introducción

En el contexto del aprendizaje automático, el objetivo de los modelos generativos es aprender una distribucion de probabilidad $\mathbb{P}_r$, de forma que podamos generar observaciones similares a los de una colección de datos $x_1, ..., x_n$. Para los problemas a los que se enfrenta el aprendizaje automático, es común que $x_i \in \mathcal{X} \in \mathbb{R}^d$, y la dimensión d sea muy grande. La forma clásica de aprender una distribución es aproximando una densidad: definimos una familia paramétrica de densidades $(f_\theta)_{\theta \in \mathbb{R}^d}$ y encontramos aquella que maximiza la verosimilitud de nuestros datos.

Sin embargo, esto no es posible para distribuciones cuyo soporte está contenido en una variedad de dimensión baja, para las cuales no existe una densidad. Esta es una situación común en la mayoría de aplicaciones. Por ejemplo, si $\mathcal{X} = [0, 1]^d$ denota el espacio de las imágenes, y queremos generar nuevas observaciones del conjuto de datos MNIST (que contiene dígitos del 0 al 9 escritos a mano), podemos asumir que el soporte de la distribución de estos datos tiene dimensión mucho menor que d.

Además, aunque pudiéramos aproximar una densidad, el muestreo a partir de ella, por ser en un espacio de dimensión alta, es computacionalmente díficil. Por tanto buscamos modelos que puedan tratar distribuciónes cuyo soporte está en una variedad de baja dimensión, y con los que podamos generar muestras de la distribución sin la necesidad de recurrir a la densidad explícitamente.

Así, en vez de tratar de estimar la densidad de $\mathbb{P}_r$ como en los modelos clásicos, los modelos de variable latente definen una variable aleatoria $Z \in \mathbb{R}^{d'}$ con $d' << d$, y tratan de aproximar $\mathbb{P}_r$ con la ley de una función paramétrica $G_\theta : \mathbb{R}^{d'} \to \mathbb{R}^d$, que genera muestras de una cierta distribución paramétrica P_θ. Las funciones $(G_\theta)_{\theta \in \Theta}$ formarán una colección de transformaciones suficientemente flexibles como para poder adaptarse a una distribución de datos compleja; el parámetro θ lo buscaremos tal que la distribución $\mathbb{P}_\theta$ sea "próxima" o "parecida" (en cierto sentido, que concretaremos) a la distribución $\mathbb{P}_r$.

Las GANs, o redes generativas adversariales, son algoritmos que entran dentro de este contexto. En ellas, el generador G_θ compite con un discriminador D_w: mientras que G_θ aprende a generar objetos más realistas, D_w aprende a distinguirlos de objetos reales: el objetivo es que el generador acabe "engañando" siempre al discriminador, de forma que haya aprendido indirectamente a generar objetos realistas. Este modelo ha tenido éxito en numerosas aplicaciones, pero también es un modelo problemático: el entrenamiento de estas redes es complicado e inestable, y tiene problemas de convergencia.

En este trabajo introduciremos las WGAN: una modificación del algoritmo GAN que usa la distancia de Wasserstein entre probabilidades. La distancia de Wasserstein es más conveniente que otras divergencias o métricas entre probabilidades, como la de Kullback-Leibler o Jensen-Shannon. Esta última es la que esencialmente tratan de minimizar las GANs. Con las WGANs, introduciremos una distancia más débil que la de Jensen-Shannon y mejoraremos el algoritmo GAN aportando una función de pérdida conveniente que mida lo alejada que está la distribución $\mathbb{P}_\theta$ de $\mathbb{P}_r$.

En la sección 1, introduciremos conceptos previos que nos serán necesarios para entender bien estos modelos: el aprendizaje supervisado, las divergencias entre probabilidades, la regresión logística, redes neuronales y el algoritmo del descenso del gradiente. En la sección 2 introduciremos las GANs y expondremos algunos de los problemas que se observan en la práctica. En la

sección 3, presentaremos la distancia de Wasserstein y explicaremos por qué puede ser una buena métrica para resolver algunas de las dificultades de las GANs. En la sección 4, explicaremos cómo tratar la distancia de Wasserstein con las GANs y como entrenar las WGAN. Y por último, en la sección 5 implementamos un ejemplo donde las WGANs funcionan considerablemente mejor que las GANs.

2. Conceptos previos

El aprendizaje automático trata de construir métodos o modelos que hagan uso de datos para aprender a realizar alguna tarea. De esta manera, no tenemos que programar explícitamente un algoritmo para resolver cierto problema, sino que construimos un modelo que se vaya adaptando a la naturaleza de los datos y acabe teniendo un buen desempeño en la tarea correspondiente. Para problemas que son demasiado complicados, un algoritmo tradicional sería demasiado complejo, o directamente imposible de tratar de forma clásica (por ejemplo, el reconocimiento de voz o la deteccion de spam).

Llamaremos conjunto de entrenamiento a la colección de datos que usaremos para que nuestros algoritmos aprendan a resolver un problema. Los problemas del aprendizaje automático se pueden clasificar principalmente en dos categorías:

1. Aprendizaje supervisado, donde los datos del conjunto de entrenamiento llevan asociadas etiquetas. El problema consiste en predecir la etiqueta de un nuevo dato que no la lleve.

2. Aprendizaje no supervisado, donde los datos del conjunto de entrenamiento no incluyen etiquetas y buscamos analizar la estructura de los datos o generar nuevas instancias. Las GANs son algoritmos de aprendizaje no supervisado, donde se nos da una colección de observaciones y se buscan generar otras similares a ellas.

Nos centramos en estas categorías ya que es todo lo que necesitamos para entender las GANs.

2.1. Aprendizaje supervisado. Clasificación binaria

En aprendizaje supervisado, asumimos que el conjunto de entrenamiento son muestras $(x_1, y_1), ..., (x_n, y_n)$ que vienen de variables i.i.d. Como hemos introducido antes, el objetivo de estos problemas es predecir una etiqueta $y \in \mathcal{Y}$, a partir de la observación del atributo $x \in \mathcal{X}$. Si $\mathbb{Y} = \mathbb{R}$, entonces diremos que el problema es de regresión, y si $\mathcal{Y} = \{1, ..., K\}$ con $K \in \mathbb{N}$ entonces el problema es de clasificación.

Para las GANs, nos será útil entender el problema de clasificación binaria, en el que consideraremos $\mathcal{Y} = \{-1, 1\}$ (nos es indiferente considerar $\mathcal{Y} = \{-1, 1\}$ o $\mathcal{Y} = \{0, 1\}$). Queremos obtener una función, o regla, $D : \mathcal{X} \to \mathcal{Y}$ tal que haga buenas predicciones: idealmente querríamos que, si $x \in \mathcal{X}$ es un atributo, e $y \in \mathcal{Y}$ es su etiqueta, entonces si $\hat{y} = D(x)$, $\hat{y} = y$. Es habitual para estos problemas tomar $f : \mathcal{X} \to \mathbb{R}$ y tomar $D(x) = \text{sgn}(f(x))$, el signo de f. Para cuantificar si nos hemos equivocado en una predicción se usa una función de pérdida. Según lo hemos presentado, la función de pérdida que aparece de forma natural es la pérdida 0-1:

$$l(x, y, f) = \mathbb{1}_{yf(x)<0}.$$

Notese que si $yf(x) < 0$ entonces $D(x) = \text{sgn}(f(x)) \neq y$. Usando la notación $f(x) = \hat{y}$, podemos escribir las funciones de pérdida como

$$l(x, y, f) = l(y, \hat{y}),$$

que es su notación mas común.

Sea $\mathcal{D}$ la distribución conjunta (desconocida) de (X, Y). Para conseguir la mejor aproximación que podamos, teniendo una función de pérdida l, y una regla f, consideramos el riesgo asociado a f:

$$\mathcal{R}_{\mathcal{D}}(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}(l(x, y, f)).$$

Nos gustaría encontrar una función con el menor riesgo posible. Veamos que, para el problema de clasificación, la regla de bayes $f_B(x) = \arg\max_{k\in\{-1,1\}} P(Y = k|X = x)$ es, de hecho, la que menos riesgo posible tiene. Enunciaremos el resultado para el caso general en que haya K clases. En ese caso, tendremos $\mathcal{Y} = \{1, ..., K\}$.

Proposición 1. *Para un problema de clasificación con K clases con $\mathcal{D}$ la distribución conjunta de (X, Y), la regla $h_B(x) = arg\max_{k\in\{1,...,K\}} P(Y = k|X = x)$ es tal que $\min_f \mathcal{R}_{\mathcal{D}}(f) = \mathcal{R}_{\mathcal{D}}(h_B)$.*

Demostración. Sea h una regla de clasificación cualquiera. Se tiene que:

$$\begin{aligned} P(h(x) \neq Y|X = x) &= 1 - P(h(x) = Y|X = x) \\ &= 1 - \sum_{k=1}^{K} P(Y = k, h(x) = k|X = x). \end{aligned}$$

Así, para x fijo, podemos escribir:

$$P(h(x) \neq Y|X = x) = 1 - \sum_{k=1}^{K} P(Y = k|X = x) I_{\{h(x)=k\}}.$$

Y también, debido a la definición que hemos dado de h_B:

$$\sum_{k=1}^{K} P(Y = k|X = x) I_{\{h(x)=k\}} \leq \max_{k\in\{1,...,K\}} P(Y = k|X = x) = P(y = h_B(x)|X = x).$$

De donde se sigue que:

$$P(h(x) \neq Y|X = x) = 1 - \sum_{k=1}^{K} P(Y = k|X = x) I_{\{h(x)=k\}} \geq P(y \neq h_B(x)|X = x).$$

Basta concluir observando que esta desigualdad se cumple para toda regla h. □

Notese que no podemos calcular la regla de Bayes directamente porque no conocemos $\mathcal{D}$, pero si la usaremos como referencia más adelante.

La expresión del riesgo es intratable puesto que no conocemos $\mathcal{D}$, y la única información que tenemos está en el conjunto de entrenamiento, las observaciones $(x_1, y_1), ..., (x_n, y_n)$. Para resolver este incoveniente, consideramos el riesgo empírico:

$$\mathcal{R}_{\mathcal{S}}(f) = \frac{1}{n} \Sigma_{i=1}^{n} l(y_i, x_i, f),$$

con el que podemos plantear, dada una clase de funciones hipótesis $\mathcal{F}$:

$$\hat{f} = \arg\min_{f\in\mathcal{F}} \mathcal{R}_{\mathcal{S}}(f).$$

Como la regla de Bayes es la que minimiza el riesgo, tiene sentido tratar de controlar el exceso de riesgo $R(\hat{f}) - R(f_B) \geq 0$, que mide cuanto se desvía la regla $\hat{f}$ de ser óptima. Si $f \in \mathcal{F}$:

$$\begin{aligned} R(\hat{f}) - R(f_B) &= R(\hat{f}) - R_{\mathcal{S}}(\hat{f}) + R_{\mathcal{S}}(\hat{f}) - R(f) + R(f) - R(f_B) \\ &\leq R(\hat{f}) - R_{\mathcal{S}}(\hat{f}) + R_{\mathcal{S}}(f) - R(f) + R(f) - R(f_B) \\ &\leq 2 \sup_{f\in\mathcal{F}} |R_{\mathcal{S}}(f) - R(f)| + \min_{f\in\mathcal{F}} (R(f) - R(f_B)). \end{aligned}$$

El término $2\sup_{f\in\mathcal{F}}|R_{\mathcal{S}}(f)-R(f)|$ es el error de estimación, mientras que $\min_{f\in\mathcal{F}}(R(f)-R(f_B))$ es el error de aproximación. El primero es mayor cuanto más amplia sea $\mathcal{F}$, mientras que el segundo es mayor cuanto menos amplia sea $\mathcal{F}$.

Cuanto más grande sea nuestra clase $\mathcal{F}$, más disminuiremos el riesgo empírico de $\hat{f}$: esto es, hemos encontrado una función que se comporta como queremos en las instancias de entrenamiento. Sin embargo, esto no significa necesariamente que $\hat{f}$ tenga un riesgo pequeño: es posible que se adapte demasiado al conjunto de entrenamiento, dependiendo demasiado de él, y no pudiendo generalizar sus buenos resultados a la distribución verdadera. Esto se conoce como sobreajuste, y es el motivo de un alto error de estimación.

Por otro lado, si la clase de funciones es demasiado pequeña, nos es más difícil encontrar una que se acerque al erorr de la regla de Bayes. El espacio de hipótesis no tiene modelos que puedan aproximar bien la distribución objetivo; esto se conoce como subajuste, y es el motivo de un alto error de estimación.

Por tanto, necesitamos considerar una clase de funciones lo suficientemente grande y flexible como para contener un modelo que aproxime bien la distribución $\mathcal{D}$, pero que no sea demasiado compleja como para adaptarse demasiado al conjunto de entrenamiento y no aprender la estructura de la distribución.

2.2. Regresión logística

Veremos que en el proceso de las GANs entrenaremos a un clasificador (en ese contexto llamado discriminador) cuya función de coste nos recordará mucho a la de la regresión logística. En el contexto de la clasificación binaria, consideramos clasificadores lineales:

$$f_{\beta_0,\beta}(x) = \beta_0 + \beta \cdot x.$$

Según lo visto antes, podemos considerar la regla que minimiza el riesgo empírico asociado a la clase $\mathcal{F} = \{f_{\beta_0,\beta} : \beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^d\}$ y a la función de pérdida 0-1:

$$\min_{f\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{y_i f(x_i)<0}.$$

Este problema de minimización, sin embargo, es intratable: la función de pérdida 0-1 es no convexa, y ni siquiera es continua: esto hace que los algoritmos de optimización usuales basados en el gradiente, que comentaremos más adelante, no se puedan aplicar. Esto es un problema demasiado costoso computacionalmente. Conocer el valor de la función de pérdida para un par β_0, β concreto no nos aporta nada sobre como modificar la solución para mejorarla.

A la hora de clasificar, en vez de considerar una función $f : \mathcal{X} \to \mathbb{R}$ y usar $D(x) = \text{sgn}(f(x))$, podemos tratar de encontrar una función $D : \mathcal{X} \to (0, 1)$ que aproxime $P(y = 1|x)$ y que prediga 1 si $D(x) \geq \frac{1}{2}$ y prediga 0 en el caso contrario. Si queremos seguir usando el modelo lineal, con $z = \beta_0 + \beta \cdot x$, entonces necesitaremos una transformación $g : \mathbb{R} \to [0, 1]$ adecuada. Vamos a intentar explicar los motivos por los que escoger la función logística $g(z) = \frac{1}{1+e^{-z}}$.

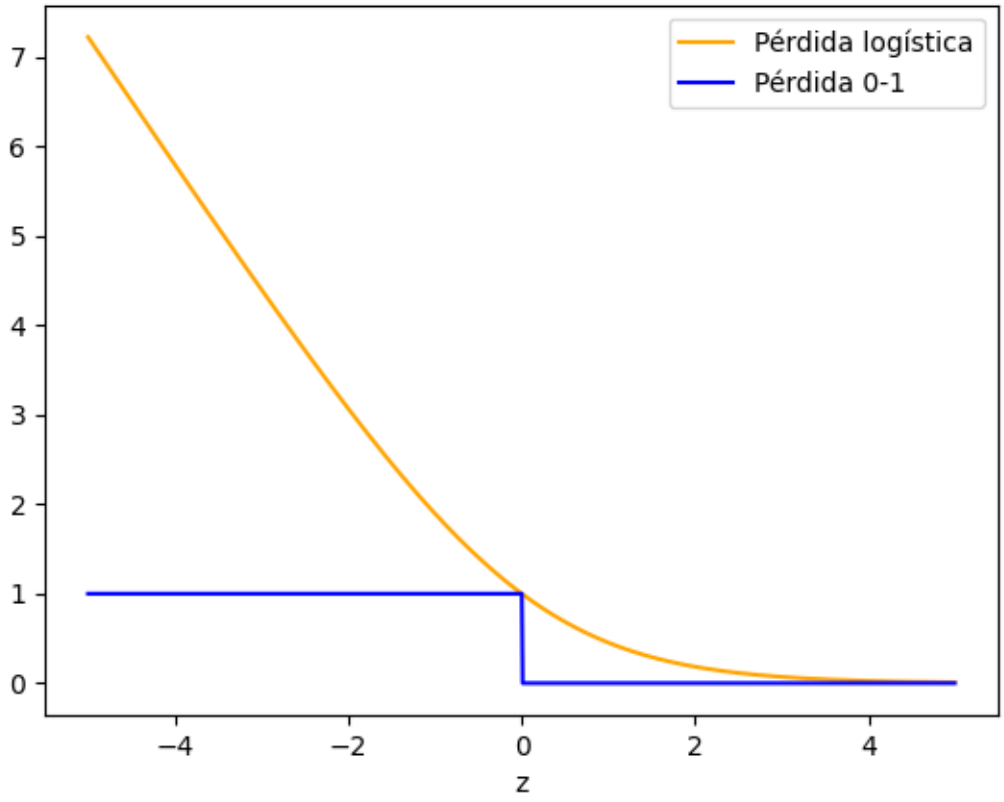
Consideraremos ahora, por comodidad en las ecuaciones y por ser común en la regresión logística, que $\mathcal{Y} = \{0, 1\}$. Sea $z_i = \beta_0 + \beta \cdot x_i$. Consideramos el modelo $g(z) = P_{\beta_0,\beta}(Y = 1|z)$ y

calculamos la log-verosimilitud asociada a él:

$$\begin{aligned} l_{\beta_0,\beta} &= \sum_{i=1}^{n} \log(P_{\beta_0,\beta}(y_i = 1|z_i)) \\ &= \sum_{i:y_i=1} \log(P_{\beta_0,\beta}(y_i = 1|z_i)) + \sum_{i:y_i=0} \log(1 - P_{\beta_0,\beta}(Y = 1|z_i)) \\ &= \sum_{i=1}^{n} (y_i \log(P_{\beta_0,\beta}(Y = 1|z_i)) + (1 - y_i) \log(1 - P_{\beta_0,\beta}(Y = 1|z_i))). \end{aligned} \tag{1}$$

Queremos que $g(z)$ estime la probabilidad $P(Y = 1|z)$. Nos gustaría que cumpliera que $g(0) = \frac{1}{2}$, $g(z) \in [0, 1]$, $\lim_{z\to\infty} g(z) = 1$, $\lim_{z\to-\infty} g(z) = 0$, y que la función sea derivable para aplicar algoritmos basados en el gradiente. La función logística $g(z) = \frac{1}{1+e^{-z}}$ es una buena candidata para modelizar esta probabilidad. Maximizar la verosimilitud es equivalente a minimizar $l(z) = -\log(g(z))$, que es la función de coste de la regresión logística.

