Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Máster en Ciencias y Tecnologías de la Computación

Análisis del espacio latente en el auto-encoder variacional

Trabajo de Fin de Máster

Julio de 2022

César Pérez Curiel

Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Máster en Ciencias y Tecnologías de la Computación

Análisis del espacio latente en el auto-encoder variacional

Trabajo de Fin de Máster

Julio de 2022

Autor: César Pérez Curiel

Tutor: Luis Miguel Pozo Coronado

Resumen

El auto-encoder variacional ha adquirido mucha fama gracias a sus capacidades como modelo generador. El interés generado, junto con unos resultados espectaculares, principalmente en el ámbito de la generación de imágenes, a producido una enorme cantidad de artículos científicos relacionados con la generación de datos usando el auto-encoder variacional. Sin embargo, sus capacidades para extraer los factores que permiten esta generación no han sido tan estudiadas.

En este trabajo se proporciona una base teórica y práctica que sirve para estudios posteriores del espacio latente del auto-encoder variacional desde la perspectiva del *representation learning*. Se pretende, en este trabajo, estudiar las capacidades que tiene el VAE para extraer de los datos de entrenamiento los factores que sirven para agrupar los ejemplos proporcionados en función de las etiquetas que les han sido asignadas. En este estudio se parte de un auto-encoder estándar como línea base con la comparar.

Los resultados obtenidos indican que el auto-encoder variacional original no es capaz de agrupar los ejemplos mejor de lo que lo hace un auto-encoder estándar. Los resultados iluminan claramente cuál es el camino a seguir en el estudio del VAE desde el punto de vista de la obtención de representaciones de un conjunto de datos: El trabajo con el auto-encoder variacional para entrenar un encoder que sea capaz de obtener representaciones con las propiedades relevantes para resolver un problema concreto pasa, casi obligatoriamente, por el estudio y el desarrollo de nuevas variantes del VAE.

Palabras clave: *auto-encoder variacional*, *espacio latente*, *representation learning*

Abstract

The variational auto-encoder has become prominent because of its capabilities as a generative model. The interest generated, along with some spectacular results, mainly in the field of image generation, has produced an enormous amount of scientific articles related to data generation using the variational auto-encoder. However, its capabilities to extract the factors that allow data generation have not been studied as much.

In this work, theoretical and practical basis are provided for further studies on the latent space of the variational auto-encoder from the perspective of representation learning. This work pretends to study the capabilities of the VAE to extract from the training data the factors that allow to group the provided examples according to the labels that have been assigned to them. In this study, a standard auto-encoder is used as the baseline to compare with.

The results obtained indicate that the original variational auto-encoder is not able to group the examples any better than a standard auto-encoder. The results clearly illuminate which the next steps in the study of the VAE from the point of view of obtaining representations from a data set: working with the variational auto-encoder to train an encoder that is capable of obtaining representations with some relevant properties to solve a specific problem will probably require the study and development of new VAE variants.

Keywords: *variational auto-encoder*, *latent space*, *representation learning*

Índice general

Índice de tablas

Índice de figuras

Índice de listados

Prefacio

Hace ya mucho tiempo, antes de toda la propaganda que rodea hoy en día a la inteligencia artificial, que me fascinó la idea de un sistema que podía optimizar funciones complejas y que había que ajustar a un conjunto de datos. Las redes neuronales hoy ya no me producen tanta fascinación porque, entre otras cosas, sé que lo que hay tras ellas no es magia o brujería. Es, simple y llanamente, una función con un conjunto de parámetros que hay que optimizar. Descubrir los secretos del funcionamiento de las redes neuronales puede implicar que se pierda el interes en éllas. Será una cuestión de edad o de experiencia pero entender el funcionamiento no ha hecho que pierda el interés sino que ha provocado que me interese más aún cómo es el ajuste de las redes neurales y qué está pasando dentro de ellas. En esto consiste este trabajo, en el análisis del auto-encoder variacional desde el punto de vista de lo que está pasando dentro del modelo y no de los resultados que produce.

En la película *"Una mente maravillosa"*, John Nash (interpretado por el actor neozelandés Russell Crowe) es acompañado durante gran parte de su vida por sus propios demonios. Él sabe que están ahí pero, finalmente, es capaz de ignorarlos. Salvando las distancias, a mi me pasa algo similar; mi propio demonio, que es mi afán perfeccionista, me acompaña en todo lo que hago. Yo sé que está tras cada programa que descarto porque no está "bien hecho", tras cada párrafo que borro porque no es "correcto" e incluso, tras cada nota que no tiene un "buen sonido" en la trompa. Este trabajo es la prueba de que, como en la película, he conseguido ignorar mi afán de hacer un trabajo perfecto. Está aún por ver si lo he conseguido definitivamente o es un logro de este trabajo concreto. Posiblemente nunca lo sepa pero, tengo claras dos cosas. Una es que esto es un demonio (un *bug* que diría un programador para limitar la gravedad del problema) con el que tengo que convivir y, más importante aún, tengo claro cómo desactivar a este demonio: sigue el plan. Desde mi punto de vista, este trabajo está muy lejos de mi estándar de calidad pero lo que importa ahora mismo es que, según el plan, **el trabajo está hecho**.

Hubiese sido más sencillo y casi más adecuado ajustarme a alguna propuesta de trabajo de fin de máster. Habría tenido más claro lo que tenía que hacer y habría recorrido el camino de forma recta. El caso es que me he empeñado en no hacer eso e ir por libre. Y eso pasa factura. Me he sentido bastante sólo durante el proceso y, en muchísimas ocasiones, he estado muy perdido. Dicen que lo que no mata, engorda. Creo que en este caso, ya que sigo vivo, he engordado. En particular, en lo relativo a cómo organizar un proyecto de investigación, diría que mis conocimientos al respecto han engordado.

Madrid, julio de 2022

Capítulo 1

Introducción

En este capítulo se contextualiza el tema sobre el que versa este trabajo, el *auto-encoder variacional*, y se pone el foco en un aspecto concreto del mismo, el *espacio latente* y su capacidad para separar las representaciones de los ejemplos de entrada en función de la clase a la que pertencen cada uno de ellos. Además se propone un modesto conjunto de objetivos a alcanzar y se describe la metodología a usar para alcanzarlos.

1.1. Antecedentes del trabajo

El auto-encoder variacional que se analiza en este trabajo [16] es un modelo capaz de generar ejemplos similares a los proporcionados durante el entrenamiento a partir de una *representación* de menor dimensionalidad que dicha entrada. Estas capacidades para generar nuevos datos hay ayudado, en gran medida, a la popularidad que ha alcanzado el VAE y a la generación de artículos científicos tanto proponiendo nuevas y sorprendentes, variantes del modelo [29, 5] como estudiando propiedades tan interesantes como el *disentanglement*[1] [23, 21].

[1]El término *disentanglement* se refiere a "independizar" (no en el sentido estadístico) los factores que generan los datos. Por ejemplo, para generar una imagen de una persona se puede pensar en qué edad se quiere que aparente, en el sexo de la persona representada, etc. Como se verá más adelante en este trabajo, los factores del espacio latente que construye el auto-encoder variacional son difícilmente explicables, esto es, están "enredados"

1.1.1. La representación de los datos

Una representación es asimilable a un punto de vista sobre algún objeto. Por ejemplo, el concepto numérico de *tres* se puede representar mediante el número arábigo 3 o el número romano III. Existen más representaciones del mismo concepto como el ideograma 三 usado en el chino. Sin embargo, no todas las representaciones son adecuadas en todos los contextos.

Por ejemplo, la tarea de dividir 210 entre 6 [12] es bastante sencilla cuando se representan los números en notación arábiga. Si, por el contrario, se escoge la representación en números romanos, la tarea de dividir CCX entre VI, parece más complicada. A la hora de llevar a cabo esta última división, sería lógico transformar cada número a la notación arábiga y, posteriormente, expresar el resultado obtenido en notación romana. Este simple ejemplo deja patente la importancia de usar una representación adecuada de los datos para la resolución de un problema.

Al hilo de este ejemplo y dado un problema concreto, ¿existe un subconjunto de todas las posibles representaciones de los datos para dicho problema que sea más adecuado que el resto de subconjuntos? En ese caso parece pertinente preguntarse cómo puede construirse ese espacio y qué propiedades tiene que lo diferencien del resto. Dentro del aprendizaje automático, el área denominada como *representation learning*[3] se encarga del estudio de las representaciones y sus propiedades.

Como ejemplo claro de éxito se pueden citar las representaciones conocidas como *word embeddings*. Los modelos estadísticos y, en particular, las redes neuronales, requieren que la entrada de datos sean valores numéricos. Cuando se trabaja con el lenguaje natural los datos se presentan en formato de texto y, por ello, hay que buscar una forma de representarlos. Dada una frase concreta como, por ejemplo, *"El gato esperó un rato"* una posible representación puede ser transformar cada palabra en un vector cuyo tamaño viene determinado por un vocabulario previo y en el que todos los elementos son cero excepto el elemento del índice de cada palabra en el vocabulario. Esta representación, conocida como representación *one-hot*, cumple con el requisito de ser numérica pero parece algo ineficiente; si el vocabulario consta de 10000 palabras, cada uno de los vectores tiene 10000 elementos de los que únicamente la palabra que se quiere representar tiene valor 1.

Los modelos como WORD2VEC[24] han abordado la tarea de representar información textual desde el punto de vista de la construcción de un modelo que permita asignar un valor a cada palabra en función sus propiedades. En concreto, WORD2VEC se basa en el contexto[2] en que se usa una palabra.

[2]El contexto de una palabra, en este caso, se corresponde con las palabras que la

De esta forma, se puede asignar un vector numérico a la palabra *lápiz* que tendrá cierto parecido al vector de la palabra *bolígrafo* porque ambas suelen ir rodeadas de palabras similares.

Esta forma de construir las representaciones sugiere, por un lado, que deben de existir otros espacios de representaciones con otras propiedades diferentes[3] de forma que para un problema concreto no tenga por qué existir una representación válida sino una colección de representaciones que comparten una propiedad que las haga adecuadas para resolver el problema. Por ejemplo, dado un problema de clasificación, lo que podría ser interesante es representar los datos de forma que las representaciones de ejemplos pertenecientes a clases diferentes estén a mayor distancia que las representaciones de ejemplos pertenecientes a la misma clase.

Por otra parte, cabe preguntarse cómo se construyen los modelos que permiten obtener estas representaciones y qué propiedades tiene el espacio que construyen para representar los ejemplos. El *auto-encoder variacional* es un modelo usado para generar nuevos ejemplos a partir de representaciones pero, en este trabajo, se estudia este modelo desde el punto de vista de la representación de los datos y las propiedades que tiene (o no tiene) el espacio latente que construye.

1.1.2. El auto-encoder

Un auto-encoder es un modelo cuyo objetivo es obtener una reconstrucción de la entrada. Esto es, una red neuronal entrenada para aprender una función que obtenga una copia de los datos proporcionados.

La figura 1.1 muestra la arquitectura de un auto-encoder estándar. El modelo g_ϕ, denominado *encoder*, transforma la entrada x en una representación z que, generalmente, tiene mucha menor dimensionalidad que la entrada. A partir de dicha representación el modelo f_θ, denominado *decoder*, se encarga de reconstruir la entrada x' a partir de la representación z.

El verdadero potencial del auto-encoder no está, sin embargo, en ser capaz de reconstruir la entrada. En muchos contextos, como en imagen, los actuales algoritmos de compresión realizan un mejor trabajo que un auto-encoder. Entonces, ¿cuál es su utilidad? Tras el ajuste del VAE, se descarta el decoder y el encoder se usa para obtener la representación de los ejemplos de entrada.