Notemos que la función de pérdida $l(z) = -\frac{1}{\log(2)} \log(g(z)) = -\frac{1}{\log(2)} \log(\frac{1}{1+e^{-z}})$ es una función diferenciable y convexa, con lo que es conveniente para aplicar algoritmos de optimización basados en el gradiente. La función $l(z)$ por tanto reemplaza convenientemente a la pérdida 0-1, que, como hemos visto antes, es intratable en la práctica:



Podemos ver la regresión logística desde otro punto de vista: en vez de considerar que los ejemplos del conjunto de entrenamiento vienen de una distribución conjunta (X, Y), podríamos estar ante la situación de que vinieran de dos distribuciones diferentes, P_0, y P_1, y nuestra tarea sería decir si futuras observaciones vienen de P_0 o P_1. Si $x_1, ...x_n$ son observaciones generadas de P_1 y $z_1, ..., z_n$ de P_0, y $D_\theta : \mathcal{X} \to [0, 1]$ es una familia de funciones parametrizadas que modelizan la probabilidad de que una nueva observación x venga de P_1, entonces podemos plantear de

forma similar la log-verosimilitud para obtener:

$$l_n(\theta|x_1,...,x_n,z_1,...,z_n) = \frac{1}{n}\sum_{i=1}^{n}\log(D_\theta(x_i)) + \frac{1}{n}\sum_{i=1}^{n}\log(1-D_\theta(z_i)).$$

Por la Ley de los Grandes Números esto converge casi seguro a:

$$V(D_\theta) = \mathbb{E}_{X\sim P_1}\log(D_\theta(X)) + \mathbb{E}_{Z\sim P_0}\log(1-D_\theta(Z)).$$

Así, podemos intentar resolver el problema $\arg\max_{\theta\in\Theta} V(D_\theta)$. Esta es la parte de clasificación en las GANs. Las funciones D_θ las llamaremos discriminadores, e intentaremos buscar un discriminador que maximice la cantidad $V(D_\theta)$.

2.3. Aprendizaje no supervisado

El aprendizaje supervisado busca analizar, generar, o aprender la estructura de datos sin etiquetas. Se tienen n observaciones $(x_1,...,x_n)$ i.i.d de una distribución $\mathbb{P}_0$, con $x_i \in \mathcal{X}$, y el objetivo es encontrar propiedades de la distribución $\mathbb{P}_0$. La dimensión de $\mathcal{X}$ suele ser muy alta, más que en los problemas de aprendizaje supervisado.

Por ejemplo, los métodos cluster tratan de agrupar los datos los datos en diferentes grupos o clases, haciendo que objetos en una clase sean más similares entre ellos (en algún sentido) que con respecto a las otras clases. El algoritmo de las k-medias es un ejemplo de un método cluster. Por otro lado, los algoritmos de reducción de la dimensionalidad tratan de buscar una representación de un conjunto de datos contenido en un espacio de dimensión alta en uno de dimensión baja, con el objetivo de visualizar los datos, reducir la capacidad computacional necesaria para tratarlos, encontrar asociaciones entre variables, o como forma de simplificar los datos para otro modelo. Un ejemplo de algoritmo de reducción de la dimensionalidad es el PCA.

Las GANs son otro ejemplo de aprendizaje no supervisado, con objetivo de generar instancias similares a las que vienen de $\mathbb{P}_0$. Los modelos con este objetivo suelen hacer uso de una forma de medir lo alejadas que están, en algún sentido, dos distribuciones de probabilidad. Por ello introducimos algunas divergencias y distancias entre probabilidades.

2.4. Divergencias y distancias

Introduzcamos la divergencia de Kullback-Leibler y veamos que la función de pérdida que dimos antes para la regresión logística está relacionada con ella. Sea $Prob(\mathcal{X})$ la familia de probabilidades definidas en el espacio de medida$(\mathcal{X},\mathcal{A})$. Supongamos que $P,Q \in Prob(\mathcal{X})$ y que $P << Q$. Denotamos por $\frac{dP}{dQ}$ a la derivada de Radon-Nikodym de P respecto a Q Se define la divergencia de Kullback-Leibler como:

$$KL(P||Q) = \int_{\mathcal{X}} \log(\frac{dP}{dQ})dP,$$

donde integramos con respecto a la medida P. Si P no es absolutamente continua con respecto a Q, entonces se define $KL(P||Q) = \infty$.

Proposición 2. *Si P,Q son dsitribuciones de probabilidad definidas en $\mathcal{X}$, se tiene que:*

1. $KL(P||Q) = 0 \iff P = Q$.
2. $KL(P||Q) \geq 0$.

Demostración. Sabemos que $\frac{dP}{dQ} = 1 \iff P = Q$. Por lo tanto, si $P = Q$:

$$KL(P||Q) = \int_{\mathcal{X}} \log(\frac{dP}{dQ})dP = \int_{\mathcal{X}} \log(1)dP = 0.$$

Para la otra implicación, notemos que considerando $\nu = \frac{P+Q}{2}$, $P << \nu$ y $Q << \nu$ y por lo tanto podemos escribir:

$$KL(P||Q) = \int_{\mathcal{X}} \log(\frac{\frac{dP}{d\nu}}{\frac{dQ}{d\nu}})\frac{dP}{d\nu}d\nu.$$

Teniendo esto en cuenta, por ser la función $\log(x)$ cóncava y aplicando la desigualdad de Jensen a la función (convexa) $-\log(x)$, nos queda que:

$$\begin{aligned} 0 = \int_{\mathcal{X}} \log(\frac{dP}{dQ})dP &= -\int_{\mathcal{X}} \log(\frac{\frac{dQ}{d\nu}}{\frac{dP}{d\nu}})\frac{dP}{d\nu}d\nu \\ &\geq -\log(\int_{\mathcal{X}} \frac{\frac{dQ}{d\nu}}{\frac{dP}{d\nu}}\frac{dP}{d\nu}d\nu) \\ &= -\log(\int_{\mathcal{X}} dQ) \\ &= 0. \end{aligned}$$

Notemos que con esto hemos demostrado ya que $KL(P||Q) \geq 0$. Como se da la igualdad en la desigualdad de Jensen, y la función a la que la estamos aplicando no es lineal, se deduce que $\frac{dP}{dQ} = a$, $a \in \mathbb{R}$. Ahora, usando el Teorema de Radon-Nikodym:

$$1 = P(\mathcal{X}) = \int_{\mathcal{X}} a \; dQ = a.$$

Luego $\frac{dP}{dQ} = 1$ y ,por tanto, $P = Q$. □

La divergencia de Kullback-Leibler está relacionada con el estimador máximo verosímil. Si la probabilidad P que estamos tratando de estimar tiene densidad f y tenemos un modelo paramétrico de la densidad f_θ, la log-verosimilitud es:

$$\arg\max_\theta \sum_{i=1}^{n} \log(f_\theta(x_i)) = -\frac{1}{n} \arg\max_\theta \sum_{i=1}^{n} \log(\frac{f_\theta(x_i)}{f(x_i)}).$$

Y por la ley de los grandes números (recordemos que la función de densidad no es más que la derivada de Radon-Nikodym):

$$-\frac{1}{n}\sum_{i=1}^{n} \log(\frac{f_\theta(x_i)}{f(x_i)}) \to_{\text{c.s}} \mathbb{E}_{x\sim P}[-\log(\frac{f_\theta(x_i)}{f(x_i)})] = -KL(P||P_\theta).$$

Luego maximimizar la verosimilitud es equivalente a minimizar la divergencia de Kullback-Leibler. Si volvemos al contexto de clasificación binaria, podemos considerar que las etiquetas y_i de cada elemento del conjunto de entrenamiento definen una probabilidad. (Tomamos aquí $\mathcal{Y} = \{0, 1\}$). En este caso, si P_i es dicha probabilidad, y $g_\theta : \mathcal{X} \to \{0, 1\}$ es un modelo parametrizado de la probabilidad $P(Y = 1|X)$:

$$KL(P_i||G_\theta) = y_i \log(y_i) + (1 - y_i) \log(1 - y_i) - y_i \log(g_\theta(x_i)) - (1 - y_i) \log(1 - g_\theta(x_i)),$$

donde G_θ es la medida de probabilidad definida por g_θ Si minimizamos con respecto a θ:

$$\arg\min_\theta KL(P_i||G_\theta) = -\arg\min_\theta y_i \log(g_\theta(x_i)) + (1-y_i)\log(1-g_\theta(x_i)).$$

Esta es la función de pérdida para cada elemento usando la divergencia de Kullback-Leibler. Si minimizamos ahora la pérdida media (que es lo habitual en problemas de aprendizaje) nos queda el problema de minimización:

$$\arg\min_\theta -\frac{1}{n}\sum_{i=1}^{n} y_i \log(g_\theta(x_i)) + (1-y_i)\log(1-g_\theta(x_i)).$$

Que es equivalente a maximizar la ecuación (1) a la que hemos llegado antes con el estimador máximo verosímil.

Con estas observaciones hemos justificado por qué la divergencia KL es usada como medida de discrepancia entre probabilidades. Sin embargo, la divergencia de Kullback-Leibler presenta algunos problemas. En primer lugar, no es una distancia: ni es simétrica ni cumple la desigualdad triangular. Por otro lado, y quizá más relacionado con las GANs: si el soporte de P y Q tienen intersección vacía, entonces $KL(P||Q) = \infty$. Este va a ser el caso en las GANs, y es algo que dificulta su entrenamiento. Para intentar solucionar estos problemas, introducimos la distancia de Jensen-Shannon. Sea $P_m = \frac{P+Q}{2}$. Entonces:

$$JS(P,Q) = \frac{1}{2}KL(P||P_m) + \frac{1}{2}KL(Q||P_m).$$

La expresión $\sqrt{JS(P,Q)}$ si que define una distancia [1]. Sin embargo, si P y Q tienen soporte disjunto entonces $\frac{dP}{d(P_m)} = 2\chi_{sop(P)}$ ya que si $A \in \mathcal{A}$:

$$\int_A 2\chi_{sop(P)} dP_m = 2P_m(A \cap sop(P)) = 2\frac{P+Q}{2}(A \cap sop(P)) = 2\frac{P(A)}{2} = P(A).$$

Por lo tanto se tendrá que:

$$KL(P||P_m) = \int \log(\frac{dP}{dP_m})dP = \int_{sop(P)} \log(2)dP = \log(2).$$

De forma análoga se ve que $KL(Q||P_m) = \log(2)$. Así:

$$JS(P,Q) = \log(2).$$

De hecho, se tiene el siguiente resultado:

Proposición 3. *Si $P, Q \in Prob(\mathcal{X})$, $JS(P,Q) \leq \log(2)$.*

Demostración. Se tiene que, $\forall A \in \mathcal{A}$:

$$\int_A \frac{dP}{d(P+Q)} d(P+Q) = P(A) \leq P(A) + Q(A) = \int_A d(P+Q).$$

Como esto se cumple $\forall A \in \mathcal{A}$, tendremos que $\frac{dP}{d(P+Q)} \leq 1$. Por tanto, teniendo en cuenta que $\frac{dP}{d\frac{P+Q}{2}} = 2\frac{dP}{d(P+Q)}$:

$$\begin{aligned} JS(P,Q) &= \frac{1}{2}(\int_{\mathcal{X}} \log(2\frac{dP}{d(P+Q)})dP + \int_{\mathcal{X}} \log(2\frac{dQ}{d(P+Q)})dQ) \\ &\leq \frac{1}{2}(\int_{\mathcal{X}} \log(2)dP + \int_{\mathcal{X}} \log(2)dQ) \\ &\leq \log(2). \end{aligned}$$

□

Aunque ahora el valor no es infinito, si P y Q tienen soporte disjunto, $JS(P,Q)$ es una constante. Si $Q = R_\theta$ es una distribución parametrizada por $\theta \in \mathbb{R}^p$, y existe un abierto U el los soportes de Q y R_θ tienen intersección disjunta $\forall \theta \in U$ entonces $\nabla_\theta(JS(P,R_\theta)) = 0$ en U. Esto hará que, en el caso de soporte disjunto, los algoritmos que usaremos para optimizar la función de pérdida, basados en el gradiente, funcionen muy lento o sean impracticables. Las GANs optimizan la distancia de Jensen-Shannon, y aquí estamos empezando a intuir los motivos por los que introduciremos una distancia con mejores propiedades.

Por otro lado, introducimos la distancia en variación total:

$$\delta(P,Q) = \sup_{A\in\mathcal{A}} |P(A) - Q(A)|.$$

La comprobación de que es una distancia es trivial. La distancia en variación total se relaciona con la divergencia de Kullback-Leibler de la siguiente manera:

Proposición 4 (Desigualdad de Pinsker)**.** *Si P y Q son probabilidades definidas en $\mathcal{X}$, entonces:*

$$\delta(P,Q) \leq \sqrt{\frac{1}{2}KL(P||Q)}.$$

Una prueba del resultado se puede ver en [2]. En el siguiente resultado vemos cómo se relacionan las conceptos que hemos introducido.

Proposición 5. *Sea $\mathbb{P}$ una distribución definida en $\mathcal{X}$ y $(\mathbb{P}_n)_{n=1}^{\infty}$ una sucesión de distribuciones en X. Entonces:*

1. $\delta(\mathbb{P}_n,\mathbb{P}) \to 0 \iff JS(\mathbb{P}_n,\mathbb{P}) \to 0$
2. *Si $KL(\mathbb{P}_n||\mathbb{P}) \to 0$ o $KL(\mathbb{P}||\mathbb{P}_n)$, entonces $JS(\mathbb{P}_n,\mathbb{P}) \to 0$.*
3. *Si $\delta(\mathbb{P}_n,\mathbb{P}) \to 0$, entonces $\mathbb{P}_n \to_d \mathbb{P}$, donde d denota la convergencia débil o en distribución.*

Demostración.

1. Supongamos primero que $JS(\mathbb{P}_n,\mathbb{P}) \to 0$. Si $\mathbb{P}_m = \frac{\mathbb{P}_n+\mathbb{P}}{2}$, se tiene que:

$$\begin{aligned}\delta(\mathbb{P}_n,\mathbb{P}_m) &= \sup_{A\in\mathcal{A}} \mathbb{P}_n(A) - \frac{\mathbb{P}(A)+\mathbb{P}_n(A)}{2} \\ &= \sup_{A\in\mathcal{A}} \frac{\mathbb{P}_n(A)-\mathbb{P}(A)}{2} \\ &= \frac{1}{2}\delta(\mathbb{P}_n,\mathbb{P}).\end{aligned}$$

   Razonando de la misma manera se ve que $\delta(\mathbb{P},\mathbb{P}_m) = \frac{1}{2}\delta(\mathbb{P}_n,\mathbb{P})$. Usando esto y aplicando la desigualdad de Pinsker:

$$\begin{aligned}JS(\mathbb{P}_n,\mathbb{P}) &= \sqrt{\frac{1}{2}KL(\mathbb{P}_n||\mathbb{P}_m) + \frac{1}{2}KL(\mathbb{P}||\mathbb{P}_m)} \\ &\geq \sqrt{\delta(\mathbb{P}_n,\mathbb{P}_m)^2 + \delta(\mathbb{P},\mathbb{P}_m)^2} \\ &= \sqrt{\frac{1}{2}\delta(\mathbb{P}_n,\mathbb{P})^2} \\ &= \frac{1}{\sqrt{2}}\delta(\mathbb{P}_n,\mathbb{P}).\end{aligned}$$

Luego si $JS(\mathbb{P}_n, \mathbb{P}) \to 0$, $\delta(\mathbb{P}_n, \mathbb{P}) \to 0$.

Ahora veamos la otra implicación. Como hemos visto antes, se tiene que:

$$\delta(\mathbb{P}_m, \mathbb{P}_n) = \frac{1}{2}\delta(\mathbb{P}, \mathbb{P}_n) \leq \delta(\mathbb{P}, \mathbb{P}_n).$$

Por lo que basta ver que $\delta(\mathbb{P}_m, \mathbb{P}_n) \to 0$. Observemos que si $\mathbb{P}_m(A) = 0$ con $A \in \mathcal{A}$, entonces $\mathbb{P}_n(A) = 0$. Por tanto, podemos considerar la derivada de Radon-Nykodim de $\mathbb{P}_n$ con respecto a $\mathbb{P}_m$, $f_n = \frac{d\mathbb{P}_n}{d\mathbb{P}_m}$. Observemos también que:

$$\mathbb{P}_n(A) = 2\mathbb{P}_m(A) - \mathbb{P}(A) \leq 2\mathbb{P}_m(A) \tag{2}$$

para todo conjunto A de Borel. Si consideramos $A = \{f_n > 3\}$, entonces, por ser f_n la derivada de Radon-Nykodim:

$$\mathbb{P}_n(A) = \int_A f_n d\mathbb{P}_m \geq 3\mathbb{P}_m(A).$$

Teniendo en cuenta la ecuación (2), nos queda que $\mathbb{P}_m(A) = 0$. Esto quiere decir que $f_n \leq 3$ casi siempre con respecto a la medida $\mathbb{P}_m$ (y por tanto también con respecto a las medidas $\mathbb{P}$ y $\mathbb{P}_n$).

Fijemos ahora $\epsilon > 0$ y definimos los conjuntos $A_n = \{f_n > 1 + \epsilon\}$.Así,

$$\mathbb{P}_n(A_n) = \int_{A_n} f_n d\mathbb{P}_m \geq (1 + \epsilon)\mathbb{P}_m(A_n),$$

de donde se sigue que:

$$\begin{aligned}
\epsilon\mathbb{P}_m(A_n) &\leq \mathbb{P}_n(A_n) - \mathbb{P}_m(A_n) \\
&\leq |\mathbb{P}_n(A_n) - \mathbb{P}_m(A_n)| \\
&\leq \delta(\mathbb{P}_n, \mathbb{P}_m).
\end{aligned}$$

Además,

$$\begin{aligned}
\mathbb{P}_n(A_n) &= \mathbb{P}_m(A_n) + \mathbb{P}_n(A_n) - \mathbb{P}_m(A_n) \\
&\leq \mathbb{P}_m(A_n) + |\mathbb{P}_n(A_n) - \mathbb{P}_m(A_n)| \\
&\leq \frac{1}{\epsilon}\delta(\mathbb{P}_n, \mathbb{P}_m) + \delta(\mathbb{P}_n, \mathbb{P}_m) \\
&\leq (\frac{1}{\epsilon} + 1)\delta(\mathbb{P}_n, \mathbb{P}).
\end{aligned}$$

Ahora, teniendo en cuenta la definición de A_n:

$$\begin{aligned}
KL(\mathbb{P}_n||\mathbb{P}_m) &= \int_X \log(f_n) d\mathbb{P}_n \\
&= \int_{X\setminus A_n} \log(f_n)\mathbb{P}_n + \int_{A_n} \log(f_n)\mathbb{P}_n \\
&\leq \log(1 + \epsilon) + \log(3)\mathbb{P}_n(A_n) \\
&\leq \log(1 + \epsilon) + \log(3)(\frac{1}{\epsilon} + 1)\delta(\mathbb{P}_n, \mathbb{P}).
\end{aligned}$$

Ahora tomamos límites superiores (notese que como estamos suponiendo que $\delta(\mathbb{P}_n, \mathbb{P}) \to 0$, entonces $\limsup \delta(\mathbb{P}_n, \mathbb{P}) = 0$) y obtenemos que:

$$0 \leq \limsup KL(\mathbb{P}_n||\mathbb{P}_m) \leq \log(1+\epsilon) \ , \forall \epsilon > 0.$$

Por lo tanto tenemos que $KL(\mathbb{P}_n||\mathbb{P}_m) \to 0$. De forma análoga (los mismos cálculos) se tiene que $KL(\mathbb{P}||\mathbb{P}_m) \to 0$. Luego $JS(\mathbb{P}_n, \mathbb{P}) \to 0$.