---

acompañan en una frase. Por ejemplo, en la frase *"El gato esperó un rato"* el contexto de la palabra *esperó* con una ventana de tamaño 3 es *gato esperó un*.

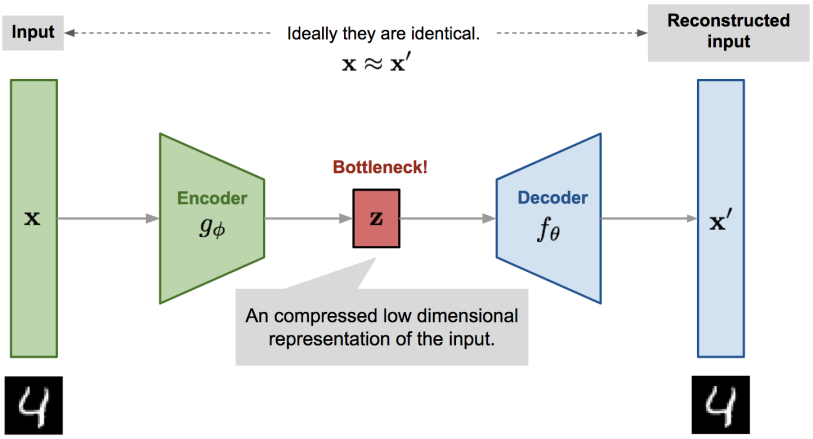


Figura 1.1: Arquitectura de un auto-encoder[30]

1.1.3. El auto-encoder variacional

El auto-encoder variacional[16] (o VAE) es un modelo que se basa en la misma idea que el auto-encoder estandar. Esto es, el el VAE se entrena con el objetivo de obtener una reconstrucción de la entrada. Sin embargo, el VAE es un modelo que permite generar nuevos datos a partir de una representación al contrario que el auto-encoder estándar.

La principal diferencia entre el auto-encoder variacional y su "hermano pequeño" está en las representaciones que generan. De alguna forma[3] el espacio latente que construye el VAE permite utilizar el decoder para generar ejemplos. Esta diferencia se traslada tanto al encoder, que debe tener una interpretación probabilística, como al propio algoritmo de entrenamiento que debe "dar una forma adecuada" al espacio latente.

Sabiendo que el auto-encoder variacional es capaz de obtener una representación a partir de la cual generar nuevos ejemplos, cabe preguntarse si este espacio latente tiene alguna otra propiedad. En particular, cuando se dispone de un conjunto de datos etiquetado según la clase a la que pertenece cada ejemplo, **¿recoge este hecho el espacio latente construido por un auto-encoder variacional?**.

[3]De forma intencionada no se desvelan los detalles del auto-encoder variacional hasta el capítulo 2

1.2. Objetivos del estudio

En este trabajo se ha definido, como objetivo principal:

> *Entender las propiedades del espacio latente que aprende el auto-encoder variacional* ***por medio de*** *una comparación de la representación aprendida por un auto-encoder estándar con la obtenida a partir del espacio latente del modelo objeto de estudio desde el punto de vista de la clasificación de ejemplos* ***para*** *facilitar la implementación de variantes que mejoren el rendimiento en tareas específicas.*

Dicho objetivo se concreta en los siguientes objetivos específicos:

1. *Construir una base teórica sobre el funcionamiento del auto-encoder variacional* ***mediante*** *el análisis de las publicaciones existentes que describan los conceptos del modelo original[16].*
2. *Demostrar los conceptos teóricos en que se basa el auto-encoder variacional* ***mediante*** *la implememementación de un modelo que ilustre de la manera más clara posible los principios del auto-encoder variacional original y que pueda ser usado como referencia en los experimentos.*
3. *Obtener evidencias de la capacidad del espacio latente aprendido por un auto-encoder variacional para la separación de las representaciones* ***mediante*** *la realización de un conjunto de experimentos que comparen el espacio latente aprendido con las representaciones que se obtienen con un auto-encoder estándar.*

1.3. Metodología de trabajo

El método de trabajo utilizado para alcanzar el objetivo enunciado en el epígrafe anterior consiste en llevar a cabo actividades que persigan cada uno de los objectivos especícos planteados en dicho epígrafe.

El primer paso en este trabajo es construir un **marco teórico** que sirva de base para la implementación del modelo y los experimentos a realizar. El estudio parte de una de las publicaciones en que se describe por primera vez el auto-encoder variacional[16]. A partir de ese punto se deben buscar referencias en que se traten los detalles supuestos o explicados de forma

confusa en la publicación original. El proceso concluye con una organización y presentación de los conceptos relevantes para el trabajo.

A partir del marco teórico construido se pretende hacer una **implementación** del auto-encoder variacional que respete todo lo posible las ideas básicas del modelo. Para llevar a cabo esta implementación es necesario disponer de los recursos *software* y *hardware* que permitan implementar y probar el modelo. Dado que estas pruebas no deben requirir grandes recursos de computación, con un entorno de desarrollo debería ser suficiente.

Para llevar a cabo la implementación se ha decidido usar el lenguaje de programación Python para codificar el modelo por su amplio uso entre la comunidad de aprendizaje automático para la implementación de modelos. Adicionalmente, dado su conocimiento previo, se ha decidido usar la biblioteca TensorFlow[1] para llevar a cabo el cómputo numérico requerido por los modelos.

Por útimo, se parte del modelo implementado para llevar a cabo un conjunto de **experimentos** en los que se pretende analizar el espacio latente aprendido por el auto-encoder variacional en comparación con las representaciones obtenidas mediante un auto-encoder estandar, haciendo un especial énfasis en las propiedades del espacio latente para separar las representaciones de los ejemplos de la entrada.

En este caso, los experimentos pueden necesitar un *hardware especializado* para el entrenamiento de los modelos. La idea inicial es replicar una implementación simple que sirva para mostrar el funcionamiento del modelo. Esta implementación debería poderse entrenar con soltura en cualquier ordenador moderno. Sin embargo, está previsto construir una implementación de mayor complejidad si los resultados de los experimentos así lo requiriesen. En este caso, el entrenamiento del modelo podría ser algo más costoso en tiempo de cómputo. Para ello se tiene acceso a una tarjeta gráfica *NVIDIA GTX 1070 Ti* con 8GB de memoría gráfica, que debe ser suficiente para la ejecución de los experimentos.

Los experimentos deben tener tres partes. En la primera se pretende demostrar las capacidades de reconstrucción de cada uno de los modelo. El resultado esperado, en este caso, es que ambos modelos sean capaces de obtener buenas reconstrucciones de los datos de entrada.

La segunda de las partes de los experimentos consiste en comparar los ejemplos generados por un auto-encoder estándar con los ejemplos generados por una auto-encoder variacional. De este experimento se espera que el auto-encoder variaciona genere ejemplos reconocibles mientras que los ejemplos generados por el auto-encoder estándar no sean reconocibles.

Para finalizar, la útima de las partes consta de los experimentos en que se compara el espacio latente aprendido por el auto-encoder variacional con el conjunto de representaciones obtenidas usando el auto-encoder estándar. En estos experimento se evaluan la separaciones de cada uno de los ejemplos con el resto de ejemplos con otras etiquetas diferentes y las distancia entre los ejemplos que tienen las mismas etiquetas.

Para simplificar el estudio se ha escogido trabajar con el conjunte de datos MNIST que consiste en una colección de imágenes que representan dígitos manuscritos. Cada una de las imágenes está etiquetada con un número del 0 al 9 que indica cuál es el número representado en la imagen. La principal ventaja de este conjunto de datos en este problema es que es lo suficiente complejo para poder implementar un modelo interesante pero no demasiado complejo como para que sea necesario implementar un modelo que exceda de los objetivos de este trabajo.

1.4. Organización de la memoria

Esta memoria se divide en cinco capítulos. El primero de ellos, este capítulo, contextualiza el trabajo y expone cuáles son los objetivos que se persiguen y cómo es la metodología a usar para alcanzarlos.

El capítulo 2 proporciona los conceptos teóricos que fundamentan el auto-encoder variacional. La finalidad de este capítulo es proporcionar una explicación asequible del modelo sin dejar de lado todo el formalismo matemático que le acompaña.

El capítulo 3 describe una implementación del auto-encoder variacional basada en los fundamentos del capítulo 2. El capítulo se centra en las decisiones de diseño que hay tras la implementación del modelo.

El capítulo 4 contiene la descripción de los experimentos realizados con el auto-encoder estándar y el auto-encoder variacional junto con los resultados obtenidos a partir de ellos. Entre dichos experimentos se incluye, además, la implementación de un VAE en el que tanto el encoder como el decoder se basan en el uso de convoluciones con el objetivo de mejorar las capacidades de representación del modelo original.

Por útimo, en el capítulo 5 se revisan los resultados de los experimentos para extraer conclusiones sobre el espacio latente del auto-encoder variacional. También se relatan las limitaciones del trabajo y, para finalizar, se detallan posibles líneas de trabajo que suponen una continuación del estudio propuesto.

Capítulo 2

El auto-encoder variacional

En el capítulo 1 se ha expuesto la idea que da sentido al auto-encoder variacional. Este capítulo, el modelo original[16] se analiza desde un punto de vista teórico para proporcionar una base sobre la que desarrollar el resto del estudio. Existen varias publicaciones en las que se proporciona información sobre el VAE objeto de estudio en este marco teórico[8, 31, 17, 26, 15, 9]. Los conceptos tratados aquí pretenden servir de resumen de los detalles incluidos en dichas publicaciones.

2.1. Conceptos previos

La pretensión de este capítulo es presentar el auto-encoder variacional evitando todo el formalismo matemático posible para facilitar la comprensión del modelo. Sin embargo es necesario establecer unos conceptos básicos que permitan entender la teoría expuesta en este capítulo.

2.1.1. Teorema de Bayes

Dados dos sucesos A y B, se tiene que la probabilidad de B condicionada por A es:

$$P(B|A) = \frac{P(B \cap A}{P(A)} \tag{2.1}$$

o lo que es lo mismo, la probabilidad de que ocurra B sabiendo que ha ocu-

rrido A es la probabilidad de que ambos sucesos ocurran simultáneamente sabiendo que ha ocurrido B. A partir de esta definición de probabilidad condicionada se puede derivar el teorema de Bayes[2] que se usa en la obtención de la función de coste del auto-encoder variacional. Partiendo de la ecuación 2.1 se tiene que:

$$P(B \cap A) = P(B|A)P(A)$$
$$P(A \cap B) = P(A|B)P(B)$$

y como la intersección es conmutativa, es decir $P(A \cap B) = P(B \cap A)$, se tiene que:

$$P(B|A)P(A) = P(A|B)P(B)$$

y operando se obtiene la expresión del teorema de Bayes:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{2.2}$$

En la ecuación 2.2 se suele denominar a las probabilidades condicionadas $P(B|A)$ y $P(A|B)$ como probabilidad *a posteriori* y *verosimilitud* respectivamente, a la probabilidad $P(B)$ como probabilidad *a priori* y, finalmente, a la probabilidad $P(A)$ se la denomina *evidencia*.

En el caso de trabajar con variables aleatorias continuas en inferencia variacional, la expresión más frecuente de este teorema es:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x|z)p(z)}{\int p(x|z)p(z)dz} \tag{2.3}$$

donde la evidencia $p(x)$ es la probabilidad marginal de los datos.

2.1.2. Divergencia de Kullback-Leibler

Cuando se quiere determinar la diferencia entre dos valores, por ejemplo, dos puntos en el espacio, una alternativa es calcular la distancia entre dichos puntos. En el caso de distribuciones de probabilidad, ¿cómo se puede obtener una medida de la diferencia entre ellas? La divergencia de Kulback-Leibler

[19] (también denominada divergencia KL o $D_{\mathbb{KL}}$) proporciona una medida de esta diferencia. En general, se puede calcular mediante:

$$D_{\mathbb{KL}}(P||Q) = \mathbb{E}_{x\sim P}\left[log\ \frac{P(x)}{Q(x)}\right] \tag{2.4}$$

Una de las propiedades de esta medida, importante en la obtención del límite inferior usado para el cálculo de la función de coste del auto-encoder variacional, es que $D_{\mathbb{KL}}(p||q) \geq 0$. Esto puede demostrarse usando la desigualdad de Jensen, que asegura que:

$$\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$$

donde $f(\cdot)$ es una función convexa. Esta desigualdad viene a decir que para funciones convexas, la media de los valores de la función es siempre mayor que los propios valores de la función. En el caso de funciones cóncavas, como el logaritmo, la desigualdad funciona a la inversa, de forma que:

$$\mathbb{E}[log\ x] \leq log\ \mathbb{E}[x]$$

2.2. La idea tras el auto-encoder variacional

Un auto-encoder variacional es, de forma intuitiva, un modelo similar a un auto-encoder estándar con una tipo de regularización "especial" sobre el espacio latente. Esta regularizacion hace que el decoder pueda ser utilizado para generar datos similares a la entrada a partir de muestras tomadas del espacio latente. La figura 2.1 ilustra la arquitectura del auto-encoder variacional.

Pero, ¿genera, el auto-encoder variacional, cualquier tipo de datos? La generación en el VAE está, realmente, "limitada" de forma que sólo es capaz de generar datos similares a los que se le han proporcionado durante el entrenamiento. Esto es, si se ha entrenado con sonidos no puede generar texto y si se le ha entrenado con imágenes de animales, no puede generar imágenes de plantas. Más aún, si las imágenes proporcionadas son imágenes en blanco y nego, el auto-encoder variacional original no puede generar imágenes parecidas en color. Esta limitación viene impuesta por el espacio latente que se construye durante el entrenamiento. Los datos se generan en función de cómo sea la muestra obtenida de dicho espacio. La figura 2.2

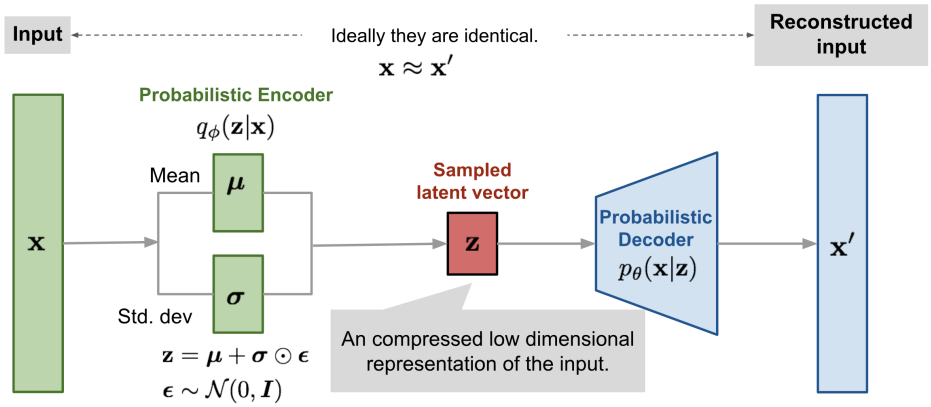


Figura 2.1: Arquitectura de un auto-encoder variacional[30]

ilustra esta dependencia entre el espacio latente y el espacio de los datos. La transformación de un valor en el espacio de los datos al espacio latente se lleva a cabo en el encoder y, de forma inversa, el decoder se encarga de obtener el dato a partir de una muestra del espacio latente.

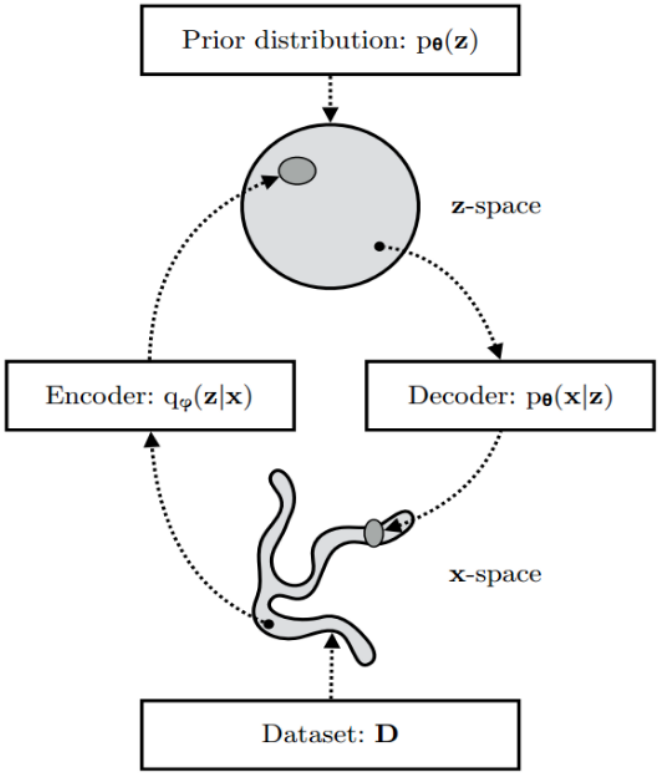


Figura 2.2: Relación entre el espacio de los datos y el espacio latente[17]

Sin embargo, esta transformación no es "perfecta"; si se obtiene una muestra a partir del espacio latente obtenido por el encoder para un dato concreto, la reconstrucción que se hace en el decoder no es exactamente igual al dato original. Por una parte el encoder produce un espacio latente que no es otra cosa que una distribución de probabilidad por lo que cualquier muestra obtenida a partir de él tiene, obligatoriamente, una componente de aleatoriedad.

Por otra parte, aunque el modelo se entrena para minimizar la diferencia entre el dato original y la reconstrucción del mismo, existe una restricción que evita que esta reconstrucción sea perfecta. Durante el entrenamiento, se obliga a la distribución del espacio latente a ser lo más parecida posible a una distribución a priori. Esto es, se obliga a que $q_\phi(z|x) \approx p_\theta(z)$. De forma simple, esto significa que se fuerza al espacio latente a mantener una "forma" que permita obtener muestras que conduzcan a datos generados que sean similares a los datos originales.

En el caso del auto-encoder variacional original[16] se supone un espacio latente que sigue una distribución de probabilidad multivariante de forma que cada una de las componentes de una muestra tomada de este expacio se corresponde con una característica o un parámetro del dato original o del dato que se pretende generar. Por ejemplo, si el auto-encoder se ha entrenado con dígitos manuscritos[20] cada una de estas características puede hacer referencia a algún aspecto concreto del dígito. Por ejemplo, una de las características podría refererirse a que el número esté más o menos inclinado. En realidad, el espacio latente en el auto-encoder variacional original no tiene una explicación sencilla.

En el epígrafe 2.3 se incluyen todos los detalles que se omiten en esta descipción del funcionamiento del VAE. Sin embargo, es importante destacar los obstaculos principales que resuelve el auto-encoder variacional a la hora de poder entrenar los modelos. En primer lugar, el encoder $p_\theta(z|x)$ depende de una integral que no se puede calcular. Por ello, se utiliza una aproximación $q_\phi(z|x)$ del encoder de forma que $q_\phi(z|x) \approx p_\theta(z|x)$. Esta aproximación (la forma de ajustarla) es lo que se conoce como *inferencia variacional*.

Por otra parte, está la dicultad de poder ajustar los modelos de forma eficiente y escalable para gran cantidad de datos. Normalmente, tanto el encoder como el decoder se implementan como redes neuronales. Estos dos modelos pueden ser entrenados sin mayor dificultad calculando el gradiente de la función de coste con respecto de los parámetros de los modelos. Sin embargo, la muestra del espacio latente, z, ilustrada en la figura 2.1 se obtiene mediante un proceso con un componente aleatorio que provoca que no sea posible calcular el gradiente en este punto. El *reparameterization trick*

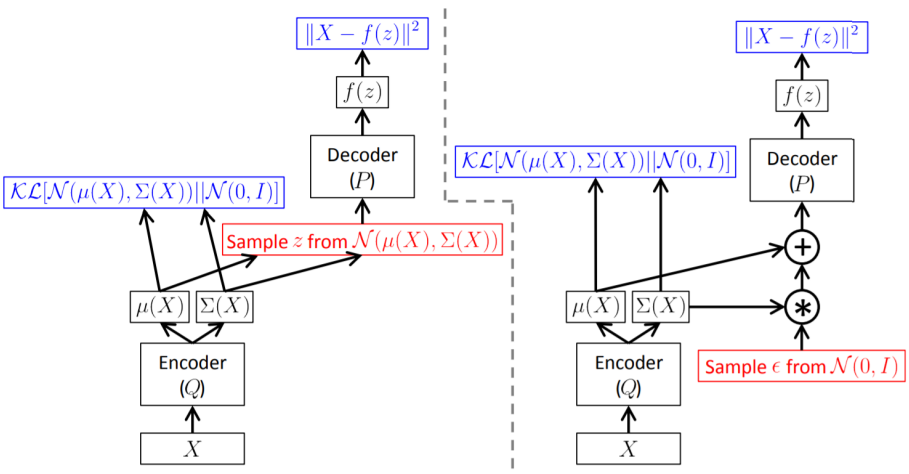


Figura 2.3: Detalle del *reparameterization trick*[9]

que se muestra en la figura 2.3 ilustra el cambio de variable que se realiza para dejar fuera el muestreo aleatorio del grafo de operaciones usado para el calculo del gradiente. En otras palabras, en lugar de tomar muestras de una distribución normal que depende de los parámetros que hay que ajustar, se expresa el cálculo de z de forma que el valor se calcule a partir de muestras de una $\mathcal{N}(0, 1)$ que no supone un problema en el cálculo de los gradientes por no depender de los parámetros del modelo.

Para poder construir el espacio latente usando el encoder y generar nuevos datos usando el decoder, es necesario ajustar los parámetros ϕ y θ de ambos modelos. Esto se hace optimizando una función de coste que busca, en resumen, minimizar el error que se comete en la reconstrucción de los datos al tiempo que mantiene el espacio latente similar a la distribución a priori. La función de coste tiene, por tanto un término que "penaliza" a diferencia entre los datos originales y sus reconstrucciones, y otro término que funciona como un regularizador del espacio latente [9, 25, 17].

En el caso, por ejemplo, de la generación de imágenes, el término que penaliza las reconstrucciones simplemente consiste en calcular la diferencia entre la imagen original y la reconstrucción. Si, por ejemplo, se consideran las imágenes como datos binarios, este término se implementa, comúnmente, como la entropía cruzada entre ambas imágenes. En el caso de valores normales se utilizara el error cuadrático medio para calcular esta penalización.

En el caso del término correspondiente a la regularización, la implementación se reduce, en el auto-encoder variacional original, a un cálculo de cómo de diferentes son la distribución del espacio latente y la distribución a

priori. Para calcular esta diferencia la opción suele ser emplear la divergencia de Kullback-Leibler.

2.3. El aprendizaje en el auto-encoder variacional

En el epígrafe 2.2 se ha descrito el auto-encoder variacional de una forma intuitiva con el objetivo de ilustrar de la forma más clara y más simple posible cuál es el funcionamiento del modelo. En esta sección se pretende ofrecer una descripción precisa de los fundamentos matemáticos existentes tras el aprendizaje del VAE.