2. Por la desigualdad de Pinsker, si $KL(\mathbb{P}_n||\mathbb{P}) \to 0$,

$$\delta(\mathbb{P}_n, \mathbb{P}) \leq \sqrt{\frac{1}{2} KL(\mathbb{P}_n||\mathbb{P})} \to 0.$$

El otro caso se demuestra de la misma forma.

3. Supongamos que $\delta(\mathbb{P}_n, \mathbb{P}) \to 0$. Tenemos que ver que si $f : \mathcal{X} \to \mathbb{R}$ es continua y acotada (en este caso toda función continua es acotada) entonces:

$$\int_{\mathcal{X}} f d\mathbb{P}_n \to \int_{\mathcal{X}} f d\mathbb{P}.$$

Sea f una función continua y acotada. Dado $\epsilon > 0$, por ser f acotada, debido a la densidad de las funciones simples en L^∞ (cuya demostración puede verse en [3]) existe una función simple $\phi = \sum_{k=1}^m a_k \chi_{A_k}$, tal que:

$$\|f - \phi\|_\infty \leq \epsilon.$$

De esta manera:

$$\begin{aligned} |\int_{\mathcal{X}} f d\mathbb{P}_n - \int_{\mathcal{X}} f d\mathbb{P}| &= |\int_{\mathcal{X}} f - \phi d\mathbb{P}_n - \int_{\mathcal{X}} f - \phi d\mathbb{P} + \int_{\mathcal{X}} \phi d\mathbb{P}_n - \int_{\mathcal{X}} \phi d\mathbb{P}| \\ &\leq 2\epsilon + |\int_{\mathcal{X}} \phi d(\mathbb{P}_n - \mathbb{P})| \\ &= 2\epsilon + |\sum_{k=1}^m a_k(\mathbb{P}_n(A_k) - \mathbb{P}(A_k)) \\ &\leq 2\epsilon + \sum_{k=1}^m |a_k||\mathbb{P}_n(A_k) - \mathbb{P}(A_k)|. \end{aligned} \tag{3}$$

Ahora puesto que $\delta(\mathbb{P}_n, \mathbb{P}) \to 0$, existe un $n_0 \in \mathbb{N}$ tal que $\forall k \in \{1, ..., m\}, \forall n \in \mathbb{N}, n \geq n_0$:

$$|\mathbb{P}_n(A_k) - \mathbb{P}(A_k)| \leq \epsilon.$$

Y juntándolo con (3), queda que:

$$|\int_{\mathcal{X}} f d\mathbb{P}_n - \int_{\mathcal{X}} f d\mathbb{P}| \leq \epsilon(2 + \sum_{k=1}^m |a_k|).$$

Luego $\int_{\mathcal{X}} f d\mathbb{P}_n \to \int_{\mathcal{X}} f d\mathbb{P}$, como queríamos.

□

2.5. Descenso del gradiente

Ya hemos visto que, en aprendizaje automático, los problemas de optimización forman una parte importantísima al plantear un algoritmo. Muchas veces necesitamos resolver problemas como:

$$\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n} l(G_\theta(x_i)), \tag{4}$$

donde $x_1, ..., x_n$ son las muestras de entrenamiento, l es la función de pérdida y $(G_\theta)_{\theta\in\Theta}$ denota el conjunto de modelos (hipótesis), parametrizados por θ. Ya hemos hecho alusión a la conveniencia de que la función de pérdida diferenciable y convexa para aplicar algoritmos de optimización basados en el gradiente.

Para una función $f : \mathbb{R}^d \to \mathbb{R}$ diferenciable, sabemos que el gradiente ∇f es la dirección de máximo crecimiento de la función. Para la función $-f$ la dirección de máximo crecimiento es $-\nabla f$; si tenemos en cuenta que cuanto más crece $-f$ más decrece f, llegamos a que $-\nabla f$ es la dirección de máximo decrecimiento de f. Esto motiva el algoritmo del descenso de gradiente, que trata de minimizar f dando pasos en la dirección $-\nabla f$. Supongamos que partimos del punto x_0, entonces se itera:

$$x_{n+1} = x_n - \gamma_n \nabla f(x_n),$$

donde la tasa de aprendizaje γ_n se puede tratar de elegir convenientemente para cada iteración. Para asegurar que el algoritmo converja al mínimo global, necesitamos que la función sea convexa, que es el caso en el que solo hay un mínimo local. En ese caso, y si suponemos que f cumple una condición adicional de suavidad, se tiene el siguiente resultado, que está desarrollado, junto con otros relativos a la velocidad de convergencia del descenso del gradiente, en [4]:

Proposición 6. *Sea f es una función en $\mathbb{R}^d$ con valores reales, convexa, y β-suave, esto es, existe $\beta > 0$ tal que:*

$$\|\nabla f(x) - \nabla f(y)\| \leq \beta\|x - y\|.$$

Sea $x^* \in arg\min f$. Entonces el algoritmo de descenso del gradiente con $\gamma_n = \frac{1}{\beta}$ $\forall n \in \mathbb{N}$ cumple que:

$$f(x_n) - f(x^*) \leq \frac{\beta\|x_0 - x^*\|^2}{2n}.$$

En particular, el algoritmo del descenso de gradiente converge cuando $n \to \infty$.

Demostración. Probemos primero que si $x, y \in \mathbb{R}^d$, entonces por ser f β-suave se tiene que:

$$|f(x) - f(y) - \nabla f(y)^T(x - y)| \leq \frac{\beta}{2}\|x - y\|^2. \tag{5}$$

Si $x, y \in \mathbb{R}^d$, notemos que

$$\int_0^1 \nabla f(y + t(x - y))(x - y)dt = f(x + t(x - y))\Big|_0^1 = f(x) - f(y).$$

Entonces:

$$\begin{aligned}|f(x)-f(y)-\nabla f(y)^T(x-y)| &= |\int_0^1 \nabla f(y+t(x-y))^T(x-y)-\nabla f(y)^T(x-y)dt| \\ &= |\int_0^1 (\nabla f(y+t(x-y))^T-\nabla f(y)^T)(x-y)dt| \\ &\leq \int_0^1 \|\nabla f(y+t(x-y))-\nabla f(y)\|dt\int_0^1 \|x-y\|dt \\ &\leq \beta t\|x-y\|dt\|x-y\| \\ &= \frac{\beta}{2}\|x-y\|^2.\end{aligned}$$

Demostremos también que para f convexa, si $x, y \in \mathbb{R}^d$:

$$f(x)+\langle \nabla f(x), y-x\rangle \leq f(y). \tag{6}$$

Si $x, y \in \mathbb{R}^d$, definimos la función $g : [0,1] \to \mathbb{R}$ dada por:

$$g(t) = f(ty+(1-t)x)-tf(y)-(1-t)f(x).$$

Por la convexidad de f, $g(t) \leq 0$. Además $g(0) = 0$. Juntando esto nos queda que:

$$g'(0) = \lim_{t\to 0^+}\frac{g(t)-0}{t} \leq 0.$$

Teniendo en cuenta que $g'(t) = \nabla f(x(1-t)+yt)^T(y-x)+f(x)-f(y)$, de la desigualdad anterior se deduce que:

$$\nabla f(x)^T(y-x)+f(x)-f(y) \leq 0,$$

que es lo que queríamos.

Usando estas desigualdades ya podemos demostrar la proposición. Haciendo uso de la desigualdad (5) y teniendo en cuenta que $x_{n+1} = x_n - \frac{1}{\beta}\nabla f(x_n)$:

$$\begin{aligned}f(x_{n+1}) &\leq f(x_n)+\langle \nabla f(x_n), x_{n+1}-x_n\rangle + \frac{\beta}{2}\|x_{n+1}-x_n\|^2 \\ &= f(x_n)-\frac{\|\nabla f(x_n)\|^2}{\beta}+\frac{\|\nabla f(x_n\|^2)}{2\beta} \\ &= f(x_n)-\frac{1}{2\beta}\|\nabla f(_n)\|^2.\end{aligned}$$

Esto nos dice que la sucesión $(f(x_n))_{n=1}^{\infty}$ es decreciente. Juntándolo con la desigualdad (6) tomando $y = x_n$, $x = x_{n+1}$:

$$\begin{aligned}f(x_{n+1}) &\leq f(x^*)+\frac{2\beta\langle \nabla\frac{f(x_n)}{\beta}, x_n-x^*\rangle}{2}-\frac{\beta}{2}\|\nabla\frac{f(x_n)}{\beta}\|^2 \\ &= f(x^*)+\frac{\beta}{2}\|x_n-x^*\|^2-\frac{\beta}{2}(\|x_n-x^*\|^2-2\langle\nabla\frac{f(x_n)}{\beta}, x_n-x^*\rangle+\|\nabla\frac{f(x_n)}{\beta}\|^2) \\ &= f(x^*)+\frac{\beta}{2}\|x_n-x^*\|^2-\frac{\beta}{2}\|x_n-\frac{\nabla f(x_n)}{\beta}\|^2-x^* \\ &= f(x^*)+\frac{\beta}{2}(\|x_n-x^*\|^2-\|x_{n+1}-x^*\|^2).\end{aligned}$$

La última igualdad invita a considerar una suma tipo telescópica. Si recordamos que la sucesión $(f(x_n))_{n=1}^{\infty}$ es decreciente:

$$n(f(x_n) - f(x^*) \leq \sum_{i=0}^{n-1}(f(x_i) - f(x^*)) \leq \frac{\beta}{2}(\|x_n - x^*\|^2 - \|x_{n+1} - x^*\|^2)$$
$$= \frac{\beta}{2}(\|x_0 - x^*\|^2 - \|x_n - x^*\|^2) \leq \frac{\beta}{2}\|x_0 - x^*\|^2.$$

Por lo tanto, deducimos que, $\forall n \in \mathbb{N}$:

$$f(x_n) - f(x^*) \leq \frac{\beta\|x_0 - x^*\|^2}{2n}.$$

Como la sucesión de la derecha converge hacia 0 cuando $n \to \infty$, el algoritmo de descenso de gradiente converge hacia $f(x^*)$, el mínimo global de f. □

Si la función f no es convexa, el algoritmo puede converger un mínimo local, lo cual es algo problemático. Otro problema práctico surge con el algoritmo del descenso del gradiente: en los algoritmos de aprendizaje automático al minimizar la esperanza de la función de pérdida como en (4) , el descenso del gradiente vendría dado por:

$$x_{n+1} = x_n - \gamma_n \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta l(G_\theta(x_i)),$$

lo que requeriría evaluar el gradiente n veces por cada iteración. Es común usar conjuntos de entrenamiento con muchos ejemplos, luego el coste computacional de este algoritmo es muy grande. Con intención de resolver estos problemas, se introduce el algoritmo del descenso de gradiente estocástico, que es el de uso más común.

En el descenso del gradiente estocástico se toma una muestra z aleatoriamente del conjunto de entrenamiento y se calcula:

$$x_{n+1} = x_n - \gamma_n \nabla_\theta l(G_\theta(z)),$$

En el descenso de gradiente estocástico con minilotes se toman m muestras aleatoriamente $z_1 ... z_m$ y se itera:

$$x_{n+1} = x_n - \gamma_n \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta l(G_\theta(z_i)),$$

En esta versión cogemos un m tal que computar m evaluaciones del gradiente por iteración sea factible. El análisis de la convergencia del descenso del gradiente estocástico es más complejo que en el descenso del gradiente, pero su uso es muy ventajoso. La convergencia es más rápida, nos permite aproximar el gradiente de una forma tratable computacionalmente, y también se ha observado que es más propenso a evitar mínimos locales, debido a su naturaleza aleatoria. También es posible asegurar que el algoritmo converge rápido en condiciones adecuadas. Para más detalles sobre la velocidad de convergencia de los algoritmos y las ventajas del descenso de gradiente estocástico se puede consultar [4] y [5].

2.6. Redes neuronales y retropropagación

Cuando planteamos el aprendizaje supervisado, queríamos resolver un problema de optimización: elegir una función dentro de una clase de funciones hipótesis $\mathcal{F}$ que minimizara la función objetivo. Dimos un ejemplo con la regresión logística, donde considerábamos $\mathcal{F}$ como

un tipo específico de clasificadores lineales parametrizados. Es importante que las funciones hipótesis estén parametrizadas para poder aplicar algún algoritmo de optimización basado en el descenso del gradiente.

Las clases de funciones que usaremos para las GANs son las redes neuronales. Las redes neuronales son funciones de la forma:

$$g_\theta(x) = f_L(W_L f_{L-1}(W_{L-1}...f_1(W_1 x)))),$$

donde W_i, $i = 1, ..., L$ son matrices de pesos, con $\dim(W_i) = d_i \times d_{i-1}$, la función $f_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ es la función de activación de la capa i y L es el número de capas. Las funciones de activación son funciones que se aplican componente a componenete. Los parámetros θ son los pesos de todas las W_i, esto es, los parámetros de las transformaciones lineales; las funciones de activación se eligen de antemano y no están parametrizadas (no se aprenden). Notese que si no fuera por las funciones de activación, las redes neuronales serían funciones lineales. Con ellas pueden introducirse no-linearidades.

A través de las funciones de activación podemos crear redes neuronales que se adapten a diferentes problemas. Para problemas de clasficación binaria, como antes, querremos que $g_{\theta x}$ aproxime $P(y = 1|x)$ luego es natural volver a considerar la sigmoide como la función de activación en la última capa, esto es, $\dim(W_L) = 1 \times d_{f_{L-1}}$ y $f_L(x) = \frac{1}{1+e^{-z}}$. En el caso de la clasificación multiclase, es común usar la softmax en la última capa: si hay K clases, entonces $\dim(W_L) = K \times d_{f_{L-1}}$ y $f_L(x) = \frac{(e^{x_1},...,e^{x_k})}{\sum_{i=1}^{k} e^{x_i}}$. Para la regresión, $\dim(W_L) = 1 \times d_{f_{L-1}}$ y la última función de activación es simplemente la identidad.

Las funciones de activación en las capas intermedias, el número de capas y las dimensiones dependen mucho del problema. La función de activación más común para capas intermedias es la $ReLU$ dada por $max(x, 0)$, que se aplica componente a componente.

Para usar un algoritmo basado en el descenso del gradiente con una red neuronal, necesitaremos calcular de forma eficiente el gradiente de la red. Para ello se usa el algoritmo de retropropagación. Consideramos una función de coste $C(g_\theta(x))$, que toma valores reales. Queremos calcular $\nabla_\theta C(g_\theta)(x)$. Definimos $a_0 = x$ y $a_k = f_k(W_k...(W_1 x))$ y $z_k = W_k a_{k-1}$. Demominamos D_k a la matriz Jacobiana de f_k. Notese que por ser f_k una función componente a componente, D_k es diagonal. Por la regla de la cadena, se tiene que:

$$\nabla_{W_L} C = (\nabla C(g_\theta(x))) \cdot (D_L(z_L) a_{L-1}) = (\nabla C(g_\theta(x)) \cdot D_L(z_k)) \cdot a_{L-1}.$$

Puesto que $D_k(z_k)$ es diagonal $\forall k \in \{1...L\}$, consideramos $f'_k = diag(D_k(z_k)) = (D_{k_{1,1}}, D_{k_{2,2}}, ..., D_{k_{d_k,d_k}})$ la diagonal de la matriz $D_k(z_k)$, y podemos escribir:

$$\nabla_{W_L} C = (\nabla C(g_\theta(x)) \odot f'_L) \cdot a_{L-1},$$

donde aquí $\odot$ denota el producto de Hadamard, que es el producto de matrices componente a componente. De forma similar, aplicando la regla de la cadena, podemos escribir:

$$\begin{aligned}
\nabla_{W_{L-1}} C &= (\nabla C(g_\theta(x))) \cdot (D_L \cdot W_L \cdot D_{L-1} \cdot a_{L-2}) \\
&= ((\nabla C(g_\theta(x) \odot f'_L \cdot W_L) \odot f'_{L-1}) \cdot a_{L-2} \\
&= (W_L^T \cdot (\nabla C(g_\theta(x) \odot f'_L) \odot f'_{L-1}) \cdot a_{L-2}.
\end{aligned}$$

Y de forma análoga podemos escribir, para $k \in \{1, ..., L-1\}$:

$$\nabla_{W_{L-k}} C = (W_{L-k+1}^T ... W_L^T \cdot (\nabla C(g_\theta(x)) \odot f'_L \odot ... \odot f'_{L-k}) \cdot a_{L-k-1}. \tag{7}$$

Lo cual nos proporciona una manera de calcular el gradiente de C con respecto a todos los parámetros θ por capas, y usando los cálculos de la capa anterior cada vez, haciendo el cálculo lo más eficiente posible. Para hacer estos cáculos necesitamos los valores z_k, $a_k = f_k(z_k)$, $f'_k = diag(D_k(z_k))$ y $\nabla C(g_\theta(x))$. Podemos calcularlos eficientemente de forma iterada. Sea $x \in \mathbb{R}^{d_1}$ y $\theta = (W_1, ..., W_L)$:

- $a_0 = x$.
- para $k = 1, ..., L$:

$$\begin{aligned} z_k &:= W_k a_{k-1}, \\ a_k &:= f_k(z_k), \\ f'_k &:= diag(D_k(z_k)). \end{aligned}$$

- Calculamos $\nabla C(g_\theta(x))$.

Una vez tengamos esto podemos guardar los valores $(W_{L-k+1}^T ... W_L^T \cdot (\nabla C(g_\theta(x)) \odot f'_L \odot ... \odot f'_{L-k})$ que aparecen repetidamente en (7) para hacer los menos cálculos posibles:

- $l = \nabla C(g_\theta(x))$.
- Para $k = L, ..., 1$:

$$\begin{aligned} l &= l \odot f'_k, \\ \nabla_{W_k} C &= l \cdot a_{k-1}^T, \\ l &= W_k^T \cdot l. \end{aligned}$$

Este es el algoritmo de retropropagación. Es de especial importancia porque permite el uso de los algoritmos de optimización comunes de los que hemos hablado en la sección anterior, junto con todas las variantes basadas en el descenso del gradiente.

3. Redes Generativas Adeversariales (GANs)

3.1. El problema minimax

El objetivo de las Redes Generativas Adversariales es el de generar objetos similares a los de una colección $x_1, ...x_n$ de muestras aleatorias i.i.d, donde $x_i \in \mathcal{X}$. Para ello, se presentan dos modelos (que serán redes neuronales, que podemos parametrizar): un generador, que creará nuevas muestras que intenten parecerse a los datos del conjunto de entrenamiento, y un discriminador, que nos dará la probabilidad de que una muestra proceda del conjunto de datos (es decir, la probabilidad de que la muestra sea "verdadera").

La generación de nuevas observaciones en alta dimensión es una tarea complicada, y en este caso, la mayoría de aplicaciones son en espacios de muy alta dimensión (por ejemplo, la generación de imágenes). Por ello, para construir la distribución $\mathbb{P}_\theta$ del generador, definimos primero un vector aleatorio $Z \in \mathcal{Z}$, que seguirá una distribución conocida, (comúnmente una distribución normal o uniforme) y le aplicaremos una transformación $G_\theta : Z \to \mathcal{X}$, que será una red neuronal parametrizada por θ, que nos transformará ruido aleatorio en un elemento de nuestro espacio $\mathcal{X}$. El objetivo es entrenar la red G_θ hasta que $\mathbb{P}_\theta$ se acerque a $\mathbb{P}_r$.

Por otro lado, consideramos un discriminador $D_w : \mathcal{X} \to [0, 1]$ que a cada $x \in \mathcal{X}$ le asignará la probabilidad de que x proceda del conjunto de datos en vez de haber sido generado por G_θ.

La idea es entrenar al discriminador y el generador a la vez, y que se vayan retroalimentando mutuamente. Entrenar un discriminador nos da un criterio con el que poder medir cuánto se parecen los objetos generados a los del conjunto de datos. Para entrenarlo, queremos que la probabilidad sea alta cuando un objeto x venga de $\mathbb{P}_r$ y que sea baja cuando un objeto $G_\theta(z)$ haya sido generado por G_θ. Así, queremos que maximice $\mathbb{E}_{x\sim\mathbb{P}_r}[\log D_w(x)]$ y $\mathbb{E}_{z\sim p_z}[\log(1 - D_w(G_\theta(z)))]$, y por tanto, nos queda la expresión:

$$\max_w \mathbb{E}_{x\sim\mathbb{P}_r}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))] \tag{8}$$

Ahora, para entrenar el generador, queremos que "engañe" al discriminador. Esto es, maximizar la probabilidad de que el discrimador (ahora fijo) prediga que el objeto generado $G_\theta(z)$ viene de $\mathbb{P}_r$. Por ello, entrenamos G_θ para maximizar $\mathbb{E}_{z\sim p_z}[\log D_w(G_\theta(z))]$, lo que es equivalente a minimizar $\mathbb{E}_{z\sim p_z}[\log(1 - D_w(G_\theta(z)))]$. Puesto que $\mathbb{E}_{x\sim\mathbb{P}_r}[\log D_w(x)]$ no depende de G_θ, entonces nos queda el problema minimax:

$$\min_\theta \max_w V(G_\theta, D_w),$$

donde:

$$V(G_\theta, D_w) = \mathbb{E}_{x\sim\mathbb{P}_r}[\log D_w(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D_w(G_\theta(z)))] \tag{9}$$

Nótese que al ecuación 8 es muy similar a la función de pérdida de la entropía cruzada que se minimiza cuando se entrena un clasificador binario, tal como habíamos adelantado en la introducción.

3.2. Relación con la distancia de Jensen-Shannon

En esta sección veremos cuál es el generador dado para un discriminador óptimo. También veremos la estrecha relación que hay entre entrenar las GAN y minimizar la distancia de Jensen-

Shannon.

Proposición 7. *Dado G fijo, el discriminador óptimo viene dado por:*

$$D_G(x) = \frac{\mathbb{P}_r(x)}{\mathbb{P}_r(x) + \mathbb{P}_\theta(x)} \tag{10}$$

Demostración. Con un generador fijo, el entrenamiento para D se basa en maximizar $V(G, D)$:

$$V(G, D) = \int_{\mathcal{X}} p_r(x) \log(D(x))dx + \int_{\mathcal{Z}} p_z(z) \log(1 - D(G(z))dz$$

Teniendo en cuenta las propiedades de la medida definida por G_θ, nos queda:

$$V(G, D) = \int_{\mathcal{X}} p_r(x) \log(D(x)) + p_\theta(x) \log(1 - D(x))dx$$

Notese que $p_r(x) \geq 0$, $p_\theta(x) \geq 0$ y $D(x) \in [0, 1]$. Así, para $x \in [0, 1]$ y $a, b \geq 0$ buscamos el máximo de la función $h(x) = a \log(x) + b \log(1 - x)$:

$$h'(x) = \frac{a}{x} - \frac{b}{1-x} = 0 \iff (1-x)a - xb = 0 \iff x = \frac{a}{a+b}$$

Fácilmente se ve que a la derecha de este valor la derivada es negativa, y a la izquierda es positiva. Por tanto estamos ante el máximo de la función. Por tanto, podemos definir el discriminador óptimo para un G fijo como:

$$D_G(x) = \frac{p_r(x)}{p_r(x) + p_\theta(x)}, \tag{11}$$

siempre que p_r o p_θ no se anulen. Fuera del soporte de ambas densidades definiríamos el discriminador como $D_G(x) = 0$ y habríamos concluido. □

Así, si nuestro generador es óptimo (esto es, $\mathbb{P}_\theta = \mathbb{P}_r$), entonces $D_G = \frac{1}{2}$. Esto quiere decir que el discriminador, dado el generador óptimo, está "adivinando al azar"para predecir si una imagen es real o no.

Teniendo esto en cuenta, si la clase de discriminadores no está restringida, y G es un generador fijo, podemos reescribir la función de pérdida para el discriminador:

$$\begin{aligned}
\max_D V(G, D) &= \mathbb{E}_{x\sim\mathbb{P}_r}[\log(D_G(x))] + \mathbb{E}_{x\sim\mathbb{P}_\theta}[\log(1 - D_G(x))] \\
&= \mathbb{E}_{x\sim\mathbb{P}_r}[\log \frac{\mathbb{P}_r(x)}{\mathbb{P}_r(x) + \mathbb{P}_\theta(x)}] + \mathbb{E}_{x\sim\mathbb{P}_\theta}[\log \frac{\mathbb{P}_\theta(x)}{\mathbb{P}_r(x) + \mathbb{P}_\theta(x)}] \\
&= -2\log(2) + KL(\mathbb{P}_r||\frac{\mathbb{P}_r + \mathbb{P}_\theta}{2}) + KL(\mathbb{P}_\theta||\frac{\mathbb{P}_r + \mathbb{P}_\theta}{2}) \\
&= -2\log(2) + 2JS(\mathbb{P}_r||\mathbb{P}_\theta)
\end{aligned} \tag{12}$$

Proposición 8. *El mínimo* $\min_G V(G, D_G)$ *se alcanza si ,y solo si,* $\mathbb{P}_\theta = \mathbb{P}_r$.

Demostración. Teniendo en cuenta la observación (12):

$$\min_G \max_D V(G, D) = \min_G[-\log(4) + 2JS(\mathbb{P}_r||\mathbb{P}_\theta)]$$

Por ser JS una distancia, el mínimo se alcanza cuando $\mathbb{P}_r = \mathbb{P}_\theta$ y su valor es $-\log(4)$. □

Notemos que la suposición de que $\mathbb{P}_r$ esté entre las distribuciones generadas por G_θ para algún $\theta \in \Theta$ no se da en la práctica.

3.3. El algoritmo GAN

Ya tenemos lo necesario para poder formular un algoritmo de aprendizaje basado en el descenso del gradiente u otro método similar basado en él. En este caso daremos el algoritmo con el descenso del gradiente con minilote. Hemos visto ya que la diferencia con el algoritmo tradicional es que en cada iteración del descenso del gradiente usamos todos los datos del conjunto de entrenamiento, lo cual requiere un gran coste computacional. Aquí, sin embargo, cogemos un conjunto pequeño de datos en cada iteración.

Sea m fijo el tamaño de un minilote. En cada iteración usaremos m muestras de Z y m muestras de $\mathbb{P}_r$. Sea también γ la tasa de aprendizaje. Cada iteración del algoritmo funcionaría de la forma siguiente:

1. Obtener m muestras de la distribución fija en Z, $\{z^{(1)}, ..., z^{(m)}\}$.

2. Elegir m datos del conjunto de entrenamiento, $\{x^{(1)}, ..., x^{(m)}\}$.

3. Actualizamos el discriminador. En este caso, como tenemos que maximizar ecuación 8, actualizaremos los parámetros de D_w siguiendo la dirección del gradiente, en vez de seguir la opuesta, ya que estamos maximizando:

$$w = w + \gamma \nabla_w \frac{1}{m} \sum_{k=1}^{m} [\log D_w(x^{(k)}) + \log(1 - D_w(G_\theta(z^{(k)}))]$$

   Nótese que estamos tomando la media muestral como estimador de la esperanza, como es habitual.

4. Obtener otras m muestras de la distribución fija en Z, $\{z^{(1)}, ..., z^{(m)}\}$.

5. Actualizar el generador. Aquí si que tenemos que seguir la dirección opuesta al gradiente puesto que estamos minimizando la ecuación 8:

$$\theta = \theta - \gamma \nabla_\theta \frac{1}{m} \sum_{k=1}^{m} \log(1 - D_w(G_\theta(z^{(k)}))$$

   Observemos que el término $\log D_w(x)$ en la ecuación 8 solo está para que el discriminador aprenda a dar una alta probabilidad a los objetos del conjunto de entrenamiento y no involucra al generador; por tanto, si derivamos con respecto a los parámetros θ, el término desaparece.

Para algoritmos de aprendizaje más tradicionales, se iteraría este algoritmo hasta que los parámetros converjan, lo cual quiere decir que los gradientes sean cero (o sea, habríamos encontrado un mínimo o máximo local, según el problema que estemos tratando). Sin embargo, las GANs son conocidas por su inestabilidad al entrenarlas, y no hay un criterio de convergencia claro. [6]. Además de esto, surgen otros problemas:

- Desvanecimiento del gradiente. Como se ve en [6], cuando el discriminador se acerca al discriminador óptimo para un generador dado, puede provocar que el discriminador sea constante, lo cual hace que no podamos aprender nada a través del descenso del gradiente. Además, tal y como vimos en la sección de conceptos previos, la divergencia de Jensen-Shannon también se hace constante cuando las distribuciones $\mathbb{P}_\theta$ y $\mathbb{P}_r$ tienen soportes disjuntos. Esto también es cierto para condiciones más generales cuando las distribuciones se cortan en un conjunto de medida nula. Los detalles pueden verse en [6].

- Escasa variedad de generaciones. Esto se da cuando la distribución $\mathbb{P}_\theta$ aprende a generar muestras que se parecen a las de $\mathbb{P}_r$, pero son poco variadas o siempre la misma, de forma que no se ha llegado a captar la complejidad de la distribución $\mathbb{P}_r$. Un análisis sobre esto hecho pueden verse en [7].

4. Distancia de Wasserstein

Las GANs no tienen una función de pérdida objetiva, de forma que nos es difícil evaluar la calidad de los objetos generados. La evaluación de un modelo se hace pues observando si G_θ está generando objetos "parecidos.ª los del conjunto de entrenamiento, lo cual no es ningún criterio para valorar si el generador está aprendiendo la distribución de p_{datos}.

Por ello, introducimos la distancia de Wasserstein, que nos va a permitir definir una función de pérdida con el comportamiento deseado y que nos sirva como tal para entrenar las GANs. Para $p \in [1, \infty]$, la p-distancia de Wasserstein entre dos medidas de probabilidad $\mathbb{P}_g$, $\mathbb{P}_r$ en X, se define como:

$$W(\mathbb{P}_g, \mathbb{P}_r) = (\inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|^p])^{\frac{1}{p}}, \tag{13}$$

donde $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denota el conjunto de todas las distribuciones γ en $X \times X$ con marginales $\mathbb{P}_r$ y $\mathbb{P}_g$. Esto es, γ cumple que:

$$\int_X \gamma(x, y)dy = \mathbb{P}_r \quad \int_X \gamma(x, y)dx = \mathbb{P}_g$$

En lo que sigue, usaremos $p = 1$, que es todo lo que vamos a necesitar.

Introducimos también la dualidad de Kantorovich-Rubinstein, una forma de representar la distancia de Wasserstein que nos será muy útil para manejarla en la práctica. La demostración queda fuera de los contenidos de este trabajo, pero se puede consultar en [8]

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]. \tag{14}$$

Podemos usar esta expresión para comprobar que W es efectivamente una distancia:

Proposición 9. *Sea $Prob(\mathcal{X})$ el conjunto de medidas de probabilidad definidas en $\mathcal{X}$. La función $W : Prob(\mathcal{X}) \times Prob(\mathcal{X}) \to \mathbb{R}$ definida en (13) es una distancia.*

Demostración.

- W es simétrica. Observemos que si f es 1-Lipschitz, entonces $-f$ también lo es. Luego si $(f_n)_{n=1}^\infty$ es una sucesión de funciones 1-Lipschitz tal que:

  $$\lim_{n\to\infty} \mathbb{E}_{x\sim\mathbb{P}_r}[f_n(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f_n(x)] = W(\mathbb{P}_r, \mathbb{P}_g),$$

  $(-f_n)_{n=1}^\infty$ es una sucesión de funciones 1-Lipschitz con:

  $$\lim_{n\to\infty} \mathbb{E}_{x\sim\mathbb{P}_g}[-f_n(x)] - \mathbb{E}_{x\sim\mathbb{P}_r}[-f_n(x)] = W(\mathbb{P}_r, \mathbb{P}_g)$$

  De lo que se deduce que $W(\mathbb{P}_r, \mathbb{P}_g) \leq W(\mathbb{P}_g, \mathbb{P}_r)$. Se puede aplicar el mismo argumento cambiando los papeles de $\mathbb{P}_r$ y $\mathbb{P}_g$ para llegar a la igualdad.

- $W \geq 0$ ya que $\forall\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)$, se tiene que $\mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \geq 0$ por ser la esperanza de una función no negativa.

- Veamos que $W(\mathbb{P}_r, \mathbb{P}_g) = 0 \iff \mathbb{P}_r = \mathbb{P}_g$. Si $\mathbb{P}_r = \mathbb{P}_g$ es claro que $W(\mathbb{P}_r, \mathbb{P}_g) = 0$. Por otro lado, si $W(\mathbb{P}_r, \mathbb{P}_g) = 0$, usando (14) nos queda que:

  $$\sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] = 0,$$

lo que implica que $\forall f$ Lipschitz con $\|f\|_L \leq 1$:

$$\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] \leq \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)]$$

Como W es simétrica, de forma análoga llegamos a la otra desigualdad, luego nos queda que $\forall f$ Lipschitz con $\|f\|_L \leq 1$:

$$\mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] = \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] \tag{15}$$

Si f es K-Lipschitz, la igualdad se cumple igual, teniendo en cuenta que $\frac{f}{K}$ es 1-Lispchitz. Sea $A \in \mathcal{B}(\mathcal{X})$. Observemos que, por la desigualdad triangular:

$$|d(x, A) - d(y, A)| \leq \|x - y\|,$$

luego $d(x, A)$ es Lispchitz. Las combinaciones lineales de funciones Lipschitz también lo son: si f, g son K-Lipschitz y $a, b \in \mathbb{R}$

$$|af(x) + bg(x) - af(y) - bg(y) \leq (|a| + |b|)K(\|x - y\|)$$

Además, el mínimo de dos funciones Lipschitz también lo es: teniendo en cuenta que $\text{mín}\{f(x), g(x)\} = \frac{f(x)+g(x)-|f(x)-g(x)|}{2}$ necesitamos comprobar que $|f(x) - g(x)|$ es Lipschitz:

$$||f(x) - g(x)| - |f(y) - g(y)|| \leq |f(x) - g(x) - f(y) + g(y)| \leq 2K\|x - y\|$$

Para A cerrado, consideremos la sucesión de funciones $f_n(x) = 1 - \text{mín}\{d(x, A)n, 1\}$. Teniendo en cuenta las consideraciones anteriores, f_n es Lipschitz, $\forall n \in \mathbb{N}$. Además,$(f_n)_{n=1}^{\infty}$ converge a la función característica de A. Como $0 < \text{mín}\{d(x, A)n, 1\} \leq 1$, $0 \leq |f_n(x)| \leq 1 \forall x \in \mathcal{X}$, y por el teorema de la convergencia dominada:

$$\mathbb{P}_g(A) = \lim_{n\to\infty} \int_{\mathcal{X}} f_n d\mathbb{P}_g = \lim_{m\to\infty} \int_{\mathcal{X}} f_n d\mathbb{P}_r = \mathbb{P}_r(A),$$

donde se ha aplicado la igualdad (15) para funciones Lipschitz. Como los conjuntos cerrados generan la σ-álgebra de Borel, se deduce entonces que $\mathbb{P}_g = \mathbb{P}_r$.

- Veamos que se cumple la desigualdad triangular. Sea $\mathbb{P}_s \in Prob(\mathcal{X})$. Teniendo en cuenta que la distancia de Wasserstein es no negativa:

$$\begin{aligned}
W(\mathbb{P}_r, \mathbb{P}_g) &= \sup_{\|f\|_L\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] \\
&= \sup_{\|f\|_L\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_s}[f(x)] + \mathbb{E}_{x\sim\mathbb{P}_s}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] \\
&\leq \sup_{\|f\|_L\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_s}[f(x)] + \sup_{\|f\|_L\leq 1} \mathbb{E}_{x\sim\mathbb{P}_s}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] \\
&= W(\mathbb{P}_r, \mathbb{P}_s) + W(\mathbb{P}_s, \mathbb{P}_g)
\end{aligned}$$

□

Vamos a usar la distancia de Wasserstein como función de pérdida para un nuevo algoritmo. Ahora bien, para que dicha función de pérdida funcione adecuadamente es conveniente que sea continua; eventualmente si tenemos distribuciones parametrizadas $\mathbb{P}_\theta$, y $\mathbb{P}_{\theta_0}$ aproxima la distribución $\mathbb{P}_r$ de los datos, queremos que si $\theta \to \theta_0$ entonces $\mathbb{P}_\theta \to \mathbb{P}_{\theta_0}$, donde la convergencia es con la distancia W. Este no es el caso para las divergencias de Jensen-Shannon o Kullback-Leibler. Veámoslo con un ejemplo:

Sea $\mathbb{P}_r$ distribución de probabilidad en $\mathbb{R}$ discreta concentrada en 0. $P_r(x = 0) = 1$ y sea $\mathbb{P}_\theta$ discreta en $\mathbb{R}$ concentrada en $\theta \in \mathbb{R}$, independiente de $\mathbb{P}_r$ donde θ es un parámetro. Así, se tiene que:

- $W(\mathbb{P}_r, \mathbb{P}_\theta) = |\theta|$. La distribución conjunta γ en $\mathbb{R}^2$ que tiene por marginales $\mathbb{P}_r$ y $\mathbb{P}_\theta$ es la distribución concentrada en $(0, \theta)$, ya que ha de ser

  $$\gamma(X = x, Y = y) = \gamma(X = x)\gamma(Y = y) = P_r(X = x)P_\theta(Y = y)$$

  De esta manera, queda que:

  $$\mathbb{E}_{(x,y)\sim\gamma}(\|x - y\|) = \|0 - \theta\|P(0, \theta) = \|\theta\|$$

  Esta función es continua con respecto al parámetro: podríamos aprender la distribución $\mathbb{P}_r$ aplicando descenso del gradiente.