La característica del auto-encoder variacional que lo diferencia de otros modelos generadores es el uso de un modelo probabilístico, $p_\theta(z|x)$ que se utiliza para aproximar la distribución del espacio latente a partir de la distribución de los datos de entrada. Este enfoque recibe el nombre de *amortized inference* [6]. Para calcular dicha distribución se puede usar el teorema de Bayes:

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)} \tag{2.5}$$

Sin embargo el denominador de la expresión 2.5 es intratable porque requiere calcular la probabilidad marginal:

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz \tag{2.6}$$

Para solventar esta dificultad en el auto-encoder variacional, en lugar de calcular la distribución a posteriori, $p_\theta(z|x)$, se busca una aproximación de forma que:

$$q_\phi(z|x) \approx p_\theta(z|x)$$

Gráficamente, el auto-encoder variacional se puede representar mediante la red bayesiana que se ilustra en la figura 2.4. En este grafo, los datos generados se obtienen a partir del espacio latente usando un decoder $p_\theta(x|z)$ y el espacio latente, que sigue una distribución a priori $p_\theta(z) \sim \mathcal{N}(\mu, \Sigma)$ se construye a partir de un encoder $q_\phi(z|x)$.

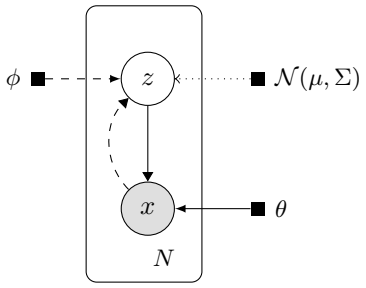


Figura 2.4: Red bayesiana del auto-encoder variacional

2.3.1. Obtención de la función objetivo

Dado un conjunto de datos $\mathcal{D} : x_1, ..., x_n$, el objetivo, a la hora de entrenar el auto-encoder variacional es maximizar la probabilidad de que los datos hayan sido generados por el modelo. Esto es:

$$\max_{\theta,\phi} \sum_{x \in \mathcal{D}} log\ p_\theta(x)$$

que, como se ha visto en la ecuación 2.6, es intratable. Para evitarlo, en lugar de maximizar directamente la probabilidad marginal de los datos se puede maximizar un límite inferior de la misma. Este límite, Ł se obtiene como:

$$\begin{aligned} log\ p_\theta(x) &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ p_\theta(x)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ \frac{p_\theta(x,z)}{p_\theta(z|x)}\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ \left(\frac{p_\theta(x,z)}{q_\phi(z|x)}\frac{q_\phi(z|x)}{p_\theta(z|x)}\right)\right] \\ &= \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ \frac{p_\theta(x,z)}{q_\phi(z|x)}\right]}_{Ł_{\theta,\phi}(x)} + \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]}_{D_{\mathbb{KL}}(q_\phi(z|x)||p_\theta(z|x))} \end{aligned} \tag{2.7}$$

A $Ł_{\theta,\phi}(x)$ se le denomina *variational lower bound*, *evidence lower bound*[4] o simplemente ELBO. Como se sabe que $D_{\mathbb{KL}} \geq 0$ entonces:

$$log\ P_\theta(x) \geq Ł_{\theta,\phi}(x) + D_{\mathbb{KL}}(q_\phi(z|x)||p_\theta(z|x)) \tag{2.8}$$

y, por ello, se puede asegurar que maximizando $Ł_{\theta,\phi}(x)$ también se maximiza $log\ p_\theta(x)$. Para obtener una expresión de este límite inferior que se pueda entender de forma más sencilla se opera de la siguiente manera:

$$\begin{aligned} Ł_{\theta,\phi}(x) &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ \frac{p_\theta(x,z)}{q_\phi(z|x)}\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ p_\theta(x,z) - log\ q_\phi(z|x)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ p_\theta(x|z) + log\ p_\theta(z) - log\ q_\phi(z|x)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ p_\theta(x|z)\right] - \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ q_\phi(z|x) - log\ p_\theta(z)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ p_\theta(x|z)\right] - \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ \frac{q_\phi(z|x)}{p_\theta(z)}\right]}_{D_{\mathbb{KL}}(q_\phi(z|x)||p_\theta(z))} \end{aligned} \tag{2.9}$$

La expresión del ELBO obtenida en la ecuación 2.9 puede entenderse como compuesta por un término, $\mathbb{E}_{z\sim q_\phi(z|x)}\left[log\ p_\theta(x|z)\right]$, que representa el error en la reconstrucción del dato de entrada y otro término, $D_{\mathbb{KL}}(q_\phi(z|x)||p_\theta(z))$, que representa la regularización que se aplica al espacio latente que aprende el auto-encoder variacional [9, 25, 17].

2.3.2. Optimización usando el gradiente

A la hora de ajustar los parámetros θ y ϕ de los modelos interesa usar algún método basado en el cálculo de gradientes como el *stochastic gradient descent* para que la optimización sea eficiente con gran cantidad de datos de entrenamiento[16]. Formalmente, dado un conjunto de datos, $\mathcal{D}$ formado por ejemplos independientes e idénticamente distribuidos, interesa maximizar:

$$Ł_{\theta,\phi}(\mathcal{D}) = \sum_{x\in\mathcal{D}} Ł_{\theta,\phi}(x)$$

y, para ello, se toma el gradiente:

$$\nabla_{\theta,\phi} Ł_{\theta,\phi}(x)$$

que resulta ser intratable[17] porque el gradiente con respecto de ϕ depende de un modelo parametrizado por ϕ tal y como se muestra en la ecuación 2.11.

Afortunadamente es posible calcular estimadores para dichos gradientes y, de ese modo, implementar el algoritmo de optimización.

En el caso del gradiente con respecto de θ, es sencillo calcular el estimador como:

$$\begin{aligned} \nabla_\theta Ł_{\theta,\phi}(x) &= \nabla_\theta \, \mathbb{E}_{z\sim q_\phi} \left[log \; p_\theta(x,z) - log \; q_\phi(z|x) \right] \\ &= \mathbb{E}_{z\sim q_\phi} \left[\nabla_\theta (log \; p_\theta(x,z) - log \; q_\phi(z|x)) \right] \\ &\simeq \nabla_\theta (log \; p_\theta(x,z)) \end{aligned} \tag{2.10}$$

que puede ser calculado con facilidad por las bibliotecas de cálculo numérico como TensorFlow o Pytorch.

Sin embargo, en el caso del gradiente con respecto de ϕ el cálculo no es tan sencillo ya que no se pueden intercambiar el gradiente y la esperanza por depender, esta última de la variable con respecto de la cual se calcula el gradiente. Esto es:

$$\begin{aligned} \nabla_\phi Ł_{\theta,\phi}(x) &= \nabla_\phi \, \mathbb{E}_{z\sim q_\phi} \left[log \; p_\theta(x,z) - log \; q_\phi(z|x) \right] \\ &\neq \mathbb{E}_{z\sim q_\phi} \left[\nabla_\phi (log \; p_\theta(x,z) - log \; q_\phi(z|x)) \right] \end{aligned} \tag{2.11}$$

En este caso, para poder calcular el gradiente es necesario implementar un cambio de variable —al que se denomina *reparameterization trick*[16]— en la forma que se tiene de obtener z como una muestra del espacio latente. En particular, se define la función $g(\cdot)$ como:

$$z = g(\phi, x, \epsilon) = \mu_\phi(x) + \sigma_\phi(x) \cdot \epsilon \tag{2.12}$$

donde $\mu_\phi(\cdot)$ y $\sigma_\phi(\cdot)$ se parametrizan usando el encoder $q_\phi(z|x)$ y ϵ se obtiene como una muestra de una distribución $\mathcal{N}(0, I)$. De esta forma, se tiene que:

$$\mathbb{E}_{z\sim q_\phi(z|x)} \left[f(z) \right] = \mathbb{E}_{\epsilon\sim p(\epsilon)} \left[f(z) \right]$$

Bajo este cambio de variable de variable es, finalmente, posible calcular

el gradiente con respecto de ϕ de la siguiente forma:

$$\begin{aligned} \nabla_\phi \, \mathbb{E}_{z\sim q_\phi(z|x)}[f(z)] &= \nabla_\phi \, \mathbb{E}_{\epsilon\sim p(\epsilon)}[f(z)] \\ &= \mathbb{E}_{\epsilon\sim p(\epsilon)} \, \nabla_\phi f(z) \\ &\simeq \nabla_\phi f(z) \end{aligned} \tag{2.13}$$

que, al igual que con el gradiente de $Ł_{\theta,\phi}(x)$ con respecto de θ se puede implementar con éxito usando las bibliotecas de cálculo numérico disponibles.

Capítulo 3

Implementación del modelo

En el capítulo 2 se han expuesto los fundamentos teóricos sobre los que se asienta el auto-encoder variacional. Partiendo de dicha base teórica, en este capítulo se propone una implementación de referencia que pretende ilustrar el modelo con la mayor claridad y precisión posibles. La tabla 3.1 recoge los paquetes software usados durante el desarrollo del modelo.

Paquete	**Versión**
Python	3.8
TensorFlow[1]	2.7.0

Tabla 3.1: Paquetes software usados para la implementación

3.1. El diseño del encoder

El listado 3.1 muestra una implementación del encoder usando como base la clase `Model` de la biblioteca Keras. En dicha implementación el modelo desarrollado recibe como entrada un lote de imágenes y devuelve los vectores que determinan la distribución a posteriori que representa a las entradas.

En el epígrafe 2.2 se describe el encoder como una función cuyo valor devuelto es una distribución de probabilidad. Esta implementación pretende reflejar de forma explícita dicha descripción devolviendo los valores que parametrizan la distribución a posteriori para cada ejemplo de la entrada. En particular, el vector `mean` de la línea 7 se corresponde con el vector de medias de la distribución del espacio latente y el vector `logvar` de la línea 8 cumple la función de la diagonal principal de la matriz de covarianzas.

```
class GaussianEncoder(Model):

    def __init__(self, hidden_units, latent_size, name=None):
        image = Input(shape=[28, 28, 1])
        features = Flatten()(image)
        hidden = Dense(units=hidden_units,
            activation=LeakyReLU())(features)
        mean = Dense(units=latent_size)(hidden)
        logvar = Dense(units=latent_size)(hidden)

        super(GaussianEncoder, self).__init__(
            inputs=image,
            outputs=[mean, logvar],
            name=name)
```

Listado 3.1: Implementación del *encoder*

3.2. El diseño del decoder

El listado 3.2 muestra una posible implementación del decoder que, al igual que el encoder, se basa en la clase `Model` de la biblioteca Keras. Para devolver la reconstrucción de la imagen original, al modelo se le proporciona una muestra obtenida del espacio latente generado por el encoder.

```
class Decoder(Model):

    def __init__(self, hidden_units, latent_size, name=None):
        sample = Input(shape=[latent_size])
        hidden = Dense(units=hidden_units,
            activation=LeakyReLU())(sample)
        features = Dense(units=784,
            activation='sigmoid')(hidden)
        image = Reshape(target_shape=[28, 28, 1])(features)

        super(Decoder, self).__init__(
            inputs=sample,
            outputs=image,
            name=name)
```

Listado 3.2: Implementación del *decoder*

En la arquitectura del decoder descrita en el epígrafe 2.2 se indica que la salida del modelo es la reconstrucción de la imagen original. La forma de implementar esta salida depende en gran medida del procesamiento previo que se haya hecho de los datos. Por ejemplo, si la entrada del modelo se procesase de forma que fuese una imagen binaria, la salida tendría que ser, también, una imagen binaria que podría modelarse mediante colección de distribuciones de Bernoulli. En este caso concreto, antes de procesar cada

una de las imágenes, los píxeles que las componen son valores enteros en el intervalo [0, 255]. Un procesamiento típico —que es el adotado en este caso—, consiste en convertir los valores de los píxeles en números reales en el intervalo [0, 1]. Este procesamiento de las imágenes obliga a utilizar la función sigmoide para obtener la reconstrucción como se muestra en la línea 6 del listado 3.2.

3.3. El algoritmo de entrenamiento

El auto-encoder variacional no es más que el algoritmo mediante el cual se ajustan un encoder y un decoder a un conjunto de datos. Esto implica que, una vez se ha terminado el entrenamiento, sólo tienen interés el encoder o el decoder. De esta forma, la implementación del VAE no es más que un bucle que se ejecuta un número concreto de iteraciones[1]. La propuesta en este trabajo es usar la clase `Model` proporcionada por la biblioteca KERAS que permite personalizar qué ocurre en la inferencia del VAE durante el entrenamiento.

El diseño del encoder descrito en el epígrafe 3.1 obliga a incluir el muestreo del espacio latente como parte del algoritmo de entrenamiento ya que no se está considerando como parte del encoder. La implementación incluida en el listado 3.3 muestra el código del *reparameterization trick* cuyo funcionamiento se ilustra en la figura 2.3.

```
1 class Sampling(Layer):
2
3     def call(self, inputs):
4         mean, logvar = inputs
5         shape = tf.shape(mean)
6         epsilon = tf.random.normal(shape=shape)
7         sample = mean + tf.exp(0.5 * logvar) * epsilon
8         return sample
```

Listado 3.3: Implementación del *reparameterization trick*

En muchas implementaciones del auto-encoder variacional se considera el muestreo como parte del modelo del encoder. La misión que éste tiene es representar cada ejemplo en un espacio probabilistico, es decir, el encoder no devuelve una representación concreta sino una distribución de probabilidad. Esta distribución está perfectamente determinada por un vector de medias y

[1]Durante la revisión de este trabajo surgió el dilema de traducir *epoch* por *época*. Es muy frecuente usar esta traducción en español. Sin embargo, tras pensarlo con calma, parece mejor usar *iteración* a falta de un término más adecuado

por la diagonal de la matriz de covarianzas devueltas por el encoder. Parece, de esta forma, más fiel a la descripción del auto-encoder variacional que la muestra de la distribución a posteriori sea responsabilidad de la implementación del algoritmo de entrenamiento del VAE y no parte del encoder.

El listado 3.4 muestra la codificación completa del modelo β-VAE[13] usado para implementar el algoritmo para el entrenamiento del encoder y del decoder. La característica particular de esta implementación es el uso del parámetro β que, cuando toma el valor 1, es equivalente a un auto-encoder variacional estándar.

```
1  class BetaVAE(Model):
2
3      def __init__(self, encoder, decoder, beta=1, name=None):
4          super(BetaVAE, self).__init__(name)
5          self._encoder = encoder
6          self._decoder = decoder
7          self._beta = beta
8          self._sampler = Sampling()
9
10     def call(self, image, training=False):
11         mean, logvar = self._encoder(image)
12         sample = self._sampler([mean, logvar])
13         reconst = self._decoder(sample)
14
15         reconst_loss = self._reconst_loss(image, reconst)
16         self.track_loss(reconst_loss, name='reconst_loss')
17
18         diverg_loss = self._diverg_loss(mean, logvar)
19         diverg_loss *= self._beta
20         self.track_loss(diverg_loss, name='diverg_loss')
21
22         return reconst
23
24     def track_loss(self, loss, name):
25         self.add_loss(loss)
26         self.add_metric(loss, name=name)
27
28     def _reconst_loss(self, original, reconstructed):
29         loss = binary_crossentropy(original, reconstructed)
30         loss = tf.reduce_sum(loss, axis=[1, 2])
31         loss = tf.reduce_mean(loss)
32         return loss
33
34     def _diverg_loss(self, mean, logvar):
35         loss = -0.5 * (1 + logvar - tf.square(mean) -
               tf.exp(logvar))
36         loss = tf.reduce_sum(loss, axis=1)
37         loss = tf.reduce_mean(loss)
38         return loss
```

Listado 3.4: Implementación del β-VAE

Es interesante destacar del diseño de la función de coste en la implementación de la figura 3.4. La biblioteca KERAS se ajusta muy bien a la implementación de modelos cuya función de coste no depende de parámetros internos del modelo. Por ejemplo, en un modelo de clasificación, la función de coste típica depende de los resultados obtenidos por el modelo para cada uno de los ejemplos de entrada y de las clases asignados a cada uno de ellos. La biblioteca se encarga de llamar a la función de coste indicada durante la compilación del modelo usando los resultados del obtenidos en cada iteración junto con las correspondientes etiquetas.

Cuando la función es una expresión cuyo valor depende de parámetros internos del modelo, no es posible especificar una función externa que cumpla este cometido sin depender de variables globales. Muchas de las implementaciones tomadas como referencia para construir el modelo del listado 3.4 dependen de este tipo de variables globales. El modelo propuesto pretende ser fiel en la medida de lo posible con el modelo teórico presentado en el capítulo 2. Sin embargo, esto no implica construir software que incurra en malas prácticas de desarrollo superadas hace muchos años como el uso de variables globales.

El modelo propuesto incluye en el proceso de inferencia (reflejado en el método `call`) el cálculo de la función de coste. Dicha función está dividida en un término que calcula el error de reconstrucción y en otro término que calcula la diferencia entre la distribución a posteriori y la distribución a priori. Estos dos términos son calculados por los métodos `_recons_loss` y `_diverg_loss` respectivamente.

Capítulo 4

Experimentos

En el capítulo 3 se ha propuesto una posible implementación del auto-encoder variacional como referencia de los conceptos teóricos expuestos en el capítulo 2. En este capítulo se usa dicha implementación para llevar a cabo una comparación del espacio latente del auto-encoder variacional con la representación de la entrada obtenida por un auto-encoder estándar.

4.1. Configuración de los experimentos

En la tabla 4.1 se incluyen los recursos *hardware* de los que se ha dispuesto para la ejecucion de los experimentos. Hay que destacar que para poder usar la aceleradora gráfica desde la biblioteca TensorFlow se ha usado la versión 11.6 del paquete CUDA (la útima disponible en el momento de la ejecución de los experimentos).

Recurso	**Descripción**
Procesador	Intel Core i5 @ 3.2 Ghz
Memoria RAM	32Gb
Aceleradora gráfica	Nvidia GTX 1070Ti

Tabla 4.1: Recursos *hardware* disponibles para los experimentos

4.1.1. Implementación de la línea base

Para establecer una base con la que comparar el auto-encoder variacional se ha implementado un auto-encoder estándar. La principal diferencia entre

ambos modelos es la función de coste. En el auto-encoder estándar, el error cometido por el modelo en la tarea de reconstruir el ejemplo de entrada se reduce a calcular la diferencia entre cada par de píxeles que tienen la misma posición en la entrada y en la salida. En el auto-encoder variacional la función de coste tiene un término adicional que refleja la diferencia entre la distribución del espacio latente y la distribución a priori. El código completo del auto-encoder estándar donde se refleja esta diferencia en la función de coste utilizada en cada modelo se incluye en la figura 4.1.

```
1  class VanillaAE(Model):
2
3      def __init__(self, encoder, decoder, name=None):
4          super(VanillaAE, self).__init__(name)
5          self._encoder = encoder
6          self._decoder = decoder
7
8      def call(self, image, training=False):
9          encoding = self._encoder(image)
10         reconst = self._decoder(encoding)
11
12         reconst_loss = self._reconst_loss(image, reconst)
13         self.add_loss(reconst_loss)
14
15         return reconst
16
17     def _reconst_loss(self, original, reconstructed):
18         loss = binary_crossentropy(original, reconstructed)
19         loss = tf.reduce_sum(loss, axis=[1, 2])
20         loss = tf.reduce_mean(loss)
21         return loss
```

Listado 4.1: Implementación del *auto-encoder estándar*

Además de esta diferencia en la función de coste, la arquitectura del encoder también difiere en ambos modelos. Mientras que en el auto-encoder estándar el valor resultante es la representación obtenida de la entrada, como se detalla en el listado 4.2, en el auto-encoder variacional la salida no es la representación del ejemplo de entrada sino los parámetros de la distribución del espacio latente.

Es importante destacar que, para establecer una comparación lo más equilibrada posible, el decoder de ambos modelos tiene la misma arquitectura. De esta forma, la comparación se centra en el espacio latente de ambos modelos.

```
1  class SimpleEncoder(Model):
2
3      def __init__(self, hidden_units, latent_size, name=None):
4          image = Input(shape=[28, 28, 1])
5          features = Flatten()(image)
6          hidden = Dense(units=hidden_units,
               activation=LeakyReLU())(features)
7          encoding = Dense(units=latent_size)(hidden)
8
9          super(SimpleEncoder, self).__init__(
10             inputs=image,
11             outputs=encoding,
12             name=name)
```

Listado 4.2: Implementación de un *encoder* sencillo

4.1.2. Descripción del conjunto de datos

El conjunto de datos *MNIST*[20] es una colección de imágenes de 28x28 píxeles en escala de grises que representan dígitos manuscritos del 0 al 9. Cada una de las imágenes está etiquetada con el número en élla representado. La principal razón para usar este conjunto de datos es, por una parte, que es suficientemente simple para los objetivos de este trabajo y, por otra parte, que parece haberse convertido en un conjunto de datos estándar utilizado para evaluar los modelos del tipo del estudiado en este trabajo. Esto último permite comparar los resultados que se han obtenido con los resultados proporcionados por otros estudios.

En estos experimentos, se ha tratado de forma diferente el conjunto de datos para el entrenamiento y el conjunto de datos para la evaluación del modelo entrenado. Para los datos de entrenamiento se ha diseñado un *pipeline* usando el interfaz `Dataset` proporcionado por la biblioteca TensorFlow. Para obtener los datos que se usan durante el entrenamiento de los modelos, se han convertido los valores enteros a valores reales y se han normalizado para que dichos valores estén en el intervalo [0, 1].

Con los datos usados para la evaluación de los modelos se ha usado el mismo procesamiento. La diferencia entre ambos métodos está en la obtención de los datos; mientras que para el conjunto de datos de entrenamiento se ha usado la biblioteca TensorFlow Datasets en el proceso de carga, en el caso de la evaluación los datos proceden directamente de la biblioteca Keras.

4.2. Reconstrucción de imágenes

En este experimento se usan, directamente, los modelos entrenados para obtener las representaciones de algunos de los primeros ejemplos del conjunto de imágenes usado para la evaluación. Lo que se pretende es validar la implementación realizada tanto del modelo base, el auto-encoder estándar, como del modelo objeto de este trabajo, el auto-encoder variacional. Además este experimento debe servir como punto de partida de la comparación entre ambos modelos ya que, de forma intencionada, se ha buscado construir implementaciones lo más parecidas posible para establecer una comparación justa entre ambos modelos.

El resultado esperado es, por lo tanto, reconstrucciones de los ejemplos de prueba parecidos a los ejemplos originales. En esta implementación, el auto-encoder variacional dispone de un mayor número de parámetros entrenables que el auto-encoder estándar. Por ello, un resultado esperable es que las reconstrucciones obtenidas mediante el VAE sean más fieles a los ejemplos originales que las reconstrucciones obtenidas usando la líneea base.