- $KL(\mathbb{P}_r\|\mathbb{P}_\theta) = KL(\mathbb{P}_\theta\|\mathbb{P}_r) = \begin{cases} \infty & \text{si } \theta \neq 0 \\ 0 & \text{si } \theta = 0 \end{cases}$. En efecto, si $\theta = 0$, entonces:

  $$KL(\mathbb{P}_r\|\mathbb{P}_\theta) = \log(\frac{P_r(0)}{P_\theta(0)})P_r(0) = 0$$

  Y por otro lado, si $\theta \neq 0$:

  $$KL(\mathbb{P}_r\|\mathbb{P}_\theta) = \log(\frac{P_r(0)}{P_\theta(0}})P_r(0) = \log(\frac{1}{0}) = \infty$$

  Se razona de forma análoga para $KL(\mathbb{P}_\theta\|\mathbb{P}_r)$

- $JS(\mathbb{P}_r, \mathbb{P}_\theta) = \begin{cases} \log(2) & \text{si } \theta \neq 0 \\ 0 & \text{si } \theta = 0 \end{cases}$. Si consideramos $\mathbb{P}_m = \frac{\mathbb{P}_r + \mathbb{P}_\theta}{2}$, y $\theta = 0$:

  $$JS(\mathbb{P}_r, \mathbb{P}_\theta) = KL(\mathbb{P}_r\|\mathbb{P}_m) + KL(\mathbb{P}_g\|\mathbb{P}_m) = \log(\frac{2P_r(0)}{(P_r + P_\theta)(0)}) + \log(\frac{2P_\theta(\theta)}{(P_r + P_\theta)(\theta)}) = 2\log 1 = 0$$

  Si $\theta \neq 0$:

  $$JS(\mathbb{P}_r, \mathbb{P}_\theta) = \log(\frac{2P_r(0)}{(P_r + P_\theta)(0)}) + \log(\frac{2P_\theta(\theta)}{(P_r + P_\theta)(\theta)}) = 2\log 2$$

Para las divergencias de Jensen-Shannon y Kullback-Leibler no podemos aplicar descenso del gradiente u otro algoritmo basado en el, ya que las funciones no son continuas respecto a los parámetros.

En el ejemplo, los soportes de las distribuciones $\mathbb{P}_r$ y $\mathbb{P}_\theta$ tienen intersección vacía. Esto es lo que hace que las funciones de pérdida JS y KL no sean continuas, algo que se cumple, en general, cuando la intersección está contenida en un conjunto de medida nula. Este precisamente es el caso que nos interesa: la dimensión del espacio X de objetos a generar será generalmente muy grande, pero el soporte de la distribución de los datos será mucho menor. Así será difícil que la intersección entre los soportes de $\mathbb{P}_r$ y $\mathbb{P}_\theta$ tenga intersección con medida > 0.

Ahora veamos que efectivamente, la distancia de Wasserstein es continua en las condiciones adecuadas:

Proposición 10. *Sea $\mathbb{P}_r$ una distribución sobre X. Sea Z una variable aleatoria sobre otro espacio Z, y sea $g_\theta : Z \times \mathbb{R}^d \to X$ una función (que en la práctica será el generador de las GANs: g_θ será una red neuronal con entrada en Z y salida en X, parametrizada por $\theta \in \mathbb{R}^d$). Sea $\mathbb{P}_\theta$ la distribución de imagen correspondiente a $g_\theta(Z)$. Se cumple que:*

1. *Si g_θ es continua con respecto a los parámetros θ, $W(\mathbb{P}_r, \mathbb{P}_\theta)$ también lo es.*

2. *Si g_θ es localmente Lipschitz y existen, para cada $(z, \theta) \in Z \times \mathbb{R}^d$ constantes de Lipschitz locales $L(z, \theta)$ tales que:*

$$\mathbb{E}_{z\sim p}[L(z, \theta)] < \infty \tag{16}$$

entonces $W(\mathbb{P}_r, \mathbb{P}_\theta)$ es continua en todo punto, y diferenciable en casi todo punto.

3. *1 y 2 son falsas para la divergencia JS y la KL.*

Demostración. Sea θ_0 fijo. Queremos ver que $W(\mathbb{P}_r, \mathbb{P}_{\theta_0})$ es continua, es decir:

$$|W(\mathbb{P}_r, \mathbb{P}_\theta) - W(\mathbb{P}_r, \mathbb{P}_{\theta_0})| \to_{\theta\to\theta_0} 0$$

Primero notar que, como W es una distancia, por la desigualdad triangular:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) \leq W(\mathbb{P}_r, \mathbb{P}_{\theta_0}) + W(\mathbb{P}_\theta, \mathbb{P}_\theta)$$

Por lo tanto, nos queda que:

$$|W(\mathbb{P}_r, \mathbb{P}_\theta) - W(\mathbb{P}_r, \mathbb{P}_{\theta_0})| \leq W(\mathbb{P}_\theta, \mathbb{P}_\theta)$$

Por lo que nos basta acotar el término $W(\mathbb{P}_\theta, \mathbb{P}_\theta)$. Recordemos que la distribución de $\mathbb{P}_\theta$ viene definida por la medida de imagen definida por g_θ. Así, la distribución conjunta (y medida de imagen) γ definida por (g_θ, g_{θ_0}) tiene por marginales $\mathbb{P}_\theta$ y $\mathbb{P}_{\theta_0}$, y por tanto:

$$W(\mathbb{P}_\theta, \mathbb{P}_{\theta_0}) = \inf_{\eta\in\Pi(\mathbb{P}_\theta, \mathbb{P}_{\theta_0})} \mathbb{E}_{(x,y)\sim\eta}[\|x - y\|] \leq \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$$

Además, se tiene que:

$$\mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] = \mathbb{E}_{z\sim p_Z}[\|g_\theta(z) - g_{\theta_0}(z)\|]$$

Puesto que g_θ es continua con respecto a θ, si $\theta \to \theta_0$, entonces $\|g_\theta(z) - g_{\theta_0}(z)\| \to 0\ \forall z \in Z$. Ahora bien, puesto que X es compacto, existe una constante $M > 0$ tal que:

$$\|g_\theta(z) - g_{\theta_0}(z)\| < M\ \forall z \in Z$$

Por ello, podemos aplicar el teorema de la convergencia dominada obteniendo que:

$$W(\mathbb{P}_\theta, \mathbb{P}_{\theta_0}) \leq \mathbb{E}_{z\sim p_Z}[\|g_\theta(z) - g_{\theta_0}(z)\|] \to_{\theta\to\theta_0} 0$$

Por lo tanto, la función $W(\mathbb{P}_r, \mathbb{P}_\theta)$ es continua con respecto a θ.

Ahora supongamos que g es localmente Lipschitziana, y veamos que $W(\mathbb{P}_r, \mathbb{P}_\theta)$ también lo es. Entonces dado un punto (z_0, θ_0) fijo, existe una constante $L(z_0, \theta_0)$ y un abierto U con (z_0, θ_0) tal que si $(z, \theta) \in U$ entonces:

$$\|g_\theta(z) - g_{\theta_0}(z_0) \leq L(z_0, \theta_0)\|(z - z_0, \theta - \theta_0)\|$$

Tomando $z = z_0$ y tomando esperanzas se obtiene que:

$$\mathbb{E}_{z_0\sim Z}[\|g_\theta(z_0) - g_{\theta_0}(z_0)\|] \leq \mathbb{E}_{z_0\sim Z}[L(z_0, \theta_0)]\|(0, \theta - \theta_0)\| = \mathbb{E}_{z_0\sim Z}[L(z_0, \theta_0)]\|\theta - \theta_0\|,$$

si $(z_0, \theta) \in U$ .En el último término estamos usando la norma en $\mathbb{R}^d$.

Definimos así el conjunto $U_0 = \pi_{\mathbb{R}^d}(U) = \{\theta|(z_0, \theta) \in U\}$, que es la proyección del abierto U en las variables θ. Las proyecciones son funciones abiertas, luego U_0 es un abierto también.

Además, por hipótesis $L(\theta_0) = \mathbb{E}_{z_0 \sim Z}[L(z_0, \theta_0)] < \infty$. Así, usando la cota vista anteriormente relacionada con la distribución conjunta γ:

$$|W(\mathbb{P}_r, \mathbb{P}_\theta) - W(\mathbb{P}_r, \mathbb{P}_{\theta_0})| \leq W(\mathbb{P}_\theta, \mathbb{P}_{\theta_0}) \leq L(\theta_0)\|\theta - \theta_0\|$$

para todo $\theta \in U_0$. Hemos encontrado así, para cada $\theta_0 \in \mathbb{R}^d$ una constante$L(\theta_0)$ y un abierto U_0 tal que si $\theta \in U_0$:

$$|W(\mathbb{P}_r, \mathbb{P}_\theta) - W(\mathbb{P}_r, \mathbb{P}_{\theta_0})| \leq L(\theta_0)\|\theta - \theta_0\|$$

Luego $W(\mathbb{P}_r, \mathbb{P}_\theta)$ es localmente Lipschitziana con respecto a los parámetros θ. Esto implica que la función es continua. Además, por el teorema de Rademacher, $W(\mathbb{P}_r, \mathbb{P}_\theta)$ es diferenciable casi siempre en $\mathbb{R}^d$.

El apartado 3 es consecuencia del ejemplo anterior. □

Necesitamos pues asegurarnos de que las redes neuronales cumplen la condición dada por la ecuación (16). Lo demostraremos para redes neuronales con funciones de activación diferenciables, pero también es un resultado cierto para redes $feedforward$ que usan la $ReLU$ como función de activación en alguna de sus capas, (que es la situación más común en la práctica). Recordemos que la $ReLU$ es la función máx$(0, x)$, y no es diferenciable en el 0.

Proposición 11. *Sea $g_\theta : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$ una red neuronal parametrizada por θ tal que sus funciones de activación sean Lipschitz, y $p(z)$ una probabilidad sobre en Z tal que $\mathbb{E}_{z\sim p(z)}[\|z\|] < \infty$. Entonces se tiene que g_θ es localmente Lipschitziana, y existen constantes de Lipschtiz locales tales que:*

$$\mathbb{E}_{z\sim p(z)}[L(z, \theta)] < \infty.$$

Demostración. Primer notemos que, por la definición de diferenciabilidad, para cada (z, θ) existe una función $\epsilon : \mathcal{Z} \to \mathcal{X}$ con $\lim_{(z^*,\theta^*)\to(z,\theta)} \|\epsilon(z^*, \theta^*)\| = 0$ tal que:

$$g_{\theta^*}(z^*) - g_\theta(z) = \nabla_{(\theta,z)} g_\theta(z)((\theta^* - \theta, z^* - z)) + \|(\theta^* - \theta, z^* - z)\|\epsilon(z^*, \theta^*).$$

De donde se deduce que:

$$\|g_{\theta^*}(z^*) - g_\theta(z)\| \leq (\|\nabla_{(\theta,z)} g_\theta(z)\| + \|\epsilon(z^*, \theta^*)\|)\|(\theta^* - \theta, z^* - z)\|.$$

Y puesto que $\lim_{(z^*,\theta^*)\to(z,\theta)} \|\epsilon(z^*, \theta^*)\| = 0$, se tiene que, para cada $\delta > 0$ y para cada punto (θ, z), $\|\nabla_{(\theta,z)} g_\theta(z)\| + \delta$ es una constante de Lispchitz local de g_θ. Así pues, basta ver que:

$$\mathbb{E}_{z\sim p(z)}[\|\nabla_{(\theta,z)} g_\theta(z)\|] < \infty.$$

Teniendo en cuenta la expresión de una red neuronal con H capas, se tiene que:

$$\nabla_z g_\theta(z) = \nabla_z(f_H(W_H f_{L-1}(...f_1(W_1 z)))) = \prod_{k=1}^{H} D_k W_k,$$

donde W_k son las matrices de parámetros y D_k las matrices Jacobianas de las no-linearidades. De forma análoga, si $k \in \{1, ..., H\}$:

$$\nabla_{W_k} g_\theta(z) = \nabla_{W_k}(f_H(W_H f_{L-1}(...f_1(W_1 z)))) = ((\prod_{i=k+1}^{H} D_i W_i) D_k) g_{k-1}(z),$$

donde g_{k-1} es la aplicación que llega hasta la capa $k-1$:

$$g_{k-1}(z) = f_{k-1}(W_{k-1}f_{k-2}(...W_1 z)),$$

y con ∇_{W_k} denotamos que estamos derivando con respecto a todos los paramétros que están en W_k. Sea también L la constante de Lipschitz de las no-linearidades (si se usan más de una diferentes, se coge el máximo de las constantes de Lispchitz de las no-linearidades). Se tiene pues que $\|D_k\| \leq L$. Además, ser Lipschitz las funciones de activación, se tiene que:

$$\begin{aligned}\|f_{k-1}(W_{k-1}...(W_1 z))\| &\leq \|f_{k-1}(W_{k-1}...(W_1 z)) - f(0) + f(0)\| \\ &\leq L\|W_{k-1}...(W_1 z)\| + \|f(0)\|,\end{aligned}$$

y razonando por recurrencia:

$$\|f_{k-1}(W_{k-1}...(W_1 z))\| \leq \|z\|L^{k-1}\prod_{i=1}^{k-1}\|W_i\| + \sum_{i=1}^{k-1}\|f_{k-1}(0)\|.$$

Juntando todo esto, obtenemos:

$$\begin{aligned}\|\nabla_{(z,\theta)}g_\theta(z)\| &\leq \|\nabla_z g_\theta(z)\| + \sum_{k=1}^{H}\|\nabla_{W_k}g_\theta(z)\| \\ &\leq L^H\prod_{i=1}^{H}\|W_i\| + \|z\|\sum_{k=1}^{H}[L^{H+1}(\prod_{i=k+1}^{H}\|W_i\|)(\prod_{i=1}^{k-1}\|W_i\|) + \sum_{i=1}^{k-1}\|f_{k-1}(0)\|].\end{aligned}$$

Ahora si tenemos en cuenta que $\mathbb{E}_{z\sim p(z)}[\|z\|] < \infty$ y que las cantidades $L^H\prod_{i=1}^{H}\|W_i\|$ y $\sum_{k=1}^{H}[L^{H+1}(\prod_{i=k+1}^{H}\|W_i\|)(\prod_{i=1}^{k-1}\|W_i\|) + \sum_{i=1}^{k-1}\|f_{k-1}(0)\|]$ son constantes respecto a z, se concluye que:

$$\mathbb{E}_{z\sim p(z)}[\|\nabla_{(z,\theta)}g_\theta(z)\|] < \infty,$$

que es lo que queríamos □

Corolario 1. *Sea $g_\theta : \mathcal{Z}\times\mathbb{R}^d \to \mathcal{X}$ una red neuronal parametrizada por θ tal que sus funciones de activación sean Lipschitz, y $p(z)$ una probabilidad sobre en Z tal que $\mathbb{E}_{z\sim p(z)}[\|z\|] < \infty$. Si $\mathbb{P}_r$ es una distribución fija en $\mathcal{X}$ y $\mathbb{P}_\theta$ es la distribución en $\mathcal{X}$ definida por g_θ, entonces $W(\mathbb{P}_r, \mathbb{P}_\theta)$ es continua y diferenciable casi seguro.*

Además de todo esto, hay una propiedad más de la distancia de Wasserstein que nos indica que puede funcionar mejor que las otras divergencias:

Proposición 12. *Sea $\mathbb{P}$ una distribución definida en un compacto $\mathcal{X}$ y $(\mathbb{P}_n)_n^\infty$ una sucesión de distribuciones en $\mathcal{X}$. Entonces:*

1. *$W(\mathbb{P}_n, \mathbb{P}) \to 0$ si, y solo si, $\mathbb{P}_n \xrightarrow{d} \mathbb{P}$, donde d denota la convergencia en distribución de variables aleatorias.*

2. *Si $JS(\mathbb{P}_n, \mathbb{P}) \to 0$, entonces $W(\mathbb{P}_n, \mathbb{P}) \to 0$.*

Demostración.

1. Supongamos que $W(\mathbb{P}_n, \mathbb{P}) \to 0$. Sea $f : \mathcal{X} \to \mathbb{R}$ una función continua (y por tanto acotada). Por ser $\mathcal{X}$ compacto, del Teorema de Stone-Weierstrass se sigue que el conjunto de funciones Lispchitz $Lip(\mathcal{X}, \mathbb{R})$ es denso en el conjunto de funciones continuas.

   Así pues, dado $\epsilon > 0$, existe $g : \mathcal{X} \to \mathbb{R}$ Lispchitz tal que:

   $$\|f - g\| < \epsilon$$

   Así,

   $$\begin{aligned} |\int_{\mathcal{X}} f d\mathbb{P}_n - \int_{\mathcal{X}} f d\mathbb{P}| &= |\int_{\mathcal{X}} (f-g) d\mathbb{P}_n - \int_{\mathcal{X}} (f-g) d\mathbb{P} + \int_{\mathcal{X}} g d\mathbb{P}_n - \int_{\mathcal{X}} g d\mathbb{P}| \\ &\leq 2\epsilon + |\int_{\mathcal{X}} g d\mathbb{P}_n - \int_{\mathcal{X}} g \mathbb{P}| \end{aligned} \tag{17}$$

   Si $L > 0$ es la constante de Lipschitz de g, $\frac{g}{L}$ es Lipschitz con constante 1. Así, usando la dualidad de Kantorovich-Rubinstein, existe un $n_0 \in \mathbb{N}$ tal que si $n \geq n_0$ se tiene que:

   $$\begin{aligned} |\int_{\mathcal{X}} \frac{g}{L} d\mathbb{P}_n - \int_{\mathcal{X}} \frac{g}{L} d\mathbb{P}| &\leq \sup_{\|h\|_L \leq 1} \int_{\mathcal{X}} h d\mathbb{P}_n - \int_{\mathcal{X}} h d\mathbb{P} \\ &= W(\mathbb{P}_n, \mathbb{P}) \\ &\leq \epsilon \end{aligned}$$

   Y por lo tanto aplicando esto a (17), nos queda que, para $n \geq n_0$:

   $$|\int_{\mathcal{X}} f d\mathbb{P}_n - \int_{\mathcal{X}} f d\mathbb{P}| \leq \epsilon(2 + L),$$

   obteniendo lo que queríamos.

   Ahora supongamos que $\mathbb{P}_n \to_d \mathbb{P}$. Debido al Teorema de representación de Skorokhod (ver [9]) existen variables aleatorias $Z_n : \Omega \to \mathcal{X}$ y $Z : \Omega \to \mathcal{X}$ definidas en un espacio de probabilidad común $(\Omega, \mathcal{F}, Q)$ tal que la distribución de Z_n es $\mathbb{P}_n$, la de Z es $\mathbb{P}$ y $Z_n \to Z$ casi seguro. Ahora, por ser $\mathcal{X}$ compacto, para cada $w \in \Omega$, se tiene que:

   $$\|Z_n(w)\| \leq M,$$

   para algún $M > 0$. Así, se tiene que:

   $$\|Z_n - Z\| \leq 2M,$$

   y como $Z_n \to Z$ casi seguro, entonces:

   $$\|Z_n - Z\| \to 0 \text{ c.s.}$$

   De esta forma, por el Teorema de la convergencia dominada, se tiene que:

   $$\mathbb{E}\|Z_n - Z\| \to 0.$$

   Además, el vector aleatorio (Z_n, Z) define una distribución en $\mathcal{X} \times \mathcal{X}$ que tiene por marginales $\mathbb{P}_n$ y $\mathbb{P}$, luego se concluye que:

   $$W(\mathbb{P}_n, \mathbb{P}) = \inf_{\gamma \in \Pi(\mathbb{P}_n, \mathbb{P})} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|] \leq \mathbb{E}\|Z_n - Z\| \to 0$$

2. Ya vimos anteriormente que la convergencia en distancia JS es equivalente a la convergencia en variación total, y que la convergencia en variación total implica la convergencia en distribución. El resultado es por tanto consecuencia de esto y el apartado 1 de esta proposición.