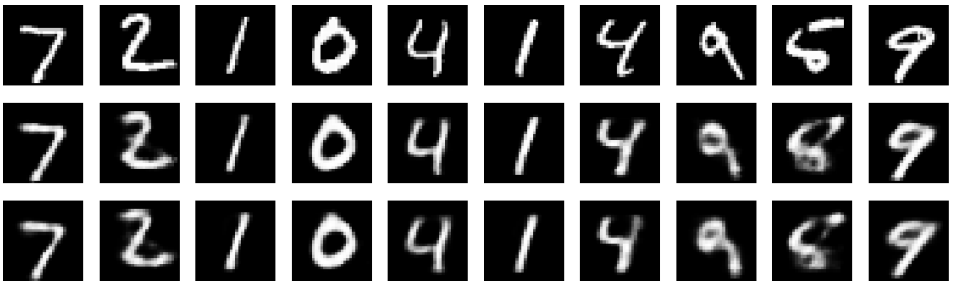


Figura 4.1: Reconstrucción de imágenes de muestra

La figura 4.1 muestra las imágenes originales junto con los resultados de la reconstrucción de las mismas usando ambos modelos. Las imágenes de la fila superior corresponden a las imágenes originales proporcionadas a los modelos. En la segunda y tercera filas se muestran las reconstrucciones obtenidas mediante el auto-encoder estándar y el auto-encoder variacional respectivamente.

A la vista de las reconstrucciones se puede concluir que las implementaciones de cada uno de los modelos parecen ser correctas. A parte de obtener reconstrucciones aparentemente buenas, sorprende ver que las reconstrucciones obtenidas son muy parecidas. Es tentador pensar que los espacios latentes aprendidos por ambos modelos deben de ser parecidos ya que el resultado de la reconstrucción es muy parecido usando el mismo decoder en ambos modelos. Sin embargo, se sabe que dicho espacio latente debe de

tener propiedades diferentes para cada modelo teniendo en cuenta conceptos expuestos en el capítulo 2.

4.3. Generación de imágenes

En esta sección se describen los experimentos enfocados a la comparación de las capacidades de generación de imágenes del auto-encoder estándar frente a las capacidades del auto-encoder variacional desde el punto de vista del espacio latente aprendido por éste último.

4.3.1. Generación a partir de los centroides de cada clase

El auto-encoder estándar no es un modelo enfocado a la generación de datos. Sin embargo, esto no implica que no pueda ser usado para generar nuevos ejemplo. El punto de partida consiste en asumir que la generación en un auto-encoder estándar no funciona porque las representaciones en el espacio latente están muy dispersas. Esto es, muchas de las representaciones no tienen una correspondencia con un ejemplo real. La hipótesis que se pretende demostrar es que representaciones similares a las de ejemplos reales también llevan a la generación de ejemplos reconocibles como reales.

Para esto se han calculado los valores medios de las representaciones e cada clase y se han utilizado para generar ejemplos. Se pretender confirmar, de esta forma, que los ejemplos de una misma clase (y, por ello, similares desde el punto de vista de una persona) llevan a generar ejemplos que, desde el punto de vista humano, podrían ser valores pertenecientes a dicha clase.

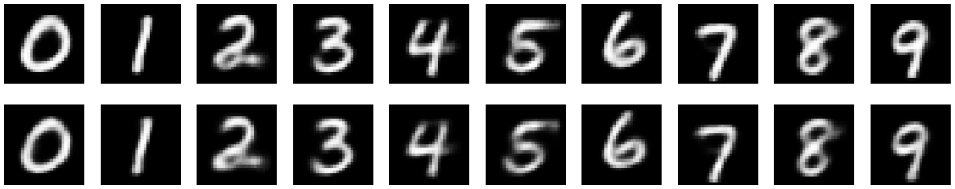


Figura 4.2: Generación usando la representación media de cada clase

En la figura 4.2 se muestran las imágenes obtenidas usando el decoder entrenado con un auto-encoder estándar (fila superior) y entrenado usando el auto-encoder variacional (fila inferior) cuando se les proporciona como entrada los vectores correspondientes a la representación media de cada una de las clases del conjunto de datos de evaluación.

En parte resulta algo llamativo que ambos modelos generen imágenes tan parecidas teniendo en cuenta que los espacios de las representaciones de cada uno es diferente. Sin embargo, se podría decir que las representaciones medias de cada una de las clases "incluyen" una pequeña parte de cada uno de los ejemplos de la clase a la que se refieren. Esto lleva a pensar que ambos modelos son capaces de extraer información sobre la estructura de las imágenes de entrada y que, por lo tanto, podría ser que los espacios que construyen estos modelos sirviesen para agrupar las representaciones en función de la clase a la que pertenecen.

4.3.2. Generación a partir de la distribución a priori

En este caso, el experimento consiste en demostrar las capacidades de generación de ejemplos del auto-encoder variacional frente a las capacidades de un auto-encoder estándar. Se busca confirmar que, para un mismo valor de la representación usada como entrada de la generación de datos, el auto-encoder variacional es capaz de obtener una representación que tiene sentido para la generación de ejemplos, mientras que el auto-encoder estándar es incapaz de obtenerla. Es importante señalar que, en ambos casos, la arquitectura del decoder es la misma y que, por lo tanto, la diferencia sólo puede estar en el espacio latente aprendido.

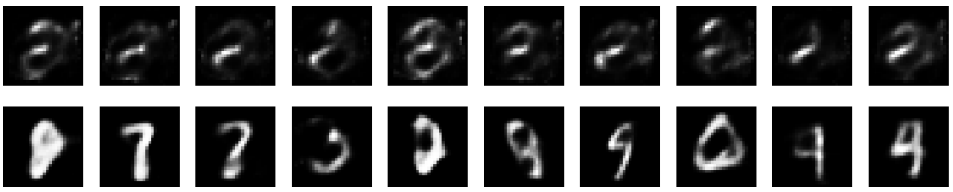


Figura 4.3: Generación de ejemplos a partir de la distribución *a priori*

En la figura 4.3 se muestran las imágenes obtenidas a partir de muestras tomadas de la distribución a priori usada para entrenar el auto-encoder variacional. Dado que en el auto-encoder variacional original la distribución a priori es una normal multivariante, las muestras se pueden tomar simplemente generando números aleatorios de dicha distribución. En la figura, la fila superior corresponde a las imágenes generadas usando el auto-encoder estándar y las imágenes de la fila inferior corresponden a las generadas usando el auto-encoder variacional.

Los resultados de la figura 4.3 evidencian que auto-encoder variacional permite generar ejemplos a partir del espacio definido por la distribución a priori. Si bien los resultados obtenidos tienen una calidad decepcionante, lo que se pone de manifiesto es la diferencia entre los espacios latentes obteni-

dos con ambos modelos a pesar de ser éstos muy parecidos en cuanto a la arquitectura. Una inquietud que provoca este resultado es cómo mejorar el auto-encoder variacional para producir resultados de mejor calidad.

4.3.3. Mejoras en la generación de imágenes

Este experimento surge como extensión del experimento de generación de ejemplos a partir de muestras de la distribución a priori descrito en el epígrafe 4.3.2. Lo que se busca es determinar si es posible mejorar las capacidades del auto-encoder variacional para la generación de ejemplos y, así, mejorar también las capacidad del encoder para representar los ejemplos de entrada.

Para ello, es necesario desarrollar un nuevo modelo de auto-encoder variacional en que los modelos se implementen sacando partido del tipo de datos que se están manejando. En el caso del conjunto de datos MNIST se han utilizado convolucionales para poder extraer características más relevantes de las imágenes del conjunto de datos.

El listado 4.3 muestra la implementación del encoder convolucional usado en este experimento. Con respecto a la implementación del encoder ilustrado en listado 3.1, el código aquí mostrado sustituye la extracción de características por una secuencia de convoluciones.

De la misma forma, en el listado 4.4 se sustituye la reconstrucción de la imagen mostrada en el listado 3.2 por una secuencia de operaciones de convolución que sirven para transformar en una imagen la muestra obtenida del espacio latente.

El auto-encoder variacional usado para entrenar estos dos nuevos modelos es, sin embargo el mismo que el utilizado para la implementación del capítulo 3. De esta forma se pretende comparar de una manera más justa la implementación original del auto-encoder variacional con esta nueva nueva implementación.

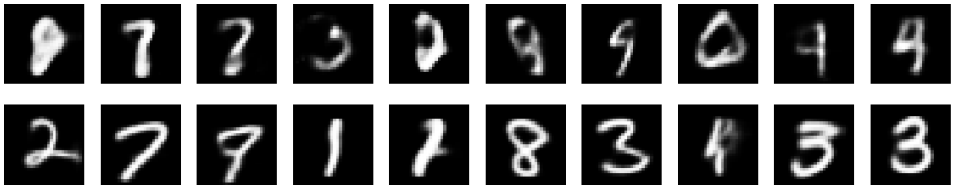


Figura 4.4: Mejoras en la generación de ejemplos en el *VAE* y el *VAE convolucional*

```
class ConvolutionalEncoder(Model):

    def __init__(self, hidden_units, latent_size, name=None):
        image = Input(shape=[28, 28, 1])
        maps = Conv2D(
            filters=32, kernel_size=5, strides=1,
            padding='same', activation=LeakyReLU())(image)
        maps = Conv2D(
            filters=32, kernel_size=5, strides=2,
            padding='same', activation=LeakyReLU())(maps)
        maps = Conv2D(
            filters=64, kernel_size=5, strides=1,
            padding='same', activation=LeakyReLU())(maps)
        maps = Conv2D(
            filters=64, kernel_size=5, strides=2,
            padding='same', activation=LeakyReLU())(maps)
        maps = Conv2D(
            filters=128, kernel_size=7, strides=1,
            padding='valid', activation=LeakyReLU())(maps)
        features = Flatten()(maps)

        mean = Dense(units=latent_size)(features)
        logvar = Dense(units=latent_size)(features)

        super(ConvolutionalEncoder, self).__init__(
            inputs=image,
            outputs=[mean, logvar],
            name=name)
```

Listado 4.3: Implementación de un *encoder* convolucional

La figura 4.4 muestra, en la fila superior, las imágenes generadas usando el VAE y, en la fila inferior, las imágenes generadas usando el VAE convolucional. Se debe tener en cuenta que, en este caso, el decoder usado ya no tiene la misma arquitectura en ambos modelos. Se puede apreciar que las imágenes generadas con el VAE mejorado son algo más parecidas a las imágenes originales que las generadas con el VAE original.

Los resultados de la figura 4.4 muestran una mejora significativa en la generación de nuevos ejemplos cuando se usa un encoder con mayor capacidad para representar los datos y un decoder más potente para la generación de imágenes a partir del espacio latente. Aún así es evidente que hay un importante margen de mejora de las capacidades del auto-encoder variacional.

```
class ConvolutionalDecoder(Model):

    def __init__(self, hidden_units, latent_size, name=None):
        sample = Input(shape=[latent_size])
        maps = Reshape(target_shape=[1, 1,
            latent_size])(sample)
        maps = Conv2DTranspose(
            filters=64, kernel_size=7, strides=1,
            padding='valid', activation=LeakyReLU())(maps)
        maps = Conv2DTranspose(
            filters=64, kernel_size=5, strides=1,
            padding='same', activation=LeakyReLU())(maps)
        maps = Conv2DTranspose(
            filters=64, kernel_size=5, strides=2,
            padding='same', activation=LeakyReLU())(maps)
        maps = Conv2DTranspose(
            filters=32, kernel_size=5, strides=1,
            padding='same', activation=LeakyReLU())(maps)
        maps = Conv2DTranspose(
            filters=32, kernel_size=5, strides=2,
            padding='same', activation=LeakyReLU())(maps)
        maps = Conv2DTranspose(
            filters=32, kernel_size=5, strides=1,
            padding='same', activation=LeakyReLU())(maps)

        image = Conv2D(
            filters=1, kernel_size=5, strides=1,
            padding='same', activation='sigmoid')(maps)

        super(ConvolutionalDecoder, self).__init__(
            inputs=sample,
            outputs=image,
            name=name)
```

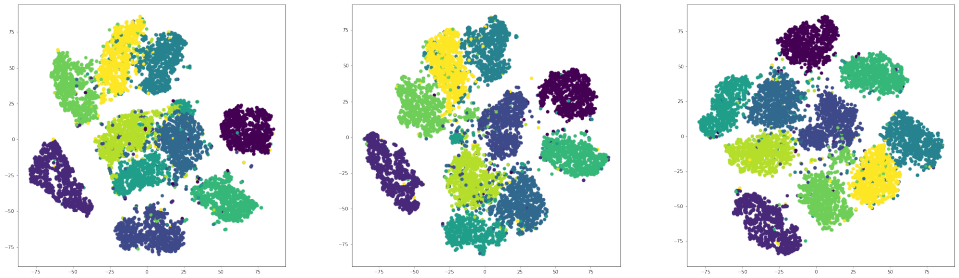
Listado 4.4: Implementación de un *decoder* convolucional

4.4. Separación de las representaciones

Una de las tareas tradicionales en aprendizaje no supervisado es el *clustering*. De forma simple, esto es agrupar los datos de entrada en función de alguna característica de los mismos. Un buen algoritmo debe separar cada grupo claramente a la vez que mantener lo más cerca posible a los ejemplos pertenecientes a un mismo grupo. Con estos experimentos se busca comparar las capacidades del *auto-encoder estándar* y del *auto-encoder variacional* para separar los ejemplos de entrada en función de la clase a la que pertenecen cada uno de ellos.

4.4.1. Visualización del espacio latente

Este experimento tiene como objetivo obtener comparar de forma visual el espacio latente de cada uno de los modelos desde el punto de vista de la separación por clases de las representaciones obtenidas a partir del conjunto de datos de evaluación. Dada la mayor capacidad de representación del auto-encoder variacional y, sobre todo del VAE convolucional usado en el epígrafe 4.3.3, es de esperar que estos dos últimos modelos sean capaces de obtener un espacio latente en el que las representaciones estén separadas de forma más clara.



(a) Clusters en el *auto-encoder*

(b) Clusters en el *auto-encoder variacional*

(c) Clusters en el *VAE convolucional*

Figura 4.5: Visualización de los clústers usando el algoritmo T-SNE

La figura 4.5 muestra una visualización del espacio latente usando el algoritmo T-SNE[14]. Este algoritmo es muy usado para obtener visualizaciones en dos o tres dimensiones de espacios de una dimensionalidad mucho mayor. En la figura 4.5a se muestra el espacio latente obtenido usando el auto-encoder estándar. De la misma forma, las figuras 4.5b y 4.5c muestran, respectivamente, el espacio latente obtenido usando el auto-encoder variacional y el VAE convolucional.

Sorprendentemente auto-encoder variacional usado en este experimento no sólo parece construir un espacio latente que no separa mejor las clases sino que también parece algo más disperso. Para verificar esta siguación es necesario calcular una medida de esta dispersión para obtener una conclusión definitiva sobre si el auto-encoder variacional mejora las capacidades del auto-encoder estándar para separar las representaciones en el espacio latente en función de la clase a la que pertenecen los ejemplos representados.

También es destacable la diferencia que existe entre los resultados obtenidos en el experimento de mejora del auto-encoder variacional descrito en el epígrafe 4.3.3 y los resultados que se arrojan en este experimento. Mientras que la mejora del VAE implicaba una mejora en las capacidades de generación del decoder, desde el punto de vista de la separación de las

representaciones no hay una mejora aparente. Esto induce a pensar que, posiblemente, el espacio latente del auto-encoder variacional tenga propiedades más interesantes que la de separar las representaciones.

4.4.2. Medida de la separación de las representaciones

A partir de la visualización de las representaciones de los ejemplos por clases del experimento descrito en el epígrafe 4.4.1 surge la duda sobe la separacion real de cada una de las clases. En este experimento se trata de medir dicha separación tanto para todas las clases en conjunto como por cada una de las clases con el objetivo de confimar la hipótesis de que las capacidades de *clustering* de ambos algoritmos son similares.

A la vista de la figura 4.5 no se esperan diferencias significativas en cuando a las distancias entre los distintos grupos. Sin embargo, sí parece que el auto-encoder variacional representa los ejemplos de algunas clases de forma más dispersa. Esto induce a pensar que los resultados de los modelos variacionales serán algo peores que los del auto-encoder estándar.

El algoritmo SILHOUETTE[28] usado en este experimento para obtener una medida del agrupamiento de cada modelo, combina en un único valor la cohesión media de cada uno de los grupos con la mínima distancia entre cada par de grupos. Formalmente:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}$$

donde $a(i)$ es la distancia media del elemento i-ésimo al resto de elementos pertencientes a su misma clase y $b(i)$ es la mínima distancia de entre las distancias medias del elemento i-ésimo al resto de elementos de clases diferentes. El valor de a puede ser entendido como una medida de cómo de "apiñados" están los elementos de la clase de un elemento. Por otra parte, b se puede ver como lo "lejos" que está el grupo distinto más cercano a un elemento.

Es importante señalar hacer dos consideraciones. En primer lugar, $s(i)$ se corresponde con el valor del algoritmo SILHOUETTE correspondiente a un único elementor. En este experimento se han considerado valores medios para hacer la comparación entre los modelos porque proporcionar un mayor detalle no es relevante para este experimento.

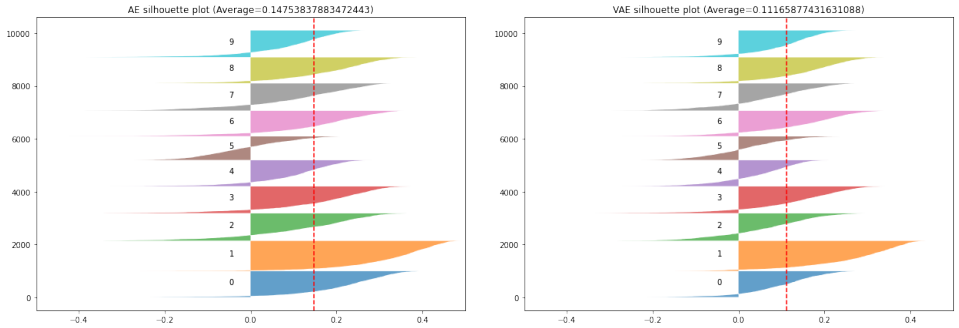
En segundo lugar, también es importante, destacar que los valores obtenidos mediante el algoritmo SILHOUETTE pertenecen al intervalo [-1, 1].

Esto puede verse como que un valor 1 indica que un elemento está muy cerca de los elementos pertenecientes a su misma clase y muy lejos de los elementos de clases diferentes. Un valor de -1 se puede ver, por su parte, que el elemento está más cerca de elementos de clases diferentes que de los elementos de su propia clase.

Modelo	**Silhouette**
AE	0,1475
VAE	0,1117
CVAE	0,1182

Tabla 4.2: Valores medios de SILHOUETTE por modelo

La tabla 4.2 recoge los valores medios obtenidos usando el algoritmo SILHOUETTE para cada uno de los tres modelos incluidos en este experimento. Es importante recordar que este valor se encuentra en el intervalo [-1, 1]. donde el valor -1 corresponde ejemplos que están lo más lejos posible de resto de ejemplos pertenecientes a su clase y el valor 1 corresponde a ejemplos que están lo más cerca posible del resto de ejemplos que pertenecen a su clase.

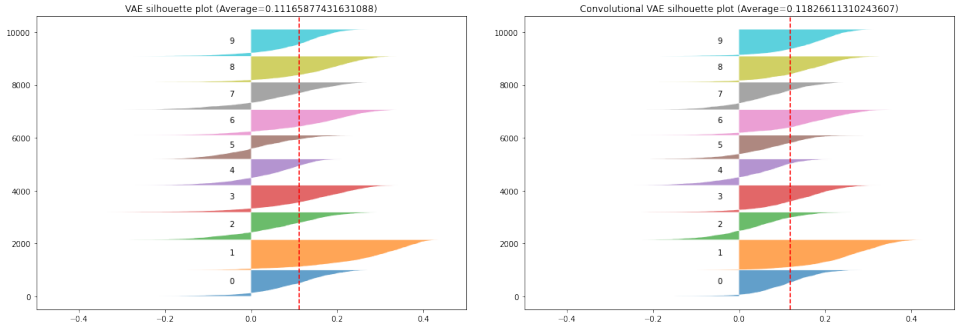


(a) Auto-encoder estándar (AE) (b) Auto-encoder variacional (VAE)

Figura 4.6: Separación de las clases en el AE y en el VAE usando el algoritmo SILHOUETTE

La figura 4.6 muestra los valores calculados usando el algoritmo SILHOUETTE para las representaciones obtenidas mediante el encoder de entrenado usando el auto-encoder estándar (en la figura 4.6a) y mediante el encoder entrenado usando el auto-encoder variacional (en la figura 4.6b). Los valores se han representado en el intervalo [-0.5, 0.5] para facilitar la visualización de los gráficos.

De la misma forma, en la figura 4.7 se ilustran los resultados del algoritmo SILHOUETTE para el auto-encoder variacional (en la figura 4.7a) y para el VAE convolucional (en la figura 4.7b). Al igual que en la figura 4.6 los valores



(a) Auto-encoder variacional (VAE) (b) VAE convolucional (CVAE)

Figura 4.7: Separación de las clases en el VAE y en el CVAE usando el algoritmo SILHOUETTE

se representan usando el intervalo [-0.5, 0.5] para facilitar su visualización.

Los resultados mostrados en la tabla 4.2 evidencian que, desde el punto de vista de la separación de las representaciones, el auto-encoder variacional no proporciona mejores resultados que un auto-encoder estándar. Esto, junto con los resultados obtenidos en los experimentos de generación descritos en el epígrafe 4.3, refuerzan hipótesis de que el espacio latente que construye auto-encoder variacional tiene otras propiedades más interesantes que las de agrupar las representaciones en función de las clases asignadas a los ejemplos originales.

Capítulo 5

Conclusiones

En el capítulo 4 se han realizado un conjunto de experimentos con el auto-encoder variacional y se han incluido los resultados obtenidos tras cada uno de ellos. En este capítulo se ofrecen las conclusiones obtenidas a partir de dichos resultados y se proponen ideas para la continuación del trabajo descrito en esta memoria.

5.1. Resultados de los experimentos

En esta sección se recuperan los resultados obtenidos en los experimentos de generación de imágenes (epígrafe 4.3) y en los experimentos de separación deblas representaciones (epígrafe 4.4) con el objetivo de extraer conlusiones sobre el espacio latente del auto-encoder variacional.

5.1.1. Generación de imágenes

Una de las principales diferencias entre un auto-encoder variacional y un auto-encoder estándar obtenidas en los experimentos de generación del epígrafe 4.3 es que el primer modelo es capaz de generar datos similares a los proporcionados a partir de muestras del espacio latente construido durante el entrenamiento mientras que el segundo, el auto-encoder estándar, casi siempre genera datos que no se parecen a los datos originales.

Esto parece deberse a la forma en que se construyen los espacios en que se representan las entradas. En el auto-encoder variacional, el espacio latente está "delimitado" por una distribución de probabilidad que determina qué

valores del espacio tienen una alta probabilidad de generar datos que tengan sentido desde el punto de vista humano. Es decir, no todos los valores del espacio latente de un VAE tienen la misma "importancia" desde el punto de vista de la generación de datos reales. En el caso del auto-encoder estándar, este espacio no tiene esta propiedad y, desde el punto de vista del decoder, cualquier valor puede producir datos realistas. Esto hace que cuando se use valor del espacio construido con un auto-encoder estándar, los resultados no sean datos realistas.