□

5. Wasserstein GANs

Como hemos visto, parece adecuado considerar $W(\mathbb{P}_r, \mathbb{P}_\theta)$ como función de pérdida del generador, en vez de la distancia JS. Así, al entrenar el generador intentaríamos reducir la distancia de Wasserstein entre $\mathbb{P}_\theta$ y $\mathbb{P}_r$, que ya sabemos que tiene buenas propiedades. Sin embargo, la definición que hemos dado de la distancia de Wasserstein es demasiado complicada como para usarla en la práctica. Usaremos pues la dualidad de Kantorovich-Rubinstein:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \tag{18}$$

Observemos también que si una función f es 1-Lipschitz, entonces Kf es K-Lipschitz. Por tanto:

$$\sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] = K \cdot W(\mathbb{P}_r, \mathbb{P}_\theta)$$

Así, podemos tratar de estimar la distancia de Wasserstein con una familia de funciones parametrizadas $(f_w)_{w\in\mathcal{W}}$ que sean K-Lipschitz para algún $K > 0$. Esta familia corresponderá con una red neuronal; en el contexto de las GANs sería el discriminador, ahora sin embargo es una función que nos ayudará a calcular la distancia de Wasserstein entre las distribuciones.

Notese que en la expresión (18), en el término $-\mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$ la esperanza está tomada con respecto a la distribución de $\mathbb{P}_\theta$; para generar los objetos de $\mathbb{P}_\theta$ usaremos, como en las GANs, una red neuronal parametrizada $(g_\theta)_{\theta\in\Theta}$. Esto corresponderá con el generador. Con este contexto podemos tratar de aproximar, con un generador g_θ fijo, la ecuación (18):

$$\max_{w\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p_z}[f_w(g_\theta(z))],$$

y de esta manera habríamos estimado $W(\mathbb{P}_r, \mathbb{P}_\theta)$. Entonces ya podemos usar $W(\mathbb{P}_r, \mathbb{P}_\theta)$ como función de pérdida del generador y diferenciarla para poder aplicar algoritmos de optimización basados en el gradiente. Veamos que este proceso tiene sentido:

Teorema 1. *Sea $\mathbb{P}_r$ una distribución fija. Sea $\mathbb{P}_\theta$ la distribución de $g_\theta(Z)$ con Z una variable aleatoria con densidad p y tal que g_θ satisface . Entonces existe una solución $f : \mathcal{X} \to \mathbb{R}$ al problema:*

$$\max_{\|f\|\leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] \tag{19}$$

y además:

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z))]$$

si ambos términos están bien definidos.

Demostración. Definimos

$$\begin{aligned} V(\hat{f}, \theta) &= \mathbb{E}_{x\sim\mathbb{P}_r}[\hat{f}(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[\hat{f}(x)] \\ &= \mathbb{E}_{x\sim\mathbb{P}_r}[\hat{f}(x)] - \mathbb{E}_{z\sim p(z)}[\hat{f}(g_\theta(z))] \end{aligned}$$

donde $\hat{f}$ está en la clase de funciones $\mathcal{F} = \{\hat{f} : \mathcal{X} \to \mathbb{R}, \|\hat{f}\| \leq 1\}$, y $\theta \in \mathbb{R}^d$. La dualidad de Kantorovich-Rubinstein nos dice que:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\hat{f}\in\mathcal{F}} V(\hat{f}, \theta) \tag{20}$$

Fijamos $x_0 \in \mathcal{X}$. Observemos que, si $\hat{f} \in \mathcal{F}$ y $f^* = \hat{f} - f(x_0)$, entonces $V(\hat{f}, \theta) = V(f^*, \theta)$. Por tanto podemos considerar la clase de funciones $\mathcal{F}* = \{f^* \in \mathcal{F}, f^*(x_0) = 0\}$; si el superior (20)se alcanza en $\mathcal{F}^*$, entonces se alcanzará en $\mathcal{F}$.

Sea $x \in \mathcal{X}$. Si $f^* \in \mathcal{F}^*$, se tiene que:

$$|f^*(x)| = |f^*(x) - f^*(x_0)| \leq \|x - x_0\|,$$

luego $\mathcal{F}^*$ es puntualmente acotado. Veamos que es uniformemente equicontinuo. Sea *epsilon* > 0, y escogemos $\delta = \epsilon$. Se tiene que, si $\|x - y\| \leq \delta$:

$$|f^*(x) - f^*(y)| \leq \|x - y\| \leq \delta = \epsilon.$$

Luego el conjunto $\mathcal{F}^* \in \mathcal{C}(\mathcal{X})$ es uniformemente equicontinuo. En virtud del Teorema de Arzelà-Ascoli, $\mathcal{F}^*$ es relativamente compacto. Veamos además que es cerrado: si $(f_n)_{n=1}^{\infty}$ es una sucesión de funciones con $f_n \in \mathcal{F}^*$ y con $f_n \to_{n\to\infty} f$, entonces:

$$\begin{aligned} |f(x) - f(y)| &= |f(x) - f_n(x) + f_n(x) - f_n(y) + f_n(y) - f(y)| \\ &\leq |f(x) - f_n(x)| + |f_n(x) - f_n(y)| + |f_n(y) - f(y)| \\ &\leq \|x - y\||f(x) - f_n(x)| + |f_n(y) - f(y)|, \end{aligned}$$

y tomando el límite cuando $n \to \infty$, queda que:

$$|f(x) - f(y)| \leq \|x - y\|.$$

Luego f es continua y es *Lipschitz* con $\|f\|_L \leq 1$. Además, como $f_n(x_0) = 0 \forall n \in \mathbb{N}$, $f(x_0) = 0$. Luego $f \in \mathcal{F}^*$ y por tanto, $\mathcal{F}^*$ es cerrado.

Por ser $\mathcal{F}^*$ cerrado, coincide con su adherencia, que es compacta por ser relativamente compacto. Por tanto $\mathcal{F}^*$ es compacto y podemos aplicar el Teorema de Weierstrass para deducir que existe un $f \in \mathcal{F}^*$ tal que:

$$\sup_{f^* \in \mathcal{F}^*} V(f^*, \theta) = V(f, \theta)$$

Como ya vimos, esto equivale a que el superior se alcance en $\mathcal{F}$. Por tanto existe la solución al problema (19).

Pasamos a la segunda parte de la proposición. Observemos que, por lo que acabamos de demostrar,

$$\begin{aligned} \nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) &= \nabla_\theta V(f, \theta) \\ &= \nabla_\theta[\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{z\sim p(z)}[f(g_\theta(z))]] \\ &= -\nabla_\theta \mathbb{E}_{z\sim p(z)}[f(g_\theta(z))], \end{aligned}$$

siempre y cuando los términos estén bien definidos. Notese que la última igualdad se da porque $\mathbb{E}_{x\sim\mathbb{P}_r}[f(x)]$ es constante respecto a θ. Probemos pues que:

$$-\nabla_\theta \mathbb{E}_{z\sim p(z)}[f(g_\theta(z))] = -\mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z))] \tag{21}$$

Sabemos que f es 1-Lipschitz, y que g_θ es una función localmente Lipschitziana. Por tanto para cada (z_0, θ_0) existe un $U \in (Z, \mathbb{R}^d)$ abierto con $(z_0, \theta_0) \in U$ y una constante $L(z_0, \theta_0)$ tal que:

$$\|g_\theta(z) - g_{\theta_0}(z_0)\| \leq L(z_0, \theta_0)\|(z - z_0, \theta - \theta_0)\|$$

Así,

$$|f(g_\theta(z)) - f(g_{\theta_0}(z_0))| \leq \|g_\theta(z) - g_{\theta_0}(z_0)\| \leq L(z_0, \theta_0)\|(z - z_0, \theta - \theta_0)\|,$$

de donde se deduce que $f(g_\theta(z))$ es localmente Lipschitziana en (z, θ) con las mismas constantes $L(z, \theta)$ para cada par (z, θ). Por el teorema de Rademacher, $f(g_\theta(z))$ es diferenciable casi siempre en las variables (z, θ).

Así, el conjunto $A = \{(z, \theta) : f(g_\theta)\text{no es diferenciable}\}$ tiene medida nula. Nos interesa analizar el caso en el que el término derecho de (21) está bien definido; como A tiene medida nula, para casi todo θ la función $z \to \nabla_\theta f(g_\theta(z))$ está bien definida casi siempre. Por tanto sea θ_0 tal que $z \to \nabla_\theta f(g_\theta)(\theta_0)$ está definida para casi todo z.

Puesto que $p(z)$ tiene densidad, si un conjunto tiene medida de Lebesgue 0 entonces su medida con respecto a $p(z)$ también es 0. Por tanto el conjunto de medida de Lebesgue nula donde $\nabla_\theta f(g_\theta(z))(\theta_0)$ no está definida también tiene medida nula con respecto a $p(z)$. Por tanto $\nabla_\theta f(g_\theta(z))(\theta_0)$ está definida $p(z)$-casi siempre. Además, observemos que para cierto U abierto con $(z, \theta_0) \in U$:

$$|f(g_\theta(z)) - f(g_{\theta_0}(z))| \leq L(z, \theta_0)\|g_\theta(z) - g_{\theta_0}(z)\|$$

Demostremos ahora que también $\|\nabla_\theta f(g_\theta(z))(\theta_0)\| \leq L(z, \theta_0)$. Por la definición de diferenciabilidad, existe una función $\epsilon : Z \times \mathbb{R}^d \to \mathbb{R}$ tal que $\lim_{\theta \to \theta_0} \epsilon(z, \theta) = 0$ y tal que:

$$\nabla_\theta f(g_\theta(z)) \cdot (\theta - \theta_0) = f(g_\theta(z)) - f(g_{\theta_0}(z)) - \epsilon(z, \theta)\|\theta - \theta_0\|$$

Por lo que, si $(z, \theta) \in U$:

$$\begin{aligned} |\nabla_\theta f(g_\theta(z)) \cdot (\theta - \theta_0)| &\leq |f(g_\theta(z)) - f(g_{\theta_0}(z))| + |\epsilon(z, \theta)|\|\theta - \theta_0\| \\ &\leq (L(z, \theta_0) + |\epsilon(z, \theta)|)\|\theta - \theta_0\| \end{aligned}$$

Ahora si $\theta \neq \theta_0$:

$$|\nabla_\theta f(g_\theta(z)) \cdot \frac{(\theta - \theta_0)}{\|\theta - \theta_0\|}| \leq (L(z, \theta_0) + |\epsilon(z, \theta)|)$$

Como esta desigualdad se cumple $\forall \theta \in U, \theta \neq \theta_0$ y los vectores $\frac{(\theta - \theta_0)}{\|\theta - \theta_0\|}$ tienen norma 1, entonces hemos demostrado que:

$$\|\nabla_\theta f(g_\theta(z))\| \leq (L(z, \theta_0) + |\epsilon(z, \theta)|)$$

Y tomando límite cuando $\theta \to \theta_0$:

$$\|\nabla_\theta f(g_\theta(z))(\theta_0)\| \leq L(z, \theta_0),$$

que es lo que queríamos. Así llegamos a las siguientes desigualdades:

$$\mathbb{E}_{z \sim p(z)}[\|\nabla_\theta f(g_\theta(z))(\theta_0)\|] \leq \mathbb{E}_{z \sim p(z)}[L(z, \theta_0)] < \infty$$

,la última desigualdad por hipótesis. De aquí se deduce que está bien definida $\mathbb{E}_{z \sim p(z)}[\|\nabla_\theta f(g_\theta(z))(\theta_0)\|]$. Podemos considerar entonces:

$$\frac{\mathbb{E}_{z \sim p(z)}[f(g_\theta(z))] - \mathbb{E}_{z \sim p(z)}[f(g_{\theta_0}(z))] - \langle(\theta - \theta_0), \mathbb{E}_{z \sim p(z)}[\nabla_\theta f(g_\theta(z))(\theta_0)]\rangle}{\|\theta - \theta_0\|}, \tag{22}$$

y por diferenciabilidad, si esto tiende hacia 0 cuando $\theta \to \theta_0$, entonces habremos demostrado lo que queríamos. Por la linealidad de la esperanza la expresión anterior es igual a:

$$\mathbb{E}_{z \sim p(z)}\left[\frac{f(g_\theta(z)) - f(g_{\theta_0}(z)) - \langle(\theta - \theta_0), \nabla_\theta f(g_\theta(z))(\theta_0)\rangle}{\|\theta - \theta_0\|}\right]$$

Además, aplicando Cauchy-Schwarz y usando que $f(g_\theta(z))$ es localmente Lipschitziana:

$$\begin{aligned}|\frac{f(g_\theta(z)) - f(g_{\theta_0}(z)) - \langle(\theta-\theta_0), \nabla_\theta f(g_\theta(z))(\theta_0)\rangle}{\|\theta-\theta_0\|}| &\leq \frac{\|\theta-\theta_0\|L(z,\theta_0) + \|\theta-\theta_0\|\|\nabla_\theta f(g_\theta(z))(\theta_0)\|}{\|\theta-\theta_0\|} \\ &= L(z,\theta_0) + \|\nabla_\theta f(g_\theta(z))(\theta_0)\| \\ &\leq 2L(z,\theta_0)\end{aligned}$$

Por hipótesis, $\mathbb{E}_{z\sim p(z)}[L(z,\theta_0)] < \infty$, luego en virtud del teorema de la convergencia dominada, la expresión (22) tiende hacia 0 cuando $\theta \to \theta_0$. Así, queda demostrado que:

$$-\nabla_\theta \mathbb{E}_{z\sim p(z)}[f(g_\theta(z))] = -\mathbb{E}_{z\sim p(z)}[\nabla_\theta f(g_\theta(z))],$$

que es lo que queríamos. □

Ya hemos dicho que para aproximar la solución f que servirá para calcular la distancia de Wasserstein usaremos una red neuronal parametrizada $(f_w)_{w\in\mathcal{W}}$. Pero todavía necesitamos asegurar que las funciones f_w son K-Lispchitz para algún K (que no dependa de w). La forma en la que lo haremos será forzando a que cada parámetro esté en un multi-intervalo compacto (por ejemplo $\mathcal{W} = [-0{,}01, 0{,}01]^l$):

Teorema 2. *Sea $(f_w)_{w\in W}$ una red neuronal parametrizada con $f_w : \mathcal{X} \to \mathbb{R}$ y parámetros en un compacto $\mathcal{W}$. Supongamos también que las funciones de activación utilizadas tienen Jacobiano acotado, (como es el caso para las funciones de activación más comunes: ReLU, sigmoide, softmax, tanh, etc.), entonces f_w es K-Lipschitz para alguna constante K y para todos $w \in \mathcal{W}$.*

Demostración. Si existe $M > 0$ tal que $\|\nabla f_w\|_\infty \leq M$ entonces por el Teorema del valor medio, para cada $x, y \in \mathcal{X}$ existe un $c \in \mathcal{X}$ tal que:

$$|f_w(x) - f_w(y)| = |\nabla f_w(c) \cdot (x-y)| \leq \|\nabla f_w\|_\infty \|x-y\| \leq M\|x-y\|$$

de donde se deduce que f_w es Lipschitz. Veamos entonces que f_w tiene gradiente acotado.

Sea L el número de capas. Sabemos pues que $\nabla_x f_w(x) = \prod_{i=1}^{H} W_i D_i(x)$ donde D_i son los Jacobianos de las funciones de activación y los W_i las matrices de pesos. Por ser $\mathcal{W}$ compacto, existe un $K > 0$ tal que $\|W_i\|_\infty \leq K$, $i = 1, ..., L$, y por estar los D_i acotados, existe $M > 0$ tal que $\|D_i\|_\infty$, $i = 1, ..., L$. Entonces:

$$\|\nabla_x f_w\|_\infty \leq \prod_{i=1}^{L} \|W_i\|_\infty \|D_i\|_\infty \leq (KM)^L,$$

basta concluir observando que las cotas $\|W_i\|_\infty \leq K$, no dependen de los pesos w.

□

5.1. Algoritmo WGAN

Ya tenemos todo lo que necesitamos para plantear el algoritmo WGAN. El algoritmo se basa en usar una red neuronal parametrizada g_θ para generar los objetos y como función de pérdida la distancia de Wasserstein. Para calcularla necesitaremos entrenar una red f_w forzando sus parámetros a un compacto.

Definimos n_f como el número de iteraciones que realizamos de descenso de gradiente cada vez que entrenamos f_w, m el tamaño del minilote al usar el descenso del gradiente, α la tasa

de aprendizaje, c el parámetro que define el compacto $\mathcal{W} = [-c, c]^l$. Las redes neuronales se inicializan con parámetros θ_0 y w_0 elegidos de antemano (o inicializados aleatoriamente).

Llamamos $clip(w, -c, c)$ a la función que actúa componente a componente sobre los parámetros w de la siguiente manera:

$$clip(w_k, -c, c) = \begin{cases} w_k & \text{if } w_k \in [-c, c] \\ c & \text{if } w_k > c \\ -c & \text{if } w_k < c \end{cases}$$

De esta manera forzamos a que los parámetros estén en un compacto. Mientras no haya convergido θ:

1. Para t = 0,...,n_f hacer:
   - *a*) Obtener $x_1, ..., x_m$, muestras de la distribución $\mathbb{P}_r$.
   - *b*) Obtener $z_1, ..., z_m$, muestras de $p(z)$.
   - *c*) $w \to w + \alpha \nabla_w [\frac{1}{m} \sum_{i=1}^{m} f_w(x_i) - \frac{1}{m} f_w(g_\theta(z_i))]$
   - *d*) $w \to clip(w, -c, c)$
2. Obtener $z_1, ..., z_m$ un minilote de muestras de $p(z)$.
3. $\theta \to \theta - \alpha \nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z_i))$

Notese que mientras que para f_w hacemos varias iteraciones de descenso de gradiente, para g_θ solo hacemos una. El motivo es que, debido a que la distancia de Wasserstein es continua y diferenciable casi siempre, nos interesa entrenar f_w lo máximo posible para obtener una mejor estimación del gradiente de la distancia. Sin embargo, cada vez que cambiamos los parámetros θ cambia la distribución $\mathbb{P}_\theta$ y por tanto tenemos que volver a aproximar otra vez $W(\mathbb{P}_r, \mathbb{P}_\theta)$.

6. Experimentos

En este apartado mostraremos algún resultado empírico de los modelos. El objetivo del experimento es visualizar el problema del desvanecimiento del gradiente de las GANs en un conjunto de datos sencillo, y ver cómo las WGAN solucionan el problema para ese caso.