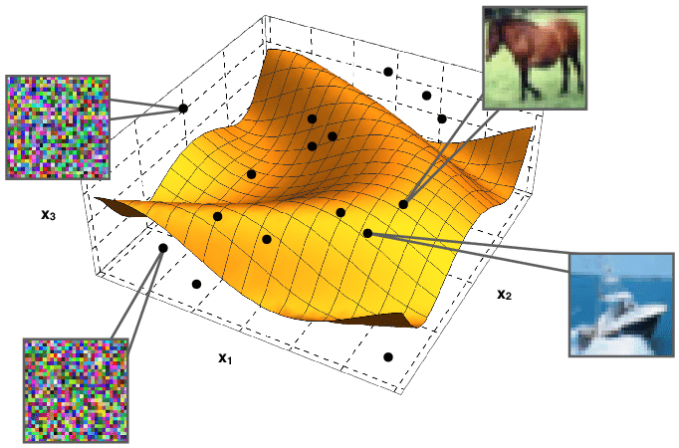


Figura 5.1: Representaciones en el espacio latente[11]

En la figura 5.1 se ilustra el espacio latente que construye un auto-encoder variacional. Los valores de dicho espacio que conducen a imágenes no reconocibles son, precisamente, aquellos valores con una baja probabilidad. En otras palabras, los valores pertenecientes a la superficie son aquellos que se pueden usar para generar ejemplos realistas. En la función de coste del auto-encoder variacional, la función que tiene la divergencia de Kullback-Leibler es dar a esta superficie de la distribución a priori. De esta forma, cuando se muestrea esta distribución, los valores resultantes tienen una alta probabilidad de generar datos realistas.

5.1.2. Separación de las representaciones

Los experimentos sobre la separación de las representaciones en el espacio latente del en el epígrafe 4.4 pretendían determinar en que medida el auto-encoder variacional tiene un mejor rendimiento a la hora de identificar las clases en comparación con un auto-encoder estándar. Sin embargo, los resultados indican que ambos modelos tienen un rendimiento similar.

Este resultado podría deberse a las restricciones impuestas a cada uno de

los dos modelos a la hora de construir el espacio latente. La única restricción impuesta viene determinada por la divergencia de Kullback-Leibler usada en el entrenamiento del auto-encoder variacional. ¿Cómo puede pretenderse que un modelo separe correctamente las representaciones en función de la clase a la que pertenecen los ejemplos usados para obtenerlas si en el proceso de entrenamiento no se ha tenido en cuenta dicha distinción? *La máquina es tonta; sólo sabe sumar y multiplicar* y, por ello, no se puede pretender que lleve a cabo una tarea sin proporcionar la información necesaria para ella.

Una forma de construir el espacio latente para obtener representaciones agrupadas en función de la clase a la que pretenecen los ejemplos originales es ajustar la distribución a priori usada para dar forma a este espacio. Ya se ha trabajado en este sentido y se ha propuesto variantes del auto-encoder variacional como GMM-VAE[7] que usa un modelo gaussiano para construir una distribución a priori adecuada para un espacio latente con propiedades adecuadas al agrupamiento pretendido en este trabajo.

5.2. Limitaciones del trabajo

Para obtener los resultados de los experimentos sólo se ha utilizado el conjunto de datos Mnist. A pesar de que estos datos han sido muy estudiados y, con frecuencia, son muy usados en la evaluación de modelos de tratamiento de imágenes, no se puede descartar que los resultados obtenidos se deban más a los datos usados que al propio modelo desarrollado. Esta circunstancia no sólo limita este estudio concreto sino, además, cualquier estudio posterior en que sólo se use un conjunto de datos para obtener conclusiones.

La forma de solventar esta carencia es simple: usar diversos conjuntos de datos. En particular, una extensión de este trabajo pasa por repetir los experimentos con otros conjuntos de datos como, por ejemplo, Cifar10[18], o CelebA[22]

De la misma forma, es posible que la implementación concreta que se ha propuesto del auto-encoder variacional tenga un efecto similar sobre los resultados de los experimentos. Esto es una característica específica de este trabajo ya que no se prentende analizar si el modelo concreto que se ha implementado funciona mejor o peor que la línea base sino determinar en qué medida los fundamentos teóricos de un modelo suponen un avance en un determinado sentido. Es decir, no podemos extraer conclusiones sobre un modelo teórico partiendo de una única implementación como se hace en este trabajo.

Esta limitación se puede solventar implementando los mismos conceptos teóricos de formas equivalentes. Por ejemplo, una opción es construir el auto-encoder variacional usando TensorFlow Probability, que ofrece un conjunto de facilidades que se ajustan bien a la implementación de este tipo de modelos o usar la biblioteca Pytorch, que es equivalente a la utilizada, para implementar el auto-encoder variacional.

Otra de las limitaciones importantes del trabajo es que los experimentos que se han realizado no son reproducibles. En primer lugar esto provoca que dos ejecuciones de los mismos puedan dar resultados muy diferentes. A pesar de que es posible esta diferencia en los resultados, no se ha observado que haya divergencias significativas. En segundo lugar, ocurre que la comparación entre los modelos no es tan justa como se ha pretendido porque los números pseudo-aleatorios generados en cada ejecución son diferentes. Por ejemplo, los resultados obtenidos en el experimento de generación a partir de los centroides de las clases descrito en el epígrafe 4.3.1, se obtienen valores muy similares para los ejemplos generados por ambos modelos. Hubiese sido muy interesante comprobar cómo de parecidos hubiesen sido los resultados si los experimentos fueran reproducibles.

5.3. Líneas de trabajo futuras

El trabajo es claramente mejorable resolviendo las limitaciones expuestas en el epígrafe 5.2. Sin embargo, al margen de dichas limitaciones han surgido varias cuestiones durante la realización del trabajo relacionadas con el tipo de experimentos que se han llevado a cabo:

- *¿Por qué en el experimento del generación a partir de los centroides descrito en el epígrafe 4.3.1 se obtienen resultados tan parecidos a partir de muestras de espacios diferentes?*

  El experimento propuesto consiste en la generación de nuevos ejemplos a partir de representaciones calculas como la media de representaciones que comparten la clase a la que pertenece el ejemplo a partir del cual se han obtenido. El experimento parecería más completo generando datos a partir de representaciones reales a las que se les ha añadido un ruido controlado. A primera vista este experimento podría provocar que las representaciones perturbadas no tuviesen una representación válida en el auto-encoder estándar mientras que podrían generar una representación distorsionada en el auto-encoder variacional.

- *A la vista de los resultados del experimento de mejora del VAE descrito*

en el epígrafe 4.3.3, ¿en qué medida la mejora de la capacidad del encoder influye en el espacio latente del auto-encoder variacional?

En el citado experimento se ha mejorado tanto el encoder como el decoder. Podría ser más interesante —y seguramente sería más adecuado para los objetivos de dicho experimento— ver en qué medida el rendimiento del decoder es mayor simplemente mejorando el encoder sin mejorar el decoder. Esto seguiría la misma idea que en las comparaciones entre el auto-encoder estándar y el auto-encoder variacional.

5.3.1. Análisis del espacio latente

En el experimento de separación de las representaciones en el espacio latente descrito en el epígrafe 4.4 se ha constatado que el auto-encoder variacional no mejora las capacidades de un auto-encoder estándar para agrupar los ejemplos de entrada en función de la clase a la que pertenecen. Esto apunta a que las representaciones del espacio latente tengan otras propiedades[3] que puedan ser de interés.

Esto apunta directamente a una de las motivaciones principales de este trabajo, que es considerar el auto-encoder variacional más como un modo de representar la entrada que como un modelo generador. Uno de los primeros pasos de esta línea de trabajo podría ser obtener un conjunto de medidas que permitan comparar diversos modelos en función de una característica determinada. Este marco de trabajo constituye una base sólida para el estudio de los modelos usados para representar los datos.

5.3.2. Estudio de las variantes del modelo

En el experimento de mejora del auto-encoder variacional descrito en el epígrafe 4.3.3 se desliza la idea de mejorar el modelo. En dicho experimento se propone una idea de mejora muy simple pero que introduce la posibilidad de variar el VAE con un objectivo concreto: mejorar las capacidades de generación del modelo con el objetivo de hacer el espacio latente más representativo del conjunto de datos usado en el entrenamiento del modelo.

A la hora de mejorar el auto-encoder variacional se puede incidir, principalmente en tres aspectos[29]. La manera más evidente de construir un VAE que obtenga un mejor rendimiento en algún aspecto concreto es modificar su arquitectura. El VAE convolucional usado en los experimentos del epígrafe 4.3.3 es una muestra de ello.

Además, desde el punto de vista de este trabajo, es especialmente interesante el trabajo tanto con la distribución a priori utilizada como con la función de coste con el objetivo de diseñar un mecanismo de regularización que permita "desenmarañar" los factores que caracterizan los ejemplos de entrenamiento. Esta idea se fundamenta en una hipótesis denominada *manifold hipothesis*[10] que asegura, de forma simple, que los datos son generados por un reducido conjunto de factores. Obtener estos factores y "desenmarañarlos" se conoce como *disentanglement*, que es una de las propiedades que busca el modelo β-VAE usado en la implementación descrita en el capítulo 3.

Bibliografía

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. *Philosophical transactions of the Royal Society of London*, (53):370–418, 1763.

[3] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[4] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[5] Marissa Connor, Gregory Canal, and Christopher Rozell. Variational autoencoder with learned latent structure. In *International Conference on Artificial Intelligence and Statistics*, pages 2359–2367. PMLR, 2021.

[6] Peter Dayan, Geoffrey E. Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

[7] Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*, 2016.

[8] Ming Ding. The road from mle to em to vae: A brief tutorial. *AI Open*, 2021.

[9] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[10] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

[11] Sebastian Goldt, Marc Mézard, Florent Krzakala, and Lenka Zdeborová. Modeling the influence of data structure on learning in neural networks: The hidden manifold model. *Physical Review X*, 10(4):041044, 2020.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[13] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[14] Geoffrey E. Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2002.

[15] Diederik P. Kingma. Variational inference & deep learning: A new synthesis. 2017.

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

[17] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[18] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[19] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[20] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[21] Yang Li, Quan Pan, Suhang Wang, Haiyun Peng, Tao Yang, and Erik Cambria. Disentangled variational auto-encoder for semi-supervised learning. *Information Sciences*, 482:73–85, 2019.

[22] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

[23] Emile Mathieu, Tom Rainforth, Nana Siddharth, and Yee Whye Teh. Disentangling disentanglement in variational autoencoders. In *International Conference on Machine Learning*, pages 4402–4412. PMLR, 2019.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[25] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[26] Stephen Odaibo. Tutorial: Deriving the standard variational autoencoder (vae) loss function. *arXiv preprint arXiv:1907.08956*, 2019.

[27] George Orwell. *Animal Farm*. Secker and Warburg, London, 1945. (Traducción de Rafael Abella para Destinolibro).

[28] Peter J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[29] Ruoqi Wei, Cesar Garcia, Ahmed El-Sayed, Viyaleta Peterson, and Ausif Mahmood. Variations in variational autoencoders-a comparative evaluation. *Ieee Access*, 8:153651–153670, 2020.

[30] Lilian Weng. From autoencoder to beta-vae. `https://lilianweng.github.io/posts/2018-08-12-vae/`, 2018.

[31] Ronald Yu. A tutorial on vaes: From bayes' rule to lossless compression. *arXiv preprint arXiv:2006.10273*, 2020.

Doce voces gritaban enfurecidas, y eran todas iguales. No había duda de la transformación ocurrida en la cara de los cerdos. Los animales asombrados, pasaron su mirada del cerdo al hombre, y del hombre al cerdo y, nuevamente, del cerdo al hombre; pero ya era imposible distinguir quién era uno y quién era otro.

—Rebelión en la granja[27]