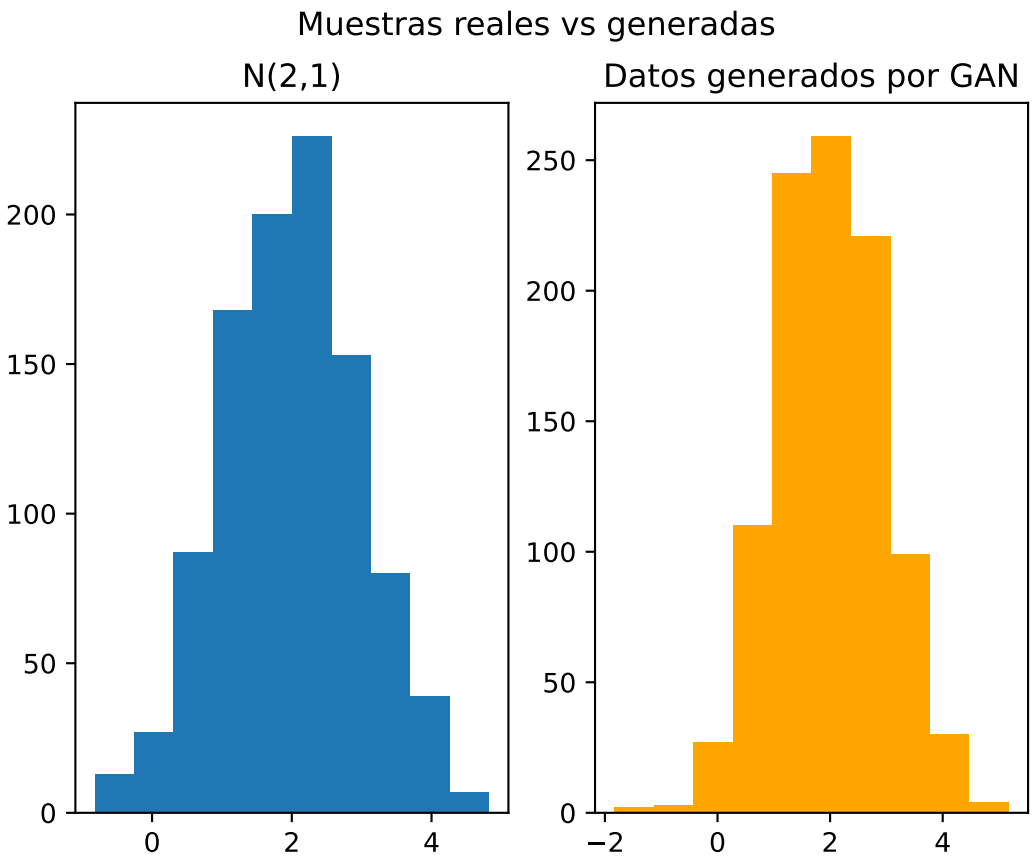
Como ya se ha expuesto anteriormente, en [6] se explica que cuando la distribución inicial del generador y la distribución objetivo tienen intersección despreciable, el aprendizaje de las GANs se dificulta mucho debido a que el gradiente del generador se anula. Intuitivamente, el discriminador da muy buenos resultados muy rápido y siempre reconoce como falsos las generaciones del generador, de tal forma que no le da ninguna información útil para generar objetos que se acerquen más a $\mathbb{P}_r$.

La práctica habitual para el muestreo en el espacio latente es usar una normal multivariante de media 0 y matriz de covarianzas identidad. Como nuestro ejemplo será en una dimensión, nuestra probabilidad en $Z = \mathbb{R}$ será simplemente una $N(0, 1)$.

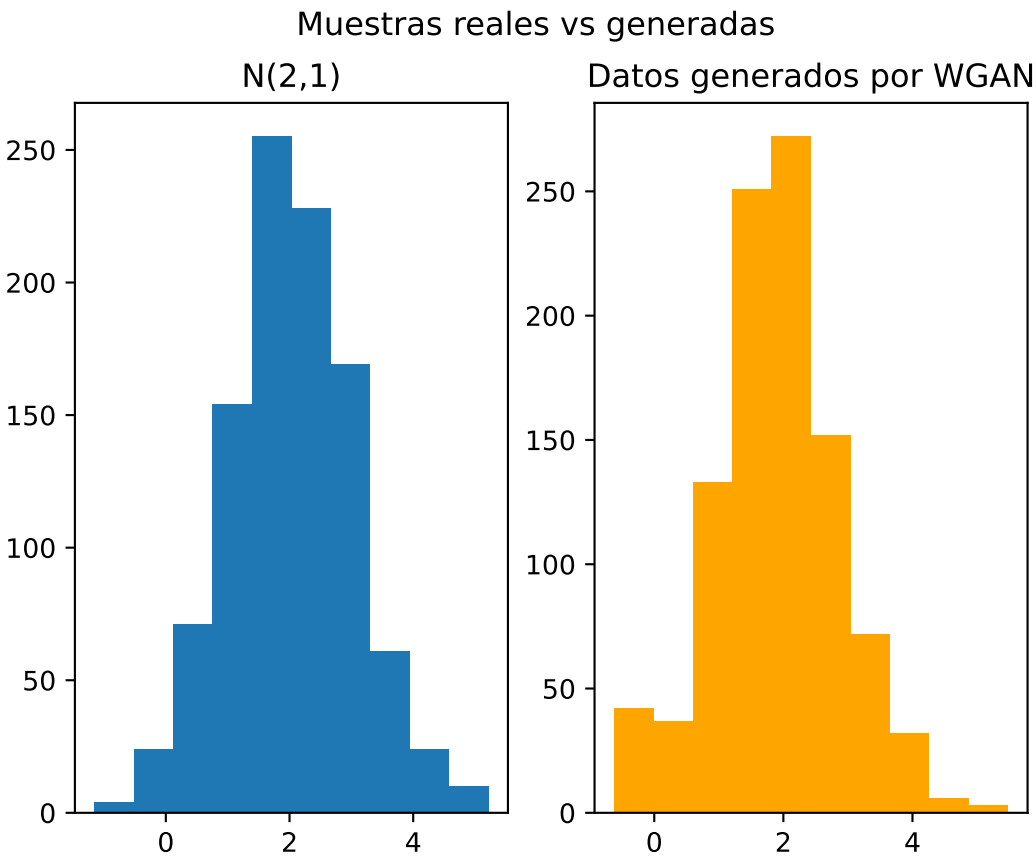
Primero, veremos cómo las GAN aprenden a generar datos de una $N(2, 1)$. Notese que lo único que deberá aprender es una aplicación lineal de la normal $N(0, 1)$. Mostraremos también cómo las WGAN son capaces de hacerlo.

Nuestro siguiente ejemplo será generar observaciones de una normal $N(40, 1)$. Aunque parezca un problema similar, el hecho de que ésta distribución esté alejada de la distribución normal hace que las GAN tengan problemas para aprenderla. Ilustraremos el fenómeno del desvanecimiento del gradiente en este caso, y veremos que las WGAN, sin embargo, sí que aprenden la distribución alejada.

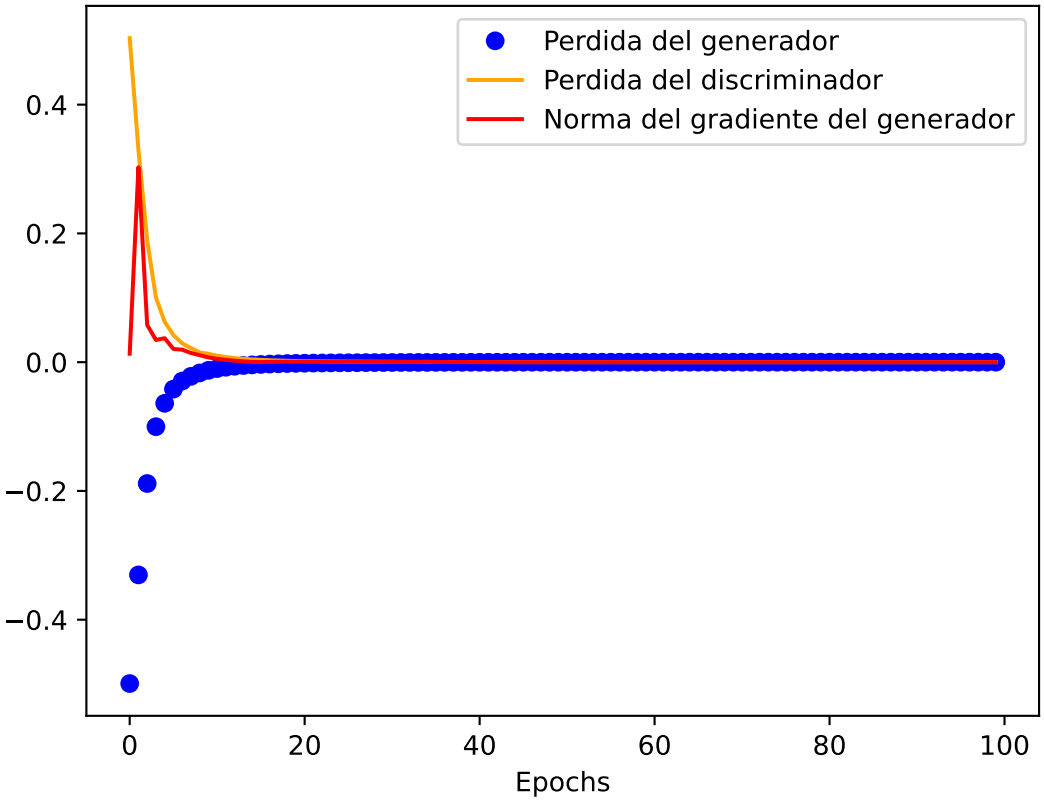
Entrenamos unas GAN para generar muestras de una normal $N(2, 1)$. Se han usado redes con pocas capas. Las funciones de activación elegidas para el generador han sido todas lineales, puesto que lo único que tiene que aprender es una función lineal. Para el discriminador, la función de activación de la última capa es una sigmoide, ya que estamos ante un problema de clasficación. El resultado se puede ver en la siguiente figura:



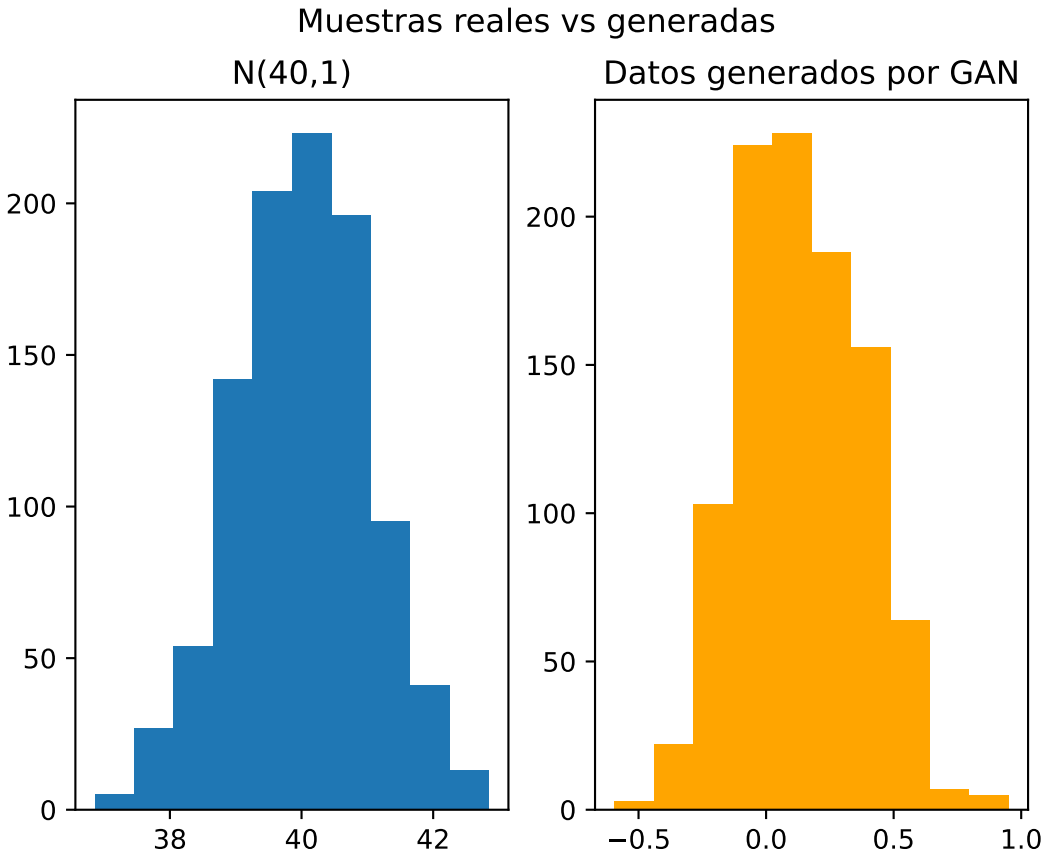
El generador ha aprendido la distribución normal. Implementamos el mismo ejemplo con las WGAN. Notese que ahora la capa de salida del critico no es una sigmoide, ya que ahora estamos aproximando la función $f : \mathcal{X} \to \mathbb{R}$ de la dualidad de Kantorovich-Rubinstein; ya no es un problema de clasificación. Para el crítico, hemos usado una red neuronal más grande, con dimensión mayor en cada capa, y con funciones de activación ReLU. Esto se debe a que un mejor crítico da una mejor aproximación a la distancia de Wasserstein y el gradiente del generador será más preciso. Es por estos motivos, como se ve en [10], que es conveniente entrenar el crítico hasta la optimalidad si es posible antes de actualizar el generador. En este caso, hemos realizado 20 iteraciones del crítico por cada actualización de los pesos del generador.



Ahora, probemos a aproximar una normal $N(40, 1)$ con las GAN. Se ha hecho el experimento con las redes de antes, y también fracasaban. Hemos elegido unas redes más grande y hemos entrenado las GANs por más tiempo para ilustrar que ese no es el problema en este caso. En la siguiente figura se muestran las gráficas de las funciones de pérdida y de la norma del gradiente del generador a lo largo del proceso de aprendizaje:

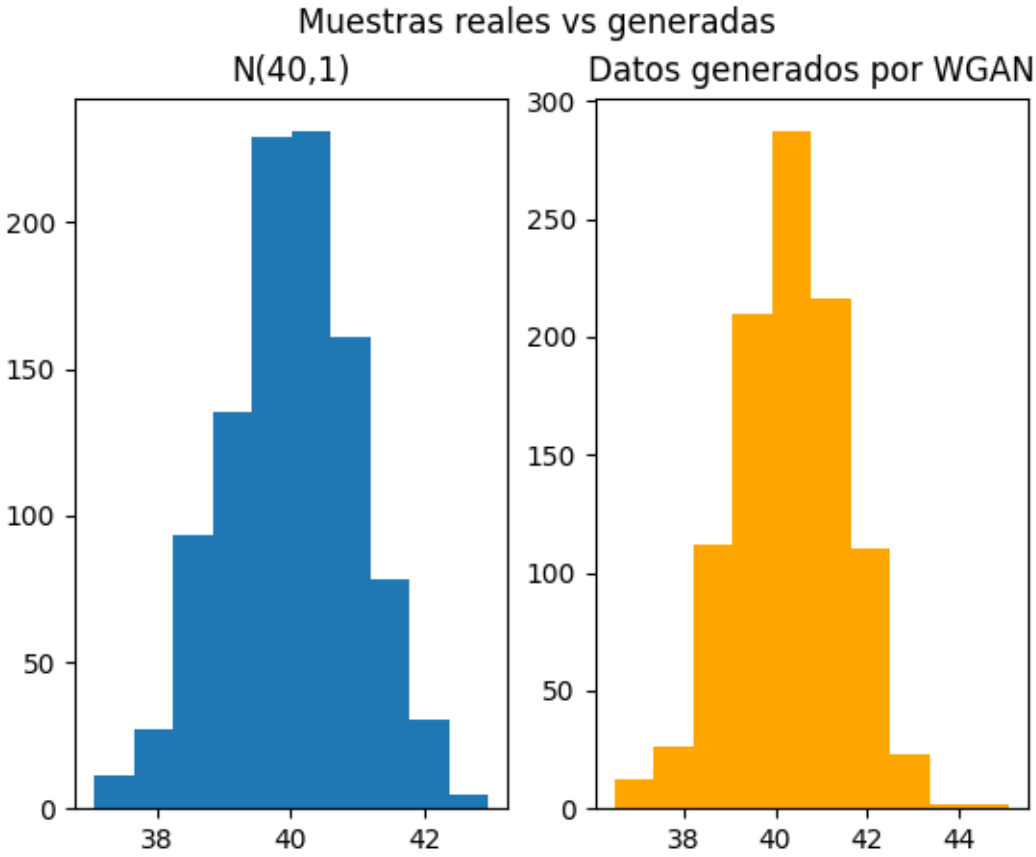


La norma del gradiente ha acabado en 0 muy rápidamente. Esto impide que los algoritmos basados en el descenso del gradiente puedan avanzar. Comparemos los objetos generados:

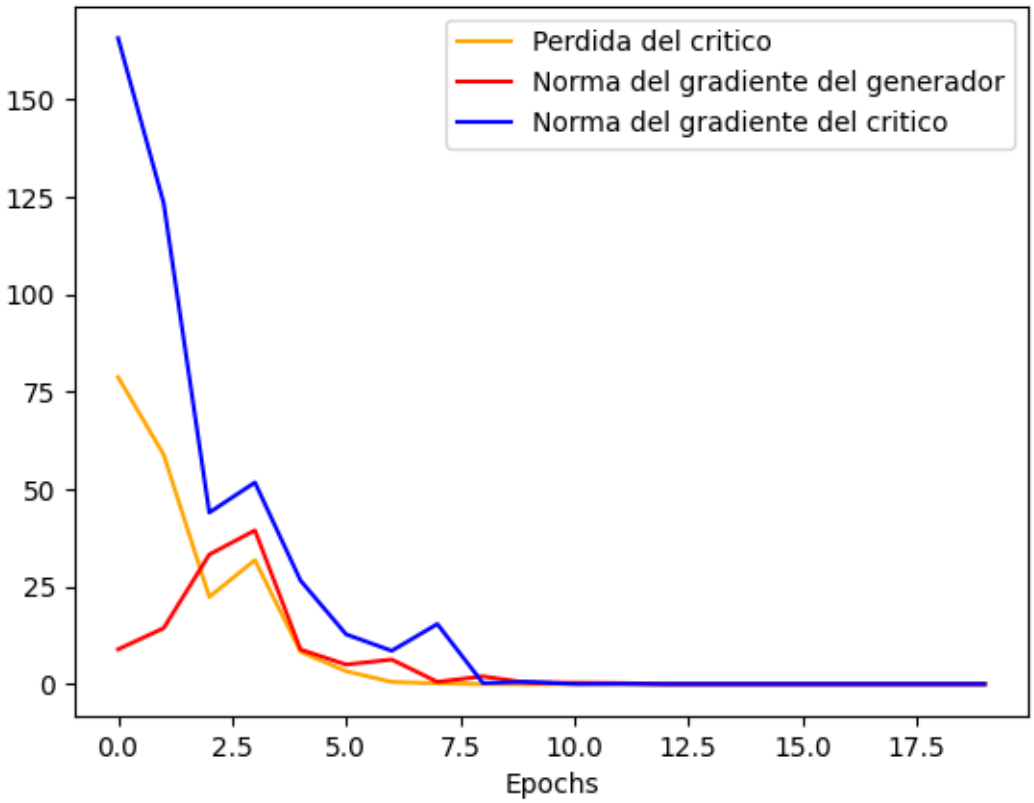


Como habíamos adelantado en el capítulo de las GANs, en este caso las GANs no han aprendido la distribución $N(40, 1)$. Vemos que se han quedado estancadas alrededor de la distribución inicial.

Probemos el mismo ejemplo con las WGAN. En este caso si que hemos usado la misma estructura de las redes que habíamos usado para el ejemplo de la $N(2, 1)$ con las WGAN anteriormente.



Observamos que las muestras generadas por las WGAN si parecen de una distribución $N(40, 1)$. Mostramos el mismo gráfico que antes de la norma del gradiente:



Vemos que el aprendizaje ha ido correctamente y la norma del gradiente se anula cuando ya ha convergido en la distribución objetivo. Las WGAN han el fenómeno de desvanecimiento del gradiente. Como hemos visto anteriormente, esto se debe a que la distancia de Wasserstein, aproximada por la pérdida del crítico, sí que da una información valiosa para aprender la distribución, a diferencia de la distancia de Jensen-Shannon, que hace que el entrenamiento no sea fácil para las GANs en estas condiciones.

Apéndices

A. Teorema de Radon-Nikodym y Teorema de Rademacher

Definición 1. *Sea $\mathcal{M}$ una σ-álgebra en $\mathcal{X}$. Consideramos ν una medida positiva sobre $\mathcal{M}$ y λ, una medida compleja o positiva sobre $\mathcal{M}$. Se dice que λ es absolutamente continua con respecto a ν, si $\lambda(A) = 0\ \forall A \in \mathcal{M}$ tal que $\nu(A) = 0$. Si λ es absolutamente continua con respecto a ν, se escribe: $\lambda << \nu$.*

Definición 2. *Sean ν, λ medidas en $\mathcal{M}$. Supongamos que existe $A \in \mathcal{M}$ tal que $\nu(E) = \nu(E \cap A)\ \forall E \in \mathcal{M}$. Se dice entonces que ν está concentrada en A.*
Si ν está concentrada en A y λ está concentrada en B con A y B disjuntos, entonces se dice que son mutuamente singulares, y se escribe: $\nu \perp \lambda$.

Proposición 13. *Sea ν una medida positiva, σ-finita en la σ-álgebra $\mathcal{M}$. Sea λ una medida compleja en $\mathcal{M}$. Entonces existen dos medidas complejas, λ_a, λ_s en $\mathcal{M}$ tales que:*

$$\lambda = \lambda_a + \lambda_s, \quad \lambda_a << \nu, \quad \lambda_s \perp \nu$$

El par (λ_a, λ_s) es la descomposición de Lesbesgue de λ, y es única.

Teorema 3 (Radon-Nikodym)**.** *En las condiciones de la proposición anterior, existe una única $g \in L^1(\nu)$ tal que:*

$$\lambda_a(E) = \int_E h d\nu$$

para todo $E \in \mathcal{M}$. En particular, si $\lambda << \nu$, entonces:

$$\lambda(E) = \int_E h d\nu$$

A la función h se le denomina la derivada de Radon-Nikodym de λ_a con respecto a ν. Suele escribirse $h = \frac{d\lambda_a}{\nu}$.

La demostración de estos resultados junto con más detalles sobre la derivada de Radon-Nikodym puede encontrarse en [11].

Teorema 4 (Teorema de Rademacher)**.** *Sea $f : \mathbb{R}^n \to \mathbb{R}^m$ una función localmente Lipschitziana. Entonces f es diferenciable casi siempre.*

La demostración de este resultado puede encontrarse en [8].

B. Implementaciones

En este apéndice exponemos el código con el que hemos generado los gráficos de la sección experimentos. Se ha usado Python para hacer las implementaciones, con los siguientes paquetes: Tensorflow y Keras para trabajar con redes neuronales, matplotlib para hacer algún gráfico y numpy para manipular arrays y hacer todo tipo de operaciones.

Para entrenar una red neuronal en Tensorflow, usualmente definimos su estructura, compilamos el modelo (con model.compile) teniendo en cuenta que método de optimización vamos a usar y qué función de pérdida, y luego usamos el método modelo.fit. Sin embargo el proceso es algo más complejo al implementar las GAN o WGAN, ya que estamos entrenando dos redes a la vez. Para definir un modelo.fit conveniente necesitaremos reescribir cómo Tensorflow va a realizar un paso del entrenamiento. Los detalles de cómo se hace esto se pueden consultar en [12]. En el apartado final también hay una implementación de las GAN, que se ha usado de referencia para la implementación de las GANs. Para consultar cualquier otra cosa se ha consultado la documentación de Tensorflow.

Primero la implemntacion la clase de las GANs:

```
#importamos todas las librerias necesarias

import tensorflow as tf
from tensorflow import keras
from keras import models
from keras import layers
import keras.backend as K

import matplotlib.pyplot as plt

import numpy as np

#Definimos la clase CustomGAN. Esto nos ayudara a crear un modelo para entrenar
    GANs. Cuando escribimos (keras.Model) significa que estamos heredando los
    metodos de la clase Model de keras: de esta manera reescribimos lo necesario
    para cambiar el modelo como nos convenga.

class CustomGAN(keras.Model):
        #metodo para inicializar el modelo. Se necesitan la red del
            discriminador y el generador, y la dimension del espacio latente
    def __init__(self, discriminador, generador, dim_latente):
        super(CustomGAN, self).__init__()
        #con super usamos un metodo heredado de la clase Model. En este caso es
            necesario que todo objeto de la clase Model se inicialice con el
            metodo __init__()
        self.discriminador = discriminador
        self.generador = generador
        self.dim_latente = dim_latente

        #metodo para compilar el modelo. Se necesitan las funciones de perdida
            de ambas redes y sus metodos de optimizacion
    def compile(self, opt_disc, opt_gen, l_disc, l_gen):
        super(CustomGAN, self).compile()
        self.opt_disc = opt_disc
        self.opt_gen = opt_gen
        self.l_disc = l_disc
        self.l_gen = l_gen
```

```
        #sobreescribimos train_step. De esta manera al llamar a modelo.fit se
            entrenaran las redes usando el algoritmo que nosotros determinemos
            aqui

    def train_step(self, x_real):
        #en este caso, x_real seran minilotes de tamano batch_size (que
            introduciremos al llamar model.fit) de los datos "reales"
        batch_size = tf.shape(x_real)[0]

        #hacemos el muestreo del espacio latente
        vector_latente = tf.random.normal(shape = (batch_size, self.dim_latente)
            )

        #para calcular el gradiente de la red, usaremos GradientTape. con el
            metodo tape.gradient() calcularemos las derivadas de las operaciones
            que se hayan introducido previamente en "with".
        with tf.GradientTape() as tape:
            x_fake = self.generador(vector_latente)
            prob_x_fake = self.discriminador(x_fake)
            prob_x_real = self.discriminador(x_real)
            perdida_disc = self.l_disc(prob_x_fake, prob_x_real)

        #calculamos el gradiente de la funcion de perdida con respecto a los
            parametros del discriminador
        d_grads = tape.gradient(perdida_disc, self.discriminador.
            trainable_weights)

        #se aplica el gradiente al metodo de optimizacion. En este paso es
            cuando actualizamos los pesos del discriminador
        self.opt_disc.apply_gradients(
            zip(d_grads, self.discriminador.trainable_weights)
        )

        #ahora vamos con la segunda parte de las GAN. Se va a actualizar el
            generador. Volvemos a sacar muestras del espacio latente
        vector_latente = tf.random.normal(shape = (batch_size, self.dim_latente)
            )

        #hacemos el mismo proceso con la funcion de perdida del generador
        with tf.GradientTape() as tape:
            x_fake = self.generador(vector_latente)
            prob_x_fake = self.discriminador(x_fake)
            perdida_gen = self.l_gen(prob_x_fake)

        g_grads = tape.gradient(perdida_gen, self.generador.trainable_weights)
        self.opt_gen.apply_gradients(
            zip(g_grads, self.generador.trainable_weights)
        )

        #calculamos las normas de los gradientes, algo que usaremos para
            estudiar el fenomeno del desvanecimiento del gradiente. gradient_norm
             es un metodo que implementaremos luego
        g_norm = gradient_norm(g_grads)
        d_norm = gradient_norm(d_grads)

        #devolvemos un diccionario con datos relevantes del proceso
        return {"perdida_gen": perdida_gen, "perdida_disc": perdida_disc, "
            gen_grad_norm": g_norm,"disc_grad_norm": d_norm}
```

Pasamos a dar la implementación del algoritmo WGAN. Es más o menos similar en estructura. Llamaremos ahora crítico a la función Lipschitz que nos ayuda a calcular la distancia de

Wasserstein, y que cumple el rol del discriminador en las GANs.

```
class CustomWGAN(keras.Model):
    #A parte de las redes y la dimension del espacio latente, se introduce c,
       que ser  la constante a partir de la cual meteremos los pesos del
       critico en [-c,c]. Tambien se inicializa n_pasos, que sera el numero de
       iteraciones de descenso del gradiente que hacemos en el critico antes de
       actualizar el generador. Recordese que en las WGAN se recomienda entrenar
        al critico hasta ser optimo antes de actualizar el generador.

    def __init__(self, critico, generador, dim_latente, c, n_pasos):
        super(CustomWGAN, self).__init__()
        self.critico = critico
        self.generador = generador
        self.dim_latente = dim_latente
        self.c = c
        self.n_pasos = n_pasos

    def compile(self, opt_crit, opt_gen, l_disc, l_gen):
        super(CustomWGAN, self).compile()
        self.opt_crit = opt_crit
        self.opt_gen = opt_gen
        self.l_crit = l_crit
        self.l_gen = l_gen

    def train_step(self, x_real):
        batch_size = tf.shape(x_real)[0]

        for i in range(self.n_pasos):
            vector_latente = tf.random.normal(shape = (batch_size, self.
               dim_latente))

            with tf.GradientTape() as tape:
                x_fake = self.generador(vector_latente)
                crit_x_fake = self.critico(x_fake)
                crit_x_real = self.critico(x_real)
                perdida_crit = self.l_crit(crit_x_fake, crit_x_real)

            c_grads = tape.gradient(perdida_crit, self.critico.trainable_weights
               )
            self.opt_crit.apply_gradients(
                zip(c_grads, self.critico.trainable_weights)
                )

            #forzamos los pesos w del critico a estar en el intervalo [-c,c]
               para convertir a f_w en una funcion Lispchitz
            for w in self.critico.trainable_variables:
                w.assign(tf.clip_by_value(w, -self.c, self.c))

        vector_latente = tf.random.normal(shape = (batch_size, self.dim_latente)
           )

        with tf.GradientTape() as tape:
            x_fake = self.generador(vector_latente)
            crit_x_fake = self.critico(x_fake)
            perdida_gen = self.l_gen(crit_x_fake)

        g_grads = tape.gradient(perdida_gen, self.generador.trainable_weights)
        self.opt_gen.apply_gradients(
            zip(g_grads, self.generador.trainable_weights)
        )
```

```
        g_norm = gradient_norm(g_grads)
        c_norm = gradient_norm(c_grads)

        return {"perdida_gen": perdida_gen, "perdida_crit": perdida_crit, "
            gen_grad_norm": g_norm,"crit_grad_norm": c_norm}
```

El código que genera muestras de una N(2,1) con las GANs:

```
#definimos los modelos del generador y discriminador. Las redes escogidas son
    sencillas con una sola capa intermedia. Notese que la capa de salida del
    discriminador es una sigmoide pues modeliza una probabilidad
generador = models.Sequential()
generador.add(layers.Dense(25, input_shape = (1,)))
generador.add(layers.Dense(15))
generador.add(layers.Dense(1))

discriminador = models.Sequential()
discriminador.add(layers.Dense(20,activation = 'relu', input_shape = (1,)))
discriminador.add(layers.Dense(1,activation = 'sigmoid'))

#definimos las funciones de perdida tal como hemos visto en la teoria. Tambien
    definimos la funcion que calcula la norma del gradiente de una red, que hemos
     visto que usamos en el algoritmo.

def gradient_norm(grads):
    ssq = [K.sum(K.square(g)) for g in grads]
    norm = K.sqrt(sum(ssq))
    return norm

def l_generador(prob_x_fake):
    return tf.reduce_mean(tf.math.log(1-prob_x_fake), axis=None)

def l_discriminador(prob_x_fake, prob_x_real):
    l_real = -tf.math.log(prob_x_real)
    l_fake = -tf.math.log(1-prob_x_fake)
    return tf.reduce_mean(l_real + l_fake, axis=None)
```

Ahora definimos y compilamos nuestro modelo CustomGAN:

```
gan = CustomGAN(discriminador, generador, 1)
gan.compile(tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9),
            tf.keras.optimizers.Adam(learning_rate=0.0001, beta_1=0.9),
            l_discriminador,
            l_generador)

#Definimos nuestro conjunto de datos. Nos gustaria que el generador aprenda a
    crear muestras similares a estas:

nu_0 = 2
sigma= 1
X = np.random.normal(nu_0, sigma,10000)

#entrenamos el modelo.
gan.fit(X, epochs = 30 , batch_size = 128, verbose = 0)

#creamos un grafico que nos muestre 100 muestras del generador, en azul, y en
    naranja, 100 muestras de una normal N(2,1)
vector_latente = tf.random.normal(shape = (100, 1))
Y = generador(vector_latente).numpy()

fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Muestras reales vs generadas')
```

```
ax1.hist(X[0:1000])
ax1.set_title('N(2,1)')
ax2.hist(Y[:,0], color = 'orange')
ax2.set_title('Datos generados por GAN')
```

El código para generar muestras de la N(2,1) con las WGAN:

```
generador_w = models.Sequential()
generador_w.add(layers.Dense(25, input_shape = (1,)))
generador_w.add(layers.Dense(15))
generador_w.add(layers.Dense(1))

critico = models.Sequential()
critico.add(layers.Dense(100,activation = 'relu', input_shape = (1,)))
critico.add(layers.Dense(100,activation = 'relu'))
critico.add(layers.Dense(1))

#definimos las funciones de perdida en las WGAN

def l_crit(crit_x_fake, crit_x_real):
    return -tf.reduce_mean(crit_x_fake, axis= None)+tf.reduce_mean(crit_x_real,
        axis= None)

def l_generador(crit_x_fake):
    return -tf.reduce_mean(crit_x_fake, axis = None)

#compilamos y entrenamos el modelo. El optimizador RMSprop es el recomendado en
    el paper de las WGAN
wgan = CustomWGAN(critico, generador_w, 1, 0.1,20)
wgan.compile(tf.keras.optimizers.RMSprop(learning_rate=-0.001),
             tf.keras.optimizers.RMSprop(learning_rate=0.001),
             l_crit,
             l_generador)

wgan.fit(X, epochs=20, batch_size=128, verbose=0)
```

El código para intentar generar muestras de una N(40,1) con GANs:

```
nu_1 = 40
sigma= 1

X_lejano = np.random.normal(nu_1, sigma,10000)

h = gan.fit(X_lejano, epochs= 100, batch_size= 64, verbose= 0)

#obtenemos los valores de las funciones de perdida, y del gradiente del
    generador:
h_dict = h.history
perdida_gen = h_dict['perdida_gen']
perdida_disc = h_dict['perdida_disc']
gen_grad_norm = h_dict['gen_grad_norm']

plt.plot(range(100), perdida_gen, 'bo', label = 'Perdida del generador')
plt.plot(range(100), perdida_disc, 'orange', label = 'Perdida del discriminador'
    )
plt.plot(range(100), gen_grad_norm, 'red', label = 'Norma del gradiente del
    generador')
plt.xlabel('Epochs')
plt.legend()
plt.savefig('desvanecimiento')
```

El c digo para generar muestras de una N(40,1) con WGAN. La generaci n de gr ficos tiene un c digo similar al anterior.

```
\begin{lstlisting}[language=Python]
generador_w = models.Sequential()
generador_w.add(layers.Dense(25, input_shape = (1,)))
generador_w.add(layers.Dense(15))
generador_w.add(layers.Dense(1))

critico = models.Sequential()
critico.add(layers.Dense(100,activation = 'relu', input_shape = (1,)))
critico.add(layers.Dense(100,activation = 'relu'))
critico.add(layers.Dense(1))

wgan.fit(X_lejano, epochs= 20, batch_size=128, verbose=0)
```

Referencias

[1] D. Endres y J. Schindelin, "A new metric for probability distributions", *IEEE Transactions on Information Theory*, vol. 49, n.º 7, págs. 1858-1860, 2003. DOI: 10.1109/TIT.2003.813506.

[2] P. Massart y J. Picard, *Concentration Inequalities and Model Selection:* ép. Concentration Inequalities and Model Selection: Ecole D'Eté de Probabilités de Saint-Flour XXXIII - 2003. Springer, 2007, ISBN: 9783540484974. dirección: https://books.google.es/books?id=O_gZAQAAIAAJ.

[3] H. Royden y P. Fitzpatrick, *Real Analysis*. Prentice Hall, 2010, ISBN: 9780131437470. dirección: https://books.google.es/books?id=0Y5fAAAACAAJ.

[4] S. Bubeck y col., "Convex optimization: Algorithms and complexity", *Foundations and Trends® in Machine Learning*, vol. 8, n.º 3-4, págs. 231-357, 2015.

[5] L. Bottou y col., "Stochastic gradient learning in neural networks", *Proceedings of Neuro-Nımes*, vol. 91, n.º 8, pág. 12, 1991.

[6] M. Arjovsky y L. Bottou, "Towards principled methods for training generative adversarial networks", *arXiv preprint arXiv:1701.04862*, 2017.

[7] H. Thanh-Tung y T. Tran, "Catastrophic forgetting and mode collapse in gans", en *2020 international joint conference on neural networks (ijcnn)*, IEEE, 2020, págs. 1-10.

[8] C. Villani, *Optimal Transport: Old and New*, ép. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008, ISBN: 9783540710509.

[9] P. Billingsley, *Probability and Measure*, ép. Wiley Series in Probability and Statistics. Wiley, 1995, ISBN: 9780471007104. dirección: https://books.google.es/books?id=z39jQgAACAAJ.

[10] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin y A. C. Courville, "Improved Training of Wasserstein GANs", en *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan y R. Garnett, eds., vol. 30, Curran Associates, Inc., 2017. dirección: https://proceedings.neurips.cc/paper_files/paper/2017/file/892c3b1c6dccd52936e27cbd0ff683d6-Paper.pdf.

[11] W. Rudin, *Real and Complex Analysis*, ép. Mathematics series. McGraw-Hill, 1987, ISBN: 9780071002769.

[12] *Customize what happens in Model.fit.* dirección: https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit.