Universidad Zaragoza
1542

Trabajo Fin de Grado

Diseño de Algoritmos de Clusterización para la Regularización de Redes Neuronales y Aprendizaje de Características Relevantes

Design of Clustering Algorithms for Neural Network Regularization and Relevant Feature Learning

Autor

Javier Antorán Cabiscol

Director

Dr. Antonio Miguel Artiaga

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2018

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

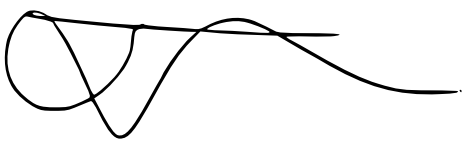
D./Dª. Javier Antorán Cabiscol,

con nº de DNI 31014934D en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Diseño de Algoritmos de Clusterización para la Regularización de Redes Neuronales y Aprendizaje de Características Relevantes,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 25 de Junio, 2018



Fdo: Javier Antorán Cabiscol

Diseño de Algoritmos de Clusterización para la Regularización de Redes Neuronales y Aprendizaje de Características Relevantes

RESUMEN

En este trabajo, exploramos técnicas novedosas de *representation learning*. Analizamos la red de cápsulas recientemente introducida y sus métodos de regularización. Presentamos una técnica de visualización de información en redes neuronales convolucionales en la que superponemos a las activaciones espaciales sus correspondientes campos receptivos. Esta nos permite ver los factores en función de los cuales nuestra red separa la información. Proponemos un nuevo método de clusterizado para las activaciones de la última capa de redes clasificadoras basado en un coste por margen. Demostramos su utilidad como método para obtener medidas robustas de incertidumbre sobre las decisiones que toma el clasificador. Adoptamos un marco probabilístico Bayesiano, proponiendo un algoritmo de autoencoder variacional novedoso. Al condicionar algunas variables latentes con valores discretos, conseguimos captar características de los datos distribuidas multimodalmente. Mostramos cómo este algoritmo permite obtener representaciones más desentrelazadas y de mayor calidad que los propuestos en la literatura de autoencoders variacionales. Proponemos un método para comparar la fidelidad de modelos generativos, entrenando un clasificador con bases de datos aumentadas con muestras generadas. Validamos experimentalmente que nuestro modelo consigue generar muestras nuevas más informativas que los modelos comparables de la literatura.

Índice

Capítulo 1

Introducción, conceptos básicos y objetivos

Los humanos tenemos la capacidad de comprender conceptos y resolver problemas, a la cual llamamos inteligencia. A su vez, entendemos por inteligencia artificial (IA) la capacidad de nuestras herramientas de realizar operaciones comparables a las que realiza la mente humana o que, a priori, requerirían de inteligencia humana.

La inteligencia artificial ha existido como disciplina desde la década de 1950 con la invención de modelos matemáticos como el del perceptron [Rosenblatt, 1958]. Sin embargo, debido a recientes avances en los algoritmos de IA, a los incrementos en la capacidad computacional de la que disponemos y a la generación masiva de datos, hoy en día, la IA, y más específicamente los algoritmos de aprendizaje automático (*machine learning*, ML), nos permiten obtener resultados comparables a los de la mente humana en una gran cantidad de tareas, e incluso superarla en algunas de ellas.

1.1. Las redes neuronales

Dentro del ML, las redes neuronales son el tipo de modelo que más éxito ha alcanzado en los últimos años. El *deep learning* (DL), entendido como la subdisciplina de ML en la que se utilizan redes neuronales con más de una capa oculta, ha permitido realizar grandes avances en una diversidad de campos: La visión por computador con redes [Krizhevsky et al., 2012], [He et al., 2015], [Zagoruyko and Komodakis, 2016], [Xie et al., 2016], el reconocimiento [Zeghidour et al., 2018] y generación de voz [van den Oord et al., 2016], el procesado de lenguaje natural [Bahdanau et al., 2014], [Mikolov et al., 2013] y los juegos de estrategia [Silver et al., 2017]. En todos ellos, las redes neuronales profundas proporcionan los mejores resultados hasta la fecha.

El término "red neuronal" (neural network, NN) tiene su origen en los intentos de encontrar representaciones matemáticas de sistemas de procesado de información biológicos como los propuestos por [McCulloch and Pitts, 1943] y [Van Der Malsburg, 1986]. Sin embargo, en este trabajo trataremos las redes neuronales como modelos de reconocimiento de patrones estadísticos.

Las redes son un modelo gráfico computacional, compuesto por operaciones lineales y no lineales alternantes. Los parámetros del modelo se obtienen utilizando un algoritmo de optimización. Los más comunes están basados en descenso de gradiente [Ruder, 2016].

Las redes neuronales convolucionales (CNN) son aquellas en las que los coeficientes

de las capas lineales están relacionados entre sí de forma que se pueden entender como filtros con los que se convoluciona la señal de entrada para dar la de salida. Las CNN fueron introducidas por primera vez para resolver la tarea de reconocimiento de dígitos [Lecun et al., 1998]. Desde entonces se han convertido en el tipo de red más común para el procesado de imagen, aunque se utilizan también en otras tareas. Una ventaja que presenta la capa convolucional es la invariancia ante translaciones espaciales de los datos de entrada. A diferencia del perceptron, que utiliza un peso distinto para cada punto de la imagen de entrada, la capa convolucional solo guarda los coeficientes de un filtro. De esta manera, al hacer la convolución, las características de la imagen que estén correladas con los pesos del filtro presentarán una salida alta, independientemente de su posición en la señal de entrada. Además, la capa convolucional es menos flexible que la de un perceptron, ya que al reutilizar los parámetros del filtro para los distintos píxeles de entrada, tiene menos coeficientes.

A priori, puede parecer que un modelo flexible es deseable para poder modelar relaciones complejas en nuestros datos. Sin embargo, un modelo demasiado flexible puede sobreajustarse al ruido presente en estos. Esta situación es análoga a intentar realizar una regresión lineal con más parámetros que datos. Aunque obtendremos unos residuos nulos, nuestro modelo no tendrá capacidad de predicción.

El *representation learning* engloba el conjunto de métodos que permiten a una máquina aprender a generar representaciones útiles de los datos al ser alimentada por estos [Bengio et al., 2013]. Los métodos de deep learning aprenden a generar estas representaciones de forma distribuida entre las distintas capas de una red neuronal. De esta forma, las primeras capas de una red de análisis de imagen aprenderán a detectar bordes en la imagen. Las siguientes buscarán composiciones de los bordes aprendidos en las capas anteriores. Así, la información fluye por la red, organizándose para representar conceptos más abstractos en sucesivas capas [Bengio, 2009]. Como consecuencia, se pueden llegar a modelar funciones verdaderamente complejas. El uso de redes neuronales permite convertir problemas de ingeniería de extracción de características en problemas de ingeniería de arquitecturas de redes y de optimización [LeCun et al., 2015]. Una explicación más detallada de las redes neuronales y su terminología se encuentra en el Apéndice A.

1.2. Planteamiento del trabajo

La búsqueda de métodos capaces de aprender representaciones útiles de los datos y la regularización de modelos para evitar el sobreajuste pueden entenderse como un mismo objetivo. Al hacer que un sistema utilice su capacidad de representación para modelar la estructura subyacente de los datos, y no el ruido, nos estamos asegurando de que aprende representaciones útiles para nuestras tareas. Buscar nuevas formas de conseguir esto es uno de los problemas fundamentales del machine learning y será el propósito de este trabajo.

Abarcaremos esta tarea desde el punto de vista del tratamiento de la información. Estudiaremos cómo los modelos de machine learning interactúan con la información y cómo los podemos modificar para que aprendan a organizarla en representaciones que nos sean útiles. Nos centraremos en algoritmos de agrupación de datos similares (clusterización) y modelos de variables latentes.

Este proyecto se divide en dos fases, una primera de exploración y una posterior de explotación. En la primera, realizamos una revisión de la literatura de técnicas de deep learning, centrándonos en la red de cápsulas [Sabour et al., 2017], analizándola

e implementándola. Extraemos una interpretación teórica propia de los mecanismos y estrategias que utiliza este tipo de red desde el punto de vista de la regularización y el tratamiento de la información. A continuación, buscamos cómo aplicar estas estrategias para afrontar distintos problemas dentro del ML. Se trata de una fase de exploración ya que probamos distintos algoritmos, realizando pequeñas baterías de experimentos con cada uno de ellos. En cada paso de esta fase, elegimos los algoritmos a utilizar y los experimentos a realizar en función de los resultados de la anterior batería de pruebas que nos parecen más prometedores. Esta fase termina al dar con un algoritmo de Inferencia Bayesiana Variacional que nos da especialmente buenos resultados. Aquí comienza la segunda fase del trabajo, en la que realizamos experimentación extensiva sobre este algoritmo. Además, implementamos distintos algoritmos de la literatura reciente de inferencia variacional y modelos generativos, comparando sus prestaciones con las del nuestro.

Los principales hitos de este trabajo son las siguientes.

- Realizamos un análisis e interpretación propia de las redes de cápsulas propuestas por [Hinton et al., 2011], [Sabour et al., 2017] y [Hinton et al., 2018].
- Proponemos un método para visualizar cómo las capas convolucionales de una CNN segmentan la información.
- Proponemos un algoritmo de clusterización supervisado basado en un coste por margen. A diferencia de algoritmos existentes, este no solo promueve la compactación intraclase sino que también tiene como objetivo la separación entre clases.
- Demostramos la utilidad de dicho algoritmo para obtener medidas de incertidumbre para las decisiones tomadas por redes neuronales.
- Proponemos un algoritmo de clusterizado no supervisado basado en el algoritmo *centerloss* introducido por [Wen et al., 2016].
- Revisamos la literatura reciente de métodos de inferencia variacional Bayesiana para *representation learning*. Nos centramos en el autoencoder variacional (VAE) propuesto por [Kingma and Welling, 2013] y en los métodos de desentrelazado.
- Proponemos un algoritmo de inferencia variacional novedoso, basado en el VAE, que permite un mayor control sobre como el modelo representa la información y que genera nuevas muestras de forma más expresiva que anteriores modelos.
- Proponemos un método novedoso para evaluar la fidelidad de modelos generativos que extiende al propuesto por [Odena et al., 2017] y lo utilizamos para validar la efectividad de nuestro algoritmo de inferencia variacional.

Capítulo 2

Estudio de la red de cápsulas

La recientemente introducida red de cápsulas ha recibido mucha atención en la comunidad de ML por sus novedosos métodos de regularización y *representation learning*. En este capítulo, analizamos esta red, ya que su mecanismo de reconstrucción será una parte importante del sistema propuesto en el Capítulo 4. Primero, resumimos su funcionamiento y la intuición detrás de sus algoritmos presentada en [Hinton et al., 2011], [Sabour et al., 2017] y [Hinton et al., 2018]. A continuación, planteamos una breve reinterpretación de las estrategias utilizadas por esta red y estudiamos cómo pueden ser aplicadas a problemas de machine learning de forma más general.

2.1. Síntesis del algoritmo e intuición

Desde su concepción, los sistemas de visión por computador han intentado imitar el funcionamiento de la visión humana. Tras la competición de clasificación de imagen LSVRC-2012 Imagenet, en la que [Krizhevsky et al., 2012] consiguió reducir la tasa de error a casi la mitad utilizando una red neuronal convolucional, las técnicas de deep learning se convirtieron en la solución estándar a los problemas de visión por computador. Las redes de cápsulas presentan un enfoque aun más similar a la visión humana y por tanto pueden ser consideradas como el siguiente paso hacia una visión por computador biológicamente plausible.

La contribución más importante de estas redes es el mecanismo de conexión o enrutamiento entre las neuronas de la red. Las activaciones de las redes convolucionales tradicionales, a la salida de los nodos de cada capa, se organizan en mapas de píxeles. La red de cápsulas sustituye estos nodos por cápsulas cuyas salidas son vectores multidimensionales. Esto permite que, a diferencia de las CNN donde el esquema de las conexiones entre los nodos de distintas capas es fijo, las cápsulas utilicen enrutamiento dinámico.

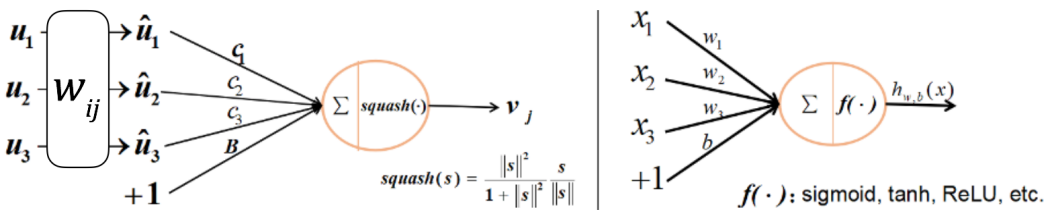


Figura 2.1: A la izquierda, la estructura de una cápsula. A la derecha, un nodo de una red neuronal tradicional.

El ángulo de cada vector a la salida de las cápsulas codifica las características de un objeto detectado por la red en la imagen. El módulo representa la confianza de existencia de este objeto en la imagen. Denominaremos u_i al vector a la salida de la capa i. Una comparación entre una cápsula y una neurona tradicional se puede observar en Figura 2.1.

$$\hat{u}_{j|i} = W_{ij} u_i \tag{2.1}$$

$$s_j = \sum_i c_{ij} \hat{u}_{j|i} \tag{2.2}$$

Para generar los vectores de entrada a cada una de las cápsulas de la siguiente capa, multiplicamos u_i por la matriz de pesos W_{ij}. Así, obtenemos $\hat{u}_{j|i}$ (2.1). Cada uno de los vectores $\hat{u}_{j|i}$ se introducen al algoritmo de enrutamiento, el cual asigna a cada uno de ellos un coeficiente de ponderación c_{ij} (2.2). El enrutamiento identifica similitudes entre los vectores de entrada y asigna pesos grandes a los vectores en los que las encuentra. Dada una cápsula inferior, sus coeficientes de enrutamiento para todas las cápsulas superiores deben sumar 1 $\sum_i c_i$=1. Existen dos propuestas de algoritmos de enrutamiento. El de [Sabour et al., 2017] está basado en la correlación entre vectores mientras que [Hinton et al., 2018] proponen un método similar al algoritmo de clusterizado EM.

Tras el proceso de enrutamiento, se aplica una no linealidad, llamada *squash* a cada uno de los vectores resultantes. Podemos verla en (2.3). Esta, mantiene el ángulo de vector pero reduce su módulo al rango $[0, 1]$, permitiéndonos interpretarlo como una probabilidad a la salida de la red.

$$v_j = \frac{s_j \|s_j\|}{1 + \|s_j\|^2} \tag{2.3}$$

Esta técnica busca modelar el proceso visual humano, en el que reconocemos objetos complejos como composiciones de objetos más sencillos. Las primeras capas de la red aprenderán a reconocer objetos sencillos. o partes de ellos. Las posteriores aprenderán a construir conceptos más elaborados a partir de las detecciones de las capas anteriores.

Finalmente, la última capa genera un vector multidimensional por cada clase posible. El módulo de cada uno de estos vectores indica la probabilidad de que la imagen de entrada pertenezca a una clase.

La red se entrena con una función de coste no estándar llamada *margin loss*. Esta genera gradientes con menos pendiente que la función de entropía cruzada utilizada habitualmente. Como consecuencia, ayuda a evitar el sobreajuste. Además, el vector de clase con mayor módulo se utiliza como información base para reconstruir la imagen de entrada a la red. La función de coste incluye un término adicional, cuyo valor es la distancia euclídea entre esta reconstrucción y la imagen original. El objetivo de esto es forzar a la red a codificar información útil sobre los objetos detectados en las distintas dimensiones del vector.

El funcionamiento de la cápsulas se puede ilustrar mejor con un ejemplo. En la figura Figura 2.2, representamos un mapa de vectores a la salida de una capa de cápsulas. Los vectores azules corresponden a la detección de un triangulo mientras que los grises a la de un rectángulo. En este caso, el ángulo del vector codifica la rotación del objeto detectado. Cabe mencionar que, en la práctica, estos vectores tendrán más de dos dimensiones y codificarán más información que solo un ángulo. Podrían guardar la información del tamaño de la detección, deformaciones del objeto, etc. A la derecha de la figura, encontramos un barco y una casa, dos objetos modelados por la capa siguiente a partir de los objetos de la capa anterior.

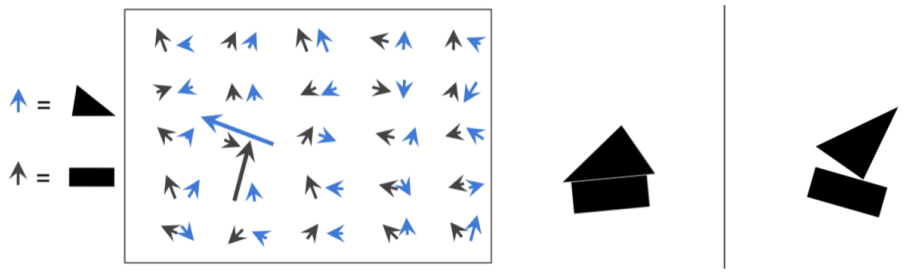


Figura 2.2: A la izquierda, los mapas de vectores a la salida de dos cápsulas, una que busca rectángulos (vectores grises) y otra que busca triángulos (vectores azules). A la derecha, dos objetos que se pueden construir con un rectángulo y un triángulo, una casa y un barco.

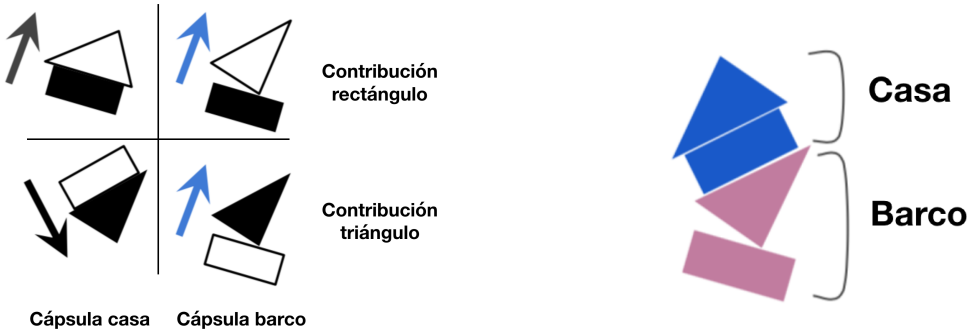


Figura 2.3: A la izquierda, las propuestas de vectores de activación de capas superiores para cada vector de activación de la capa inferior. En el caso del barco, las predicciones concuerdan mientras que para la casa no. A la derecha, una imagen con un barco y una casa clasificados correctamente.

El algoritmo de enrutamiento se puede entender como un proceso en el que cada cápsula del nivel inferior predice las características, en este ejemplo la rotación, del objeto de la cápsula superior. Si las predicciones de varias cápsulas inferiores concuerdan, aumentan sus coeficientes y se fortalecen sus contribuciones a la cápsula superior. Como podemos ver en la parte de la derecha de la figura Figura 2.3, tanto la predicción del rectángulo como la del triángulo concuerdan en el caso del barco. Sin embargo, la casa propuesta por cada cápsula anterior tiene un ángulo distinto. La cápsula superior correspondiente al barco daría una activación con mayor módulo.

Una ventaja interesante de este algoritmo es su capacidad para decodificar imágenes con múltiples objetos, una tarea tradicionalmente difícil para los algoritmos de visión por computador. Debido a la restricción $\sum_i c_i{=}1$, cada cápsula inferior podrá contribuir significativamente a un número limitado de cápsulas superiores. De esta manera, cada grupo de cápsulas es responsable de una parte de la imagen, evitando que se confundan características de objetos distintos entre sí. Esto lo podemos ver en la parte derecha de la Figura 2.3. A pesar de la proximidad de los objetos base, la casa y el barco son identificados correctamente.

2.2. Implementación

Implementamos la red de cápsulas propuesta en [Sabour et al., 2017] en Python, haciendo uso de la librería de machine learning PyTorch [1]. Utilizamos la arquitectura de computación paralela CUDA[2]. Esta nos permite entrenar redes neuronales en tarjetas gráficas con gran rapidez, ya que las operaciones realizadas por estos modelos se pueden expresar como productos matriciales. El cómputo de estos es fácilmente paralelizable. Entrenamos el modelo sobre el cluster de computación del grupo de investigación ViVoLab[3]. La arquitectura concreta de la red implementada y el pseudo-código del algoritmo de enrutamiento se pueden encontrar en [Sabour et al., 2017].

Desafortunadamente, los algoritmos de enrutamiento entre cápsulas son de carácter iterativo. Como los datos de entrada para cada iteración dependen de los resultados de la anterior, no se trata de un proceso paralelizable. Esto ralentiza el entrenamiento de forma muy significativa comparado con redes convencionales de similar tamaño.

En su publicación, [Sabour et al., 2017], no especifican si se debería computar el gradiente del error respecto de los parámetros de enrutaminto en cada iteración del algoritmo o sencillamente ignorar el proceso iterativo y tratar los pesos de enrutamiento como constantes en la fase de *backpropagation*. Después de probar ambos métodos, nos decantamos por la segunda opción. Los resultados obtenidos con ambas son iguales pero no computar los gradientes de cada iteración proporciona una ligera disminución en coste computacional.

Entrenamos la red para resolver la tarea de clasificación de dígitos MNIST [LeCun and Cortes, 2010]. Se trata de una base de datos de 70000 números escritos a mano. Empleamos 60000 muestras para entrenar nuestro modelo y 10000 para evaluarlo. Hacemos uso de la misma técnica de aumento de datos empleada por [Sabour et al., 2017], desplazamiento aleatorio de 2 píxeles con relleno de ceros. Utilizamos el optimizador ADAM [Kingma and Ba, 2014] con el régimen de decaimiento exponencial de la tasa de aprendizaje propuesto en [Sabour et al., 2017].

Tabla 2.1: Error de clasificación en la base de datos MNIST. La media y la desviación estándar se calculan a partir de 3 experimentos.

Implementación	media (%)	std (%)
[Sabour et al., 2017]	0.25	0.0005
Nuestra	0.30	0.002

Como se muestra en la Tabla 2.1, obtenemos un error de clasificación de 0.3 %, competitivo con los mejores resultados en MNIST. Aun así, no conseguimos reproducir los resultados de [Sabour et al., 2017].

2.3. Nuestro análisis de las cápsulas

En un sistema de procesado tradicional, filtraríamos cada canal de la señal de forma independiente. En una CNN, a cada canal de la imagen de entrada se le aplican tantos filtros como canales hayamos definido en la siguiente capa. Para formar cada canal de la capa

[1] https://pytorch.org
[2] https://developer.nvidia.com/about-cuda
[3] http://vivolab.unizar.es

superior, se suman las contribuciones de todos los canales de la capa inferior. De esta manera, una capa convolucional tendrá $N_{c1} \times N_{c2}$ filtros. Esta conexión de todos a todos en los canales de capas adyacentes, hacen estos modelos muy flexibles. Como resultado, es común que los pesos de la red se sobreajusten a los datos de entrenamiento.

Una posible solución a esto es el uso de convoluciones por grupo. Este método fue empleado por primera vez por [Krizhevsky et al., 2012]. Desde entonces se ha popularizado con ejemplos de uso en [Gastaldi, 2017], [Xie et al., 2016] y [Zhang et al., 2017]. Consiste en establecer grupos de convoluciones de manera que las salidas de un filtro solo se transmiten a los filtros de la siguiente capa del mismo grupo.

Otra solución posible es el uso de *attention*. Esta técnica, propuesta por [Bahdanau et al., 2014], permite a una red neuronal centrarse exclusivamente en un subconjunto de sus entradas. Esto se consigue multiplicando los datos de entrada con una máscara generada dinámicamente. En lugar de procesarse todas las señales, la red elige las más relevantes y procesa esas.

El algoritmo de enrutamiento de cápsulas se puede entender como un mecanismo dinámico de creación de grupos de convolución. La información de cada cápsula solo se propaga a algunas de las cápsulas de la capa siguiente. La decisión de a cuales lo hace se toma dinámicamente mediante el enrutamiento. Además, se aplica una forma de *attention*, ya que si las características detectadas por una cápsula no son coherentes en el contexto de las detectadas por las demás cápsulas de la capa, no se propaga su información. Es esta interpretación, la que nos lleva a perseguir la aplicación de estas ideas a sistemas de ML convencionales, fuera del entorno de las cápsulas. En el Capítulo 3, intentaremos buscar formas de visualizar cómo la red representa la información. Además, intentaremos utilizar técnicas de clusterizado, similares a la propuesta en [Hinton et al., 2018] para agregar esta información en representaciones útiles y regularizar las redes convolucionales.

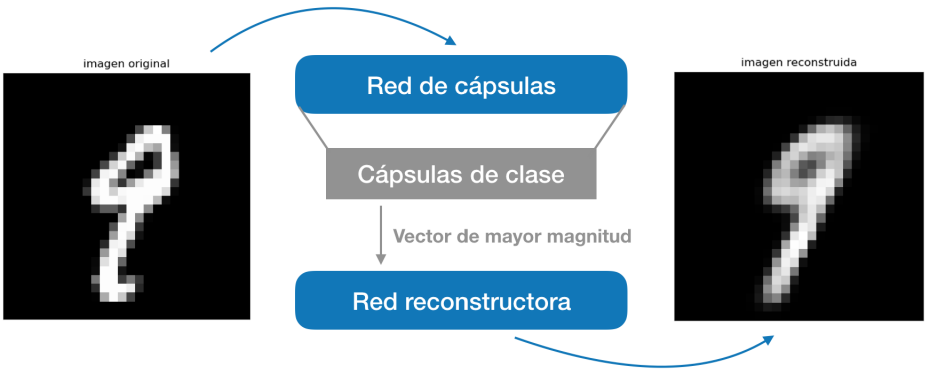


Figura 2.4: Estructura de autoencoder con la red de cápsulas como codificador. Las dimensiones del vector de clase de mayor módulo hacen de características de baja dimensión. Estas son la entrada al decodificador o reconstructor.

En su red de cápsulas, [Sabour et al., 2017] introducen un error de reconstrucción como regularización. Sin embargo, nosotros identificamos que la estructura conjunta de la red de cápsulas y la red reconstructora es equivalente a la de un autoencoder. Este tipo de red busca aprender a proyectar los datos de entrada sobre un espacio de bajas dimensiones. Esta proyección debe mantener la máxima información posible para luego poder reconstruir fielmente la entrada a partir de los datos comprimidos. Como podemos ver en Figura 2.4, las dimensiones de las cápsulas de clase toman el lugar de los datos codificados o comprimidos.

Los autoencoders se encuentran explicados en mayor detalle en el Apéndice A.

La mayor diferencia entre un autoencoder y las cápsulas con regularización de reconstrucción es la introducción de la información de clase. Tradicionalmente, los autoencoders son algoritmos no supervisados, los cuales intentan modelar la distribución de los datos de entrada sin intentar clasificar cada muestra. Al utilizar solo las dimensiones del vector ganador como datos para el decodificador, se crea una dependencia entre clasificar bien un dígito y reconstruirlo bien. Además, mientras que las redes de clasificación convencionales van reduciendo la información de entrada hasta la mínima indispensable para clasificar en la última capa, la reconstrucción obliga a la red de cápsulas a propagar la información sobre las características de la imagen hasta el final de la red. Esto es importante para el correcto funcionamiento del algoritmo de enrutamiento.

En la Figura 2.5 representamos un barrido de las reconstrucciones de un dígito al variar la amplitud de cada dimensión del vector de clase sumándole valores equiespaciados en el rango $[-0.5, 0.5]$. Observamos que pequeños cambios en las dimensiones del vector de clase producen variaciones importantes en la reconstrucción. Obtenemos un espacio no estructurado en el que los factores de variación de la reconstrucción, como el grosor de trazo o inclinación, están compartidos entre múltiples dimensiones. Cuando ocurre esto, se dice que es una representación "entrelazada".

Estas ideas nos incitan a explorar la posibilidad de incorporar información de clase en modelos generativos en el Capítulo 4, haciendo uso del autoencoder variacional [Kingma and Welling, 2013]. Este nos permite aprender representaciones estructuradas e interpretables de la información.

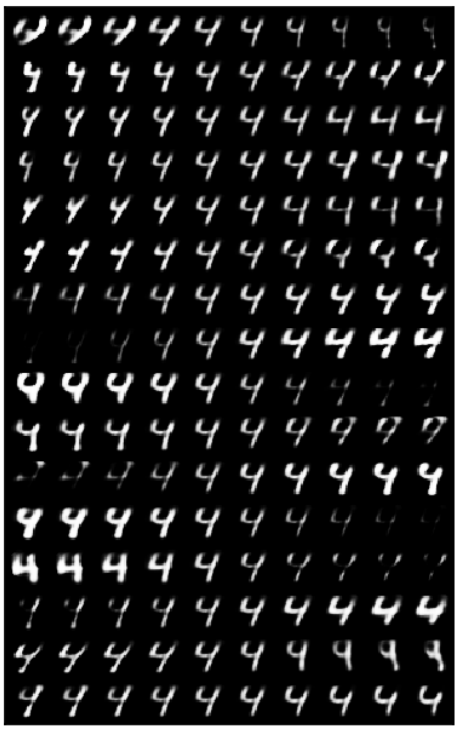


Figura 2.5: Reconstrucción de la imagen de entrada a la red de cápsulas sumando a la amplitud de cada dimensión en 10 pasos, uno por cada columna, un valor en el rango $[-0.5, 0.5]$. Hay una fila por cada una de las 16 dimensiones de la cápsula de clase. Observamos que, para variaciones grandes, no se llega a reconstruir nada o se pueden llegar a reconstruir aberraciones que no se parecen al dígito original.

Capítulo 3

Clusterizado y visualización de representaciones

¿Pueden los conceptos introducidos en el análisis de la red de cápsulas ser aplicados a otros tipos de redes? Intentaremos responder a esta pregunta en la fase de exploración de este trabajo. En ella, probamos una amplia variedad de técnicas y algoritmos. Dedicamos este capítulo a describir los más representativos, haciendo especial énfasis en los algoritmos de clusterizado, en su uso para obtener medidas de incertidumbre en redes neuronales y en los resultados que finalmente nos llevan a la adopción del marco de la inferencia variacional en el siguiente capítulo.

3.1. Arquitectura de red y Entorno experimental

Los experimentos de esta sección se realizarán sobre una CNN pequeña, compuesta por capas convolucionales y capas densas. Estas últimas también son conocidas como lineales, o *fully connected* (FC). Nos decidimos por un modelo sencillo con un comportamiento fácilmente interpretable. De esta manera, nos será mas fácil el análisis de los resultados de los experimentos realizados.

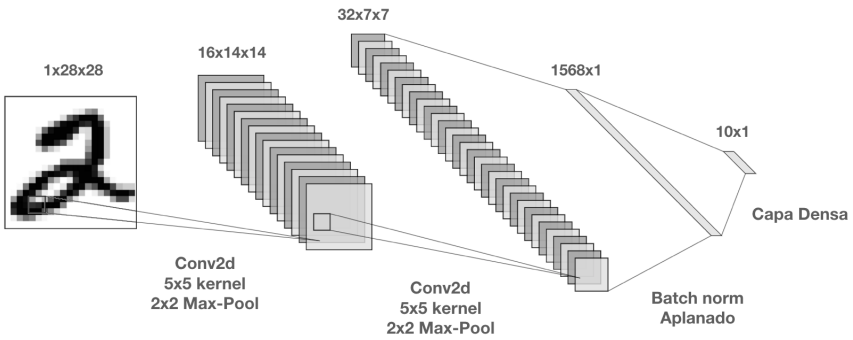


Figura 3.1: Red utilizada en experimentos de clusterizado y visualización. Vemos apilados los mapas de activaciones, o canales, tras cada capa convolucional. Cada cuadrado representa un canal de la imagen. Tras la operación de aplanado (conversión de tensor a vector), la información de toda la imagen queda representada en un vector.

La arquitectura utilizada se puede ver en la Figura 3.1. En total, cuenta con 30 mil parámetros. Estos se organizan en dos capas convolucionales con filtros de tamaño 5×5 y en una capa densa. Esta última actúa de clasificador lineal, tomando como entrada los píxeles de

salida de los filtros y devolviendo 10 valores, uno por cada clase. Tras cada capa convolucional, realizamos un diezmado de 2×2 píxeles con la técnica *max-pooling*. Es decir, por cada ventana cuadrada de 4 píxeles de la imagen, nos quedamos solo con el píxel de mayor amplitud. Entre la última capa convolucional y la capa lineal, aplicamos la técnica de *batch normalization* propuesta por [Ioffe and Szegedy, 2015].

Tras la última capa, aplicamos la función *softmax* para convertir las puntuaciones de clase en valores positivos cuya suma vale 1. La función de coste será la entropía cruzada (CE). No empleamos técnicas de aumento de datos u otros métodos de regularización ya que podrían afectar a los resultados de nuestros experimentos. Una descripción más detallada de esta red se encuentra en el Apéndice B.

Programamos nuestros modelos con Python, utilizando la librería de deep learning PyTorch. Realizamos los experimentos sobre la plataforma Google Colaboratory. Esta, nos da acceso a una maquina virtual en forma de interprete de Jupyter Notebook e incorpora aceleración de cómputo con CUDA.

3.2. Visualización de representaciones

Diseñamos una técnica para poder ver cómo la red segmenta la información y qué características aprenden a buscar los filtros. Combinamos métodos de reducción de dimensionalidad, con la representación de activaciones espaciales y la superposición de sus respectivos campos receptivos.

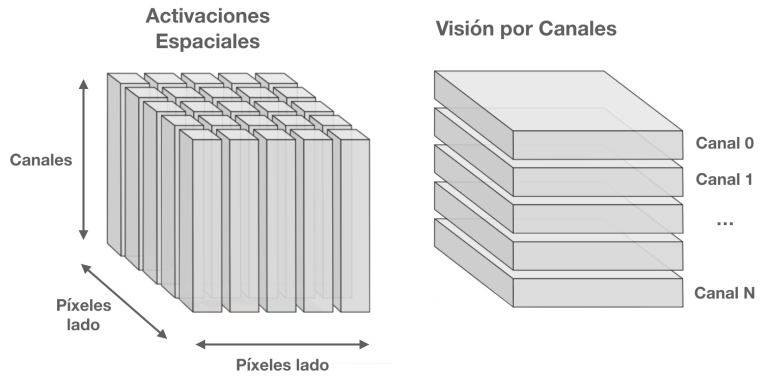


Figura 3.2: A la izquierda, esquema de la organización de las activaciones de la red como vectores multidimensionales. Estos son conocidos como activaciones espaciales. A la derecha, organización tradicional de la misma información, por canales.

Las activaciones espaciales [Olah et al., 2018] de una capa de la red son vectores de tantas dimensiones como canales tiene la imagen. Cada vector se corresponde con un píxel. La amplitud del píxel en cada canal será el valor de cada dimensión del vector. Este concepto queda ilustrado en la Figura 3.2. Para poder representar las activaciones espaciales en dos dimensiones, emplearemos técnicas de reducción de dimensionalidad como PCA [Bishop, 2006], t-SNE [van der Maaten and Hinton, 2008], o una capa de reducción de dimensiones en la red. Está última queda esquematizada en la figura Figura 3.3. Cabe destacar que solo utilizaremos esta técnica para generar representaciones, no para obtener resultados numéricos, ya que el cuello de botella de información que introduce podría afectar al rendimiento del modelo.

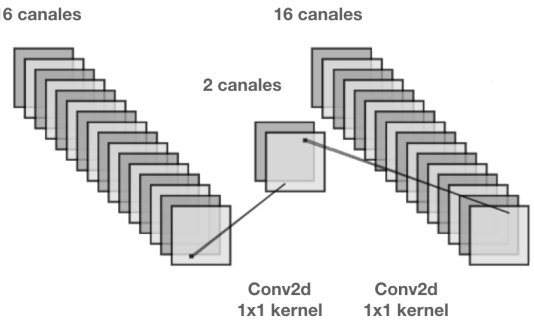


Figura 3.3: Técnica de reducción de dimensionalidad para la visualización. La red aprende una transformación lineal a un espacio de dos canales y una transformación de vuelta a un espacio con el número de dimensiones original. En dos dimensiones podemos ver cómo la red segmenta la información.

Tras comparar la técnica de reducción de dimensionalidad descrita en la Figura 3.3 con t-SNE y con PCA, decidimos que es la mejor para nuestras tareas. Por tanto, es la que utilizamos para representar espacios de altas dimensiones en el resto del capítulo. Los detalles de nuestras comparaciones se pueden encontrar en el Apéndice C.

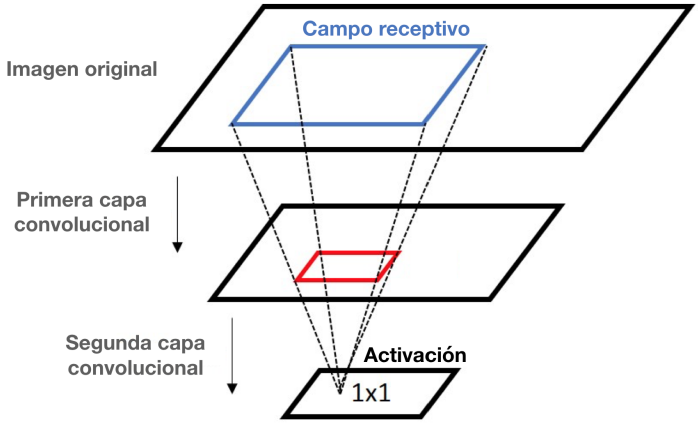


Figura 3.4: Campo receptivo para un píxel (activación) de la segunda capa convolucional de una red. El espacio marcado en azul es la zona de la imagen original de la que depende el cálculo de la activación en cuestión.

En una CNN, el campo receptivo de una activación de la red, se refiere a la sección de la imagen de entrada de la que depende el valor de dicha activación. Este concepto queda ilustrado en la Figura 3.4. El tamaño del campo receptivo dependerá del tamaño de los filtros usados, el avance de estos filtros y de las capas de pooling utilizadas. Describimos cómo calcular el campo receptivo para las activaciones de cualquier capa convolucional de una CNN en el Apéndice C.

Visualizar las activaciones espaciales de una red nos permite ver la distribución de la información a la salida de las capas convolucionales. Superponer los campos receptivos nos permite interpretar esta distribución. Así, podemos ver en función de qué factores de la imagen de entrada se segmenta la información.

En la Figura 3.5 se puede ver un ejemplo de la aplicación de esta técnica a las activaciones de la primera capa de nuestra red. Se trata de un espacio de 16 dimensiones, con 14×14 píxeles por imagen. A través de la superposición de los campos receptivos a las activaciones

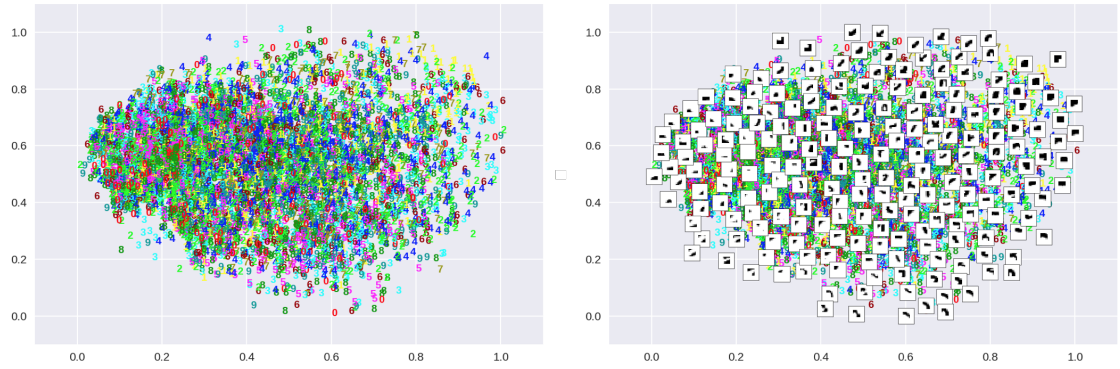


Figura 3.5: A la izquierda, las activaciones espaciales de la primera capa reducidas a dos dimensiones con la técnica de la Figura 3.3. Datos tomados del conjunto de pruebas (*test set*). Cada clase se representa con un color distinto. No se observa segmentación por clase. A la derecha, los mismos puntos con los campos receptivos de algunos de ellos superpuestos. Se observa que, la izquierda, se agrupan los trazos más finos. A la derecha, los más gruesos. En la parte superior, se agrupan trazos con pendiente creciente, en la de abajo decreciente.

espaciales, podemos ver cómo las capas convolucionales asignan representaciones similares a trazos similares. Se observa que los puntos correspondientes a las distintas clases están mezclados. La red separa las activaciones en función del grosor, la longitud y la inclinación del trazo.

3.3. Clusterizado supervisado

Queremos extender la idea de clusterizar representaciones similares vista en las cápsulas [Hinton et al., 2018] para hacer posible su uso en redes de cualquier tipo. Como punto de partida, implementamos y analizamos el algoritmo de clusterizado *centerloss* propuesto por [Wen et al., 2016].

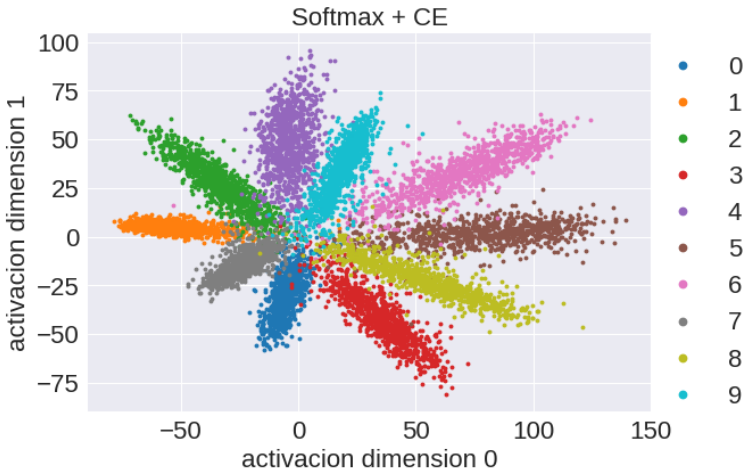


Figura 3.6: Distribución de activaciones aprendida en la penúltima capa de la red cuando utilizamos activación *softmax* y coste CE. Datos tomados del *test set*. Observamos que las clases se agrupan angularmente alrededor del origen.

Estos, junto con [Liu et al., 2016], explican que el coste de CE combinado con la función *softmax* causan que las redes converjan a una solución de clasificación por ángulo o distancia

coseno. Esto queda ilustrado en la Figura 3.6. Argumentan que esto no es óptimo desde el punto de vista de la compactación de representaciones intraclase y separación interclase. Además, sostienen que minimizar una función objetivo más complicada puede ayudar a reducir el sobreajuste en redes con *softmax* + CE.

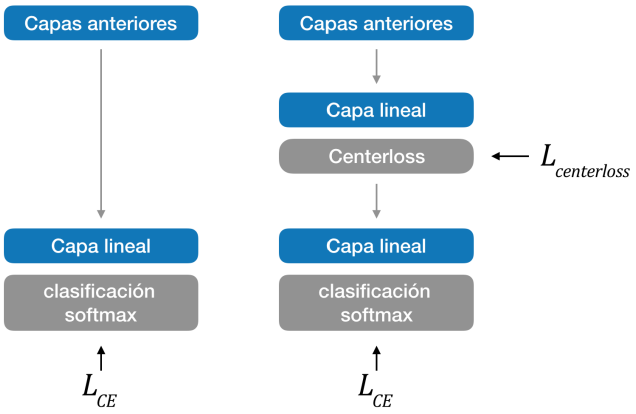


Figura 3.7: A la izquierda, estructura de la última capa de una red clasificadora. A la derecha, estructura una de red clasificadora con *centerloss*.

Centerloss añade una capa lineal, sin función de activación, antes de la última capa de la red, Figura 3.7. En este nuevo espacio d dimensional, se define un centroide por cada clase. En cada iteración del entrenamiento de la red, la posición de cada centroide se mueve hacia el promedio de los puntos de su clase con un paso regulado por el parámetro α. Definimos y_i como la clase a la que pertenece el dato i, $c_{y_i} \in \mathbb{R}^d$ como su centroide correspondiente y N como el número de datos en un *minibatch*. La actualización del centroide correspondiente a la clase j en la iteración t, viene dada por (3.1), refiriéndose $\mathbb{1}$ a la función indicadora. Esta vale 1 si la condición de su argumento se cumple y cero en caso contrario. Se añade una señal de supervisión adicional a la red a través de un coste dado por la distancia euclídea entre cada punto de una clase y su centroide (3.2). De esta forma, el coste final queda como la suma de la entropía cruzada L_{CE} y el coste *centerloss* L_c (3.3).

$$c_j^{t+1} = c_j^t - \alpha \frac{\sum_{i=1}^{N} \mathbb{1}(y_i = j) \cdot (c_j - x_i)}{1 + \sum_{i=1}^{N} \mathbb{1}(y_i = j)} \tag{3.1}$$

$$L_c = \frac{\lambda}{2N} \sum_{i=1}^{N} \|x_i - c_{y_i}\|_2^2 \tag{3.2}$$

$$L = L_{CE} + L_c \tag{3.3}$$

Motivados por este método y siguiendo la filosofía de clasificación por margen propuesta por [Weinberger et al., 2006] y empleada en *Support Vector Machines* (SVM) y en [Sabour et al., 2017], proponemos nuestro propio algoritmo, al que llamamos *margin-centerloss*. La diferencia respecto a *centerloss* está en el cálculo del coste, el cual definimos en (3.4), siendo $D(a, b) = \frac{1}{2}\|a - b\|_2^2$. Penalizamos los puntos clasificados incorrectamente y los puntos clasificados correctamente que superan un margen de distancia a los centroides, m. Escalamos m por el número de dimensiones del espacio d. La función

objetivo de la red será (3.5).

$$L_{mc} = \frac{1}{N}\sum_{i=1}^{N} \text{máx}(D(x_i, c_{y_i}) + m \cdot d - \min_{j \neq y_i} D(x_i, c_j)\ , 0) \tag{3.4}$$

$$L = L_{CE} + L_{mc} \tag{3.5}$$

En contraste con *centerloss*, el algoritmo propuesto no solo minimiza la distancia intraclase, sino que también maximiza la interclase. En situaciones en las que los centroides de las clases se encuentren próximos entre sí, *centerloss* daría un coste bajo, a pesar de que podría existir solape entre datos de distintas clases. *Margin-centerloss* no tiene este problema. En nuestros experimentos, observamos que con *Margin-centerloss* el entrenamiento es más estable y es menos común la convergencia a soluciones subóptimas que con *centerloss*.

Implementamos ambos algoritmos en forma matricial para aprovechar la aceleración CUDA. En el Apéndice C, mostramos una comparativa entre tiempo de cómputo de la implementación original de *centerloss* y la nuestra. En la Figura 3.8 podemos apreciar que las agrupaciones formadas por *Margin-centerloss* tienen una densidad más uniforme y son más compactas que las formadas por *centerloss*. Además, al comparar las distribuciones de datos obtenidas con las de la Figura 3.6, vemos que el esquema de clasificación por ángulo desaparece.

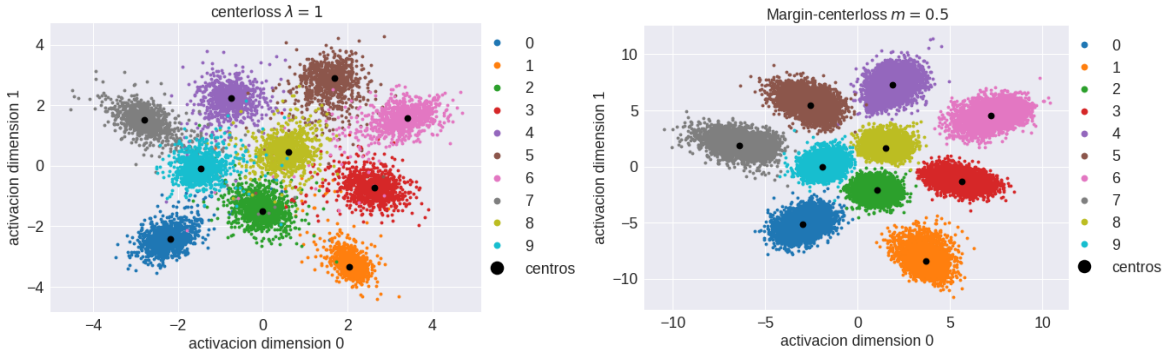


Figura 3.8: A la izquierda, activaciones en la penúltima capa con coste *centerloss*. A la derecha, el mismo experimento con coste *margin-centerloss*. Observamos que el coste con margen comprime menos los grupos pero permite menos puntos fuera del cluster. En ambos casos, los datos son tomados del *test set* de MNIST.

Concurrentemente con el desarrollo de este trabajo, el uso de los algoritmos de clusterizado para mejorar las prestaciones de la función de activación *softmax* ha ganado mucha popularidad. Durante el invierno y primavera de 2018, [Wan et al., 2018], [Zheng et al., 2018] y [He et al., 2018] realizaron publicaciones en esta línea. El método propuesto en la última es similar al nuestro. La principal diferencia es, que mientras que ellos calculan el coste para grupos de *triplets* [Schroff et al., 2015], nosotros lo hacemos para todo el *minibatch*. Anteriormente, [Variani et al., 2015] habían propuesto un modelo en el que la última capa se corresponde con una mezcla de Gaussianas, parametrizada por las salidas de la capa anterior. Por otro lado, [Rippel et al., 2015] habían propuesto un modelo de clusterización con estimación local de densidad.

Las publicaciones mencionadas anteriormente se centran en tareas muy específicas en las que sus métodos de clusterizado de última capa parecen dar buenos resultados en términos de error de clasificación. Decidimos no perseguir esta línea debido a que vemos poca mejora

al probar estos métodos con tareas distintas. Buscamos algoritmos de regularización de aplicabilidad general.

3.4. Medidas de incertidumbre con clusterizado supervisado

La obtención de medidas certeras de incertidumbre en los sistemas de ML es un campo relativamente joven y poco explorado pero que actualmente genera mucho interés. En su tesis, [Gal, 2016] utiliza un marco Bayesiano para conseguir medidas robustas de confianza con sus modelos. En su publicación, [Wan et al., 2018] demuestran que su función de clasificación, basada en mezclas de Gaussianas, consigue cierta resistencia ante ataques de *ejemplos adversarios*.

Una aplicación interesante de los algoritmos de clusterizado es la reducción del efecto de exceso de confianza en las predicciones, muy habitual en clasificadores *softmax* con coste CE. Es común que este tipo de sistemas asignen probabilidades cercanas al 100 % en casos en los que clasifican erróneamente. Esto se debe a la naturaleza exponencial de la función *softmax*. Esta magnifica diferencias pequeñas entre las activaciones de distintas clases y tiende a saturar en la dimensión de la clase ganadora. El exceso de confianza supone un problema importante para el uso de sistemas de ML en escenarios reales. Intuitivamente, podemos utilizar la distancia de un punto a su centroide más cercano como una medida de confianza de que el punto pertenece a esa clase. La distribución de datos obtenida con *softmax* + CE, vista en la Figura 3.6, no permite obtener el mismo tipo de medida de forma sencilla.

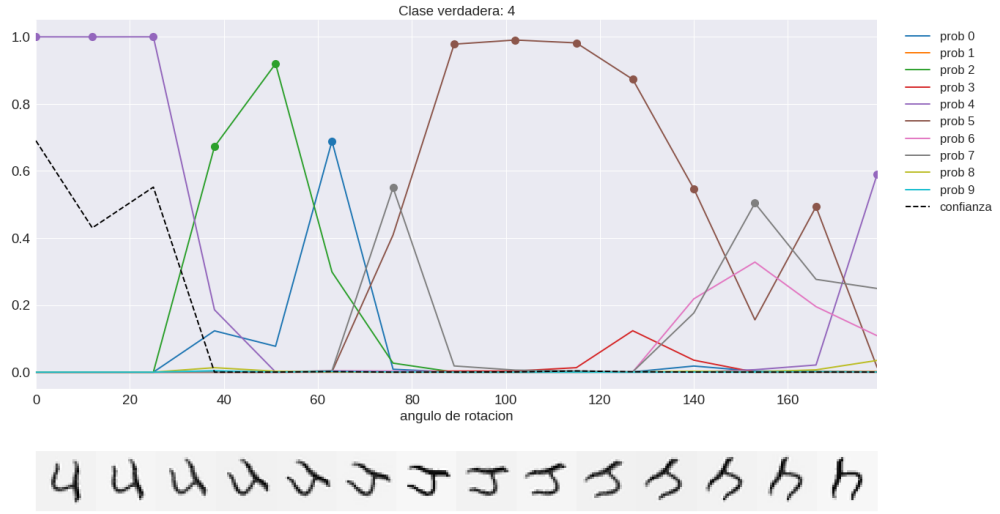


Figura 3.9: Probabilidades asignadas por nuestra red a cada una de las clases al aplicar una rotación progresiva a un ejemplo del *test set*. También se muestra, con la linea discontinua, el valor dado por nuestra medida de confianza. Observamos que para rotaciones superiores a 50°, la red se equivoca pero la probabilidad de la clase ganadora, marcada con un circulo, sigue siendo alta.

Proponemos el uso de nuestro algoritmo, *margin-centerloss*, para obtener medidas de confianza en la clasificación. Validamos su utilidad en un entorno en el que generamos ejemplos distintos a los del dataset original rotando dígitos MNIST. El objetivo es mostrar que nuestro sistema no solo es capaz de establecer un umbral de separación entre las clases, sino que la distribución clusterizada permite distinguir ejemplos similares a los de entrenamiento de datos completamente nuevos.

Dado un dato de validación asignado a la clase j y siendo c_j su correspondiente centroide, proponemos la siguiente medida de confianza. Calculamos el número de ejemplos de entrenamiento etiquetados con j que se encuentran más cerca del centroide que nuestro dato de validación divididos entre el número total de datos de entrenamiento correspondientes a la clase j. Así, evaluamos cómo de atípico es el nuevo dato en el contexto de los datos con los que ha sido entrenado el modelo. Podemos interpretar esta medida como una probabilidad empírica de que la decisión tomada por la red sea correcta. En un sistema real, podríamos elegir una probabilidad a partir de la cual rechazaríamos el dato y consultaríamos la opinión de un experto.

$$\mathcal{C}(x) = \frac{\sum_{i=1}^{N^{tr}} \mathbb{1}(y_i = j)\mathbb{1}(\|x_i^{tr} - c_j\|_2^2 > \|x - c_j\|_2^2)}{\sum_{i=1}^{N^{tr}} \mathbb{1}(y_i = j)} = \frac{\sum_{i=1}^{N_j^{tr}} \mathbb{1}(\|x_i^{j\_tr} - c_j\|_2^2 > \|x - c_j\|_2^2)}{N_j^{tr}} \tag{3.6}$$

Tomamos $j = \arg\min_j(\|x - c_j\|_2^2)$ como el índice del centroide de la clase asignada a nuestro dato x y x^{j_tr} como los datos del *train set* etiquetados con j. Definimos nuestra medida de confianza $\mathcal{C}$ en (3.6).

La Figura 3.9 muestra el cálculo de esta medida sobre las rotaciones de un dígito de valor 4. Los resultados obtenidos con todo el conjunto de validación se encuentran representados en la Figura 3.10.

Apreciamos que la confianza se correlaciona fuertemente con la probabilidad de la clase correcta. Además, para casos atípicos (rotaciones grandes), mientras que la desviación típica de las probabilidades de clase se hace grande, la de la confianza disminuye. Se trata de un método robusto. En este caso, un buen valor para un umbral de rechazo podría ser $\gamma = 0.1$.

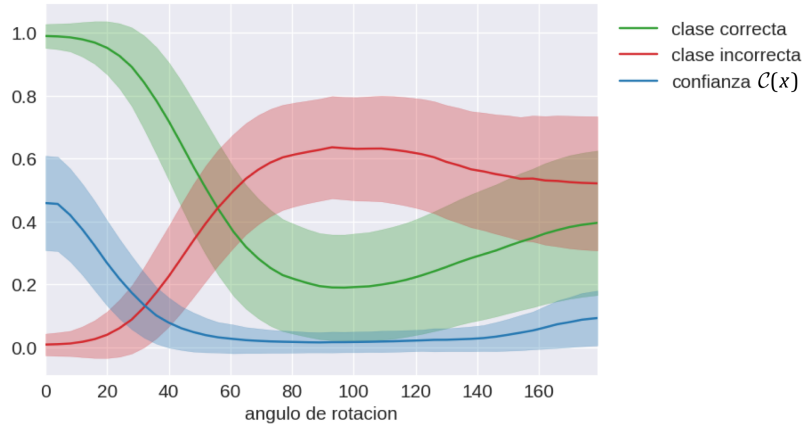


Figura 3.10: Valores medios y desviaciones típicas (en transparente) de las probabilidades de la clase correcta y clase incorrecta de máxima probabilidad. Medidas tomadas con todos los valores del *test set* de MNIST para cada ángulo de rotación. También se muestran la media y desviación típica del parámetro de incertidumbre calculado. Cabe destacar que la probabilidad correcta y la confianza aumentan para rotaciones cercanas a 180 debido a que los dígitos 1 y 8 son, por lo general, invariantes ante esta transformación. El 0 lo es ante todas las rotaciones.

La ventaja de esta medida respecto a otras propuestas de la literatura es que no hacemos suposiciones a priori sobre la distribución de los puntos en los clusters. Al probar a ajustar mezclas de Gausianas a los clusters, obtenemos valores bajos de verosimilitud. Los puntos que

forman los clusters no se distribuyen de forma normal. Por consiguiente, obtenemos medidas de incertidumbre pobres con este tipo de métodos.

Obtenemos excelentes resultados utilizando *margin-centerloss* en combinación con el método propuesto para calcular medidas de incertidumbre. Esto, combinado con la facilidad de despliegue de este método en cualquier red de clasificación, nos hace ver este campo como una atractiva vía de trabajo futuro.

3.5. Clusterizado no supervisado

En una CNN, las representaciones extraídas en las capas convolucionales se unen en la capa lineal para dar una puntuación a cada clase. Los algoritmos de clusterizado supervisado, al depender los centroides de las clases, no pueden ser utilizados para la agrupación de información en capas intermedias. Estas capas no aprenden a clasificar sino a obtener representaciones útiles. Como vemos en la Figura 3.5 los patrones detectados se distribuyen independientemente de la clase. Por consiguiente, la aplicación de *margin-centerloss* está limitada a la última capa de la red.

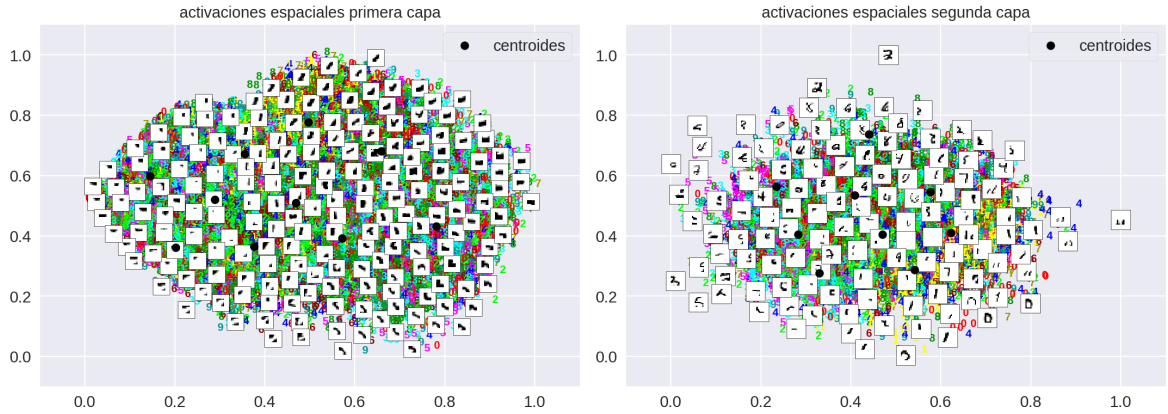


Figura 3.11: Campos receptivos para las activaciones espaciales de los datos del *test set* en la primera y segunda capa de la red utilizando clusterizado no supervisado. Cabe destacar que, debido a las operaciones de diezmado entre capas, los campos receptivos en la segunda capa abarcan una porción mayor de la imagen de entrada. Gráfica obtenida con la técnica de visualización explicada en la Sección 3.2.

Motivados por el enrutamiento EM de [Hinton et al., 2018], buscamos algoritmos no supervisados para agrupar las activaciones espaciales en las capas convolucionales de la red. Esperamos poder agrupar las características que buscan los filtros de estas capas. Con esto, deberíamos poder evitar la propagación de información sobre pequeños detalles en la imagen de entrada que permitan a la red sobreajustarse a los datos de entrenamiento. Así, esperamos que las capas convolucionales aprendan a buscar solo información relevante, útil para clasificar ejemplos nunca antes vistos.

En ese mismo sentido, aprender a segmentar las representaciones intermedias de una red de forma no supervisada puede ser útil de cara a realizar *transfer learning* [Yosinski et al., 2014]. Un ejemplo de esto sería aprender a realizar una tarea de clasificación de imagen en la que tuviéramos pocos ejemplos. Podríamos entrenar nuestro modelo con otra base de datos más grande para que las capas convolucionales aprendieran a detectar características útiles y luego entrenar el clasificador lineal de la última capa con la base de datos pequeña.

El único uso anterior de clusterizado no supervisado en capas intermedias que conocemos es el de [Aytar et al., 2016]. Estos utilizan mezclas de Gausianas con EM para establecer distribuciones a priori de activaciones y hacer aprendizaje invariante al dominio (*cross domain learning*).

Como primer experimento, modificamos el algoritmo *centerloss* para que no utilice información de clase. El coste dependerá de la distancia de cada punto x_i a su centroide más cercano $c_{n_i^t}$. Lo definimos en (3.7), refiriéndose $n_i^t = \arg\min_j \left(\|x_i - c_j^{t-1}\|_2^2 \right)$, a la clase del dato i en la iteración t. A su vez, la regla de actualización de los centroides moverá a cada uno de ellos hacia la media de los puntos que le hayan sido asignados (3.8). Este método es similar al algoritmo *k-means*, una particularización del algoritmo EM.

$$L_c = \frac{\lambda}{2N} \sum_{i=1}^{N} \|x_i - c_{n_i^t}\|_2^2 \tag{3.7}$$

$$c_j^{t+1} = c_j^t - \alpha \frac{\sum_{i=1}^{N} \mathbb{1}(n_i^t = j) \cdot (c_j - x_i)}{1 + \sum_{i=1}^{N} \mathbb{1}(n_i^t = j)} \tag{3.8}$$

En la Figura 3.11, representamos las activaciones espaciales de la primera y segunda capa de nuestra red utilizando nuestra versión de *centerloss* no supervisada. Las activaciones espaciales con campos receptivos similares siguen organizándose de forma continua. No conseguimos separarlas en clusters. Al aumentar λ para dar más importancia al coste de clusterización, en lugar de formarse estructura en las activaciones, estas tienden a 0. Esto se debe a que, cuando todos los puntos y centroides colapsan al origen, el coste de clusterización desaparece.

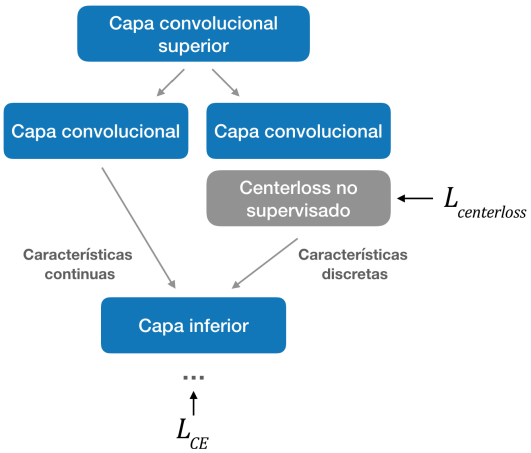


Figura 3.12: Diagrama de una CNN con dos caminos. En el de la derecha, no aplicamos coste adicional. En el de la izquierda, aplicamos el algoritmo de clusterizado no supervisado propuesto a las activaciones espaciales. Aunque ambos caminos tienen los mismos datos de entrada, esperamos que cada uno de ellos aprenda a identificar patrones distintos.

Realizamos un segundo experimento en el que definimos dos caminos paralelos en la red. En uno utilizamos clusterizado no supervisado y en el otro no añadimos coste adicional. Un diagrama de este montaje se muestra en Figura 3.12. Con esto, esperamos separar la información discretizable de la continua en las activaciones espaciales de la red. En la Figura 3.13, mostramos las activaciones en cada uno de los caminos. El grupo de datos que se propagan por el camino con *centerloss* no supervisado tiene menor variedad de trazos

en sus campos receptivos que el camino sin coste adicional. En lugar de discretizar las representaciones, la red aprende a enviar más información por la vía sin coste adicional ya que esto le permite mayor flexibilidad. La atracción a los centros acaba sencillamente empujando todas las activaciones hacia un punto central, habitualmente el origen.

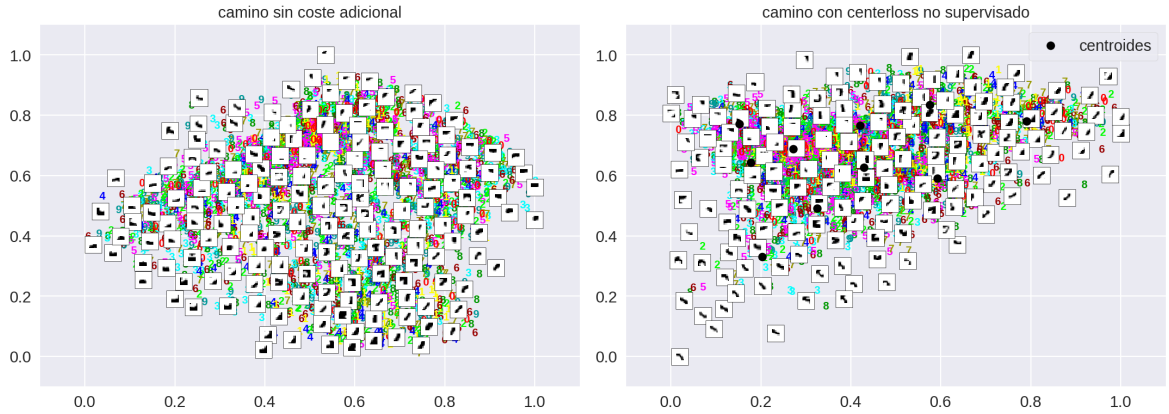


Figura 3.13: Activaciones espaciales junto a sus campos receptivos del *test set* tras la primera capa de nuestro modelo de dos caminos. solo aplicamos el coste de clusterizado no supervisado al camino representado a la derecha.

Hemos visto que es sencillo clusterizar las activaciones de las últimas capas de la red gracias a que la señal de entrenamiento supervisada asegura una segmentación en función de la clase. Sin embargo, encontramos que la información en las capas anteriores de la red, donde se generan las representaciones que nos interesan, forma espacios poco estructurados y difíciles de segmentar. En lugar de buscar estructura en un modelo de red neuronal tradicional, nos planteamos el uso de un modelo Bayesiano de variables latentes. Esto nos permite más flexibilidad a la hora de agrupar la información a través de la elección de distribuciones de probabilidad a priori para nuestro espacio latente.

Capítulo 4

Modelos probabilísticos Bayesianos y representaciones internas

En este capítulo, adoptamos un marco probabilístico Bayesiano que nos permite obtener representaciones de nuestros datos estructuradas e interpretables. Primero, exponemos el concepto de variables latentes y la teoría relevante del campo de la inferencia variacional. Hacemos especial énfasis en los modelos de variables latentes parametrizados por redes neuronales profundas y en las técnicas de desentrelazado.

Apoyándonos en esta teoría, proponemos un nuevo modelo de variables latentes. Este permite modelar distribuciones de datos más complejas que los modelos actuales. Como consecuencia, nos permite obtener representaciones internas más desentrelazadas y un modelo generativo más expresivo. Es la principal contribución de este trabajo.

A continuación, en la sección de validación experimental, presentamos una forma novedosa de evaluar modelos generativos que extiende el método propuesto por [Odena et al., 2017]. Las ventajas de nuestras propuestas teóricas se ven reflejadas en los resultados experimentales.

4.1. Modelos de variables latentes

En estadística, las variables latentes u ocultas son aquellas que no pueden ser observadas directamente pero pueden ser inferidas a partir de las variables de las que sí disponemos (observadas). Un modelo de variables latentes es aquel que intenta expresar la información contenida en los datos observados en términos de variables latentes.

Planteamos un ejemplo sencillo. Supongamos que disponemos de una base de datos de imágenes de caras de personas. Las variables observadas de las que disponemos son los píxeles. Existen infinidad de características sobre la imagen que no se pueden extraer directamente de las amplitudes de los píxeles. Ejemplos de estas pueden ser si la persona de la foto esta sonriendo, el color de sus ojos o la dirección en la que está mirando.

Nos centramos en la sonrisa del individuo fotografiado. A grandes rasgos, sabemos que si los extremos de la boca del sujeto forman una curva hacia arriba, esta sonriendo. Una curvatura hacia abajo indica seriedad. Existen innumerables expresiones faciales para expresar desde sentimientos muy negativos, hasta muy positivos. Así, podríamos modelar la emoción expresada en la foto como una variable latente continua. Por otro lado, el género del sujeto podría ser modelado como una variable latente discreta binaria. El color de los ojos podría ser una variable continua, con una distribución fuertemente trimodal; habiendo un modo por cada color posible.

A menudo, los datos de entrada a los modelos de ML presentan distribuciones muy complejas. Como ejemplo, podemos volver a considerar la distribución de un conjunto de datos compuesto por fotos de caras. Hay una serie de relaciones que se deben cumplir entre las amplitudes de los píxeles de la imagen para que esta muestre una cara. Se trata de una distribución muy difícil de imaginar ya que tiene tantas dimensiones como píxeles tiene la imagen y su forma probablemente sea complicada. Sin embargo, sí que podemos imaginarnos todos los factores de variación que presentan las caras humanas. Nuestro objetivo es diseñar un modelo capaz de expresar la información contenida en nuestros datos de entrada en función de los factores de variación subyacentes que la rigen. Las variables latentes modelan estos factores de variación. Además, intentaremos que estas presenten distribuciones manejables y tengan un significado interpretable.

4.2. Inferencia bayesiana variacional

Dados nuestros datos de entrada $x{=}x_{1:N}$ y un conjunto de variables latentes $z{=}z_{1:d}$, queremos obtener la distribución condicional de las variables latentes $p(z|x)$. Podemos descomponer $p(z|x)$ utilizando el teorema de Bayes (4.1). Desafortunadamente, no podemos utilizar este método para la obtención de la probabilidad condicional o a posteriori. Esto se debe a que el cálculo de la probabilidad marginal, $p(x) = \int p(x,z)\,dz$, es un problema intratable para los modelos de redes neuronales profundas en los que estamos interesados. Por tanto, recurriremos a técnicas de inferencia aproximada.

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x,z)}{p(x)} = \frac{p(x,z)}{\int p(x,z)\,dz} \tag{4.1}$$

Comenzamos definiendo la divergencia Kullback–Leibler en (4.2). Se trata de una medida de lo diferentes que son dos distribuciones, p y q. Siempre es mayor o igual a 0. En inferencia variacional, p representa una distribución concreta y q representa una aproximación a p.

$$D_{KL}(q(z) \,\|\, p(z)) \triangleq \int q(z) \log(\frac{q(z)}{p(z)})\,dz \tag{4.2}$$

La inferencia variacional define una familia de funciones, ℓ, de forma que cada $q(z|x) \in \ell$ es un candidato para aproximar la distribución de z. Queremos encontrar q^* tal que $q^*(z|x) = \arg\min_{q\in\ell} D_{KL}(q(z|x) \,\|\, p(z|x))$. Introducimos un modelo de inferencia paramétrico $q(z|x)$. Sus parámetros serán los parámetros variacionales, los cuales intentaremos optimizar de forma que $q(z|x) \approx p(z|x)$.

Para poder minimizar $D_{KL}(q(z|x) \,\|\, p(z|x))$, comenzamos expandiendo la expresión (4.3). Utilizando el teorema de Bayes (4.1) llegamos a (4.4). Cabe destacar que, podemos sacar $\log p(x)$ de la integral ya que $\mathbb{E}_{q(z|x)}[f(x)] = f(x)$.

$$\begin{aligned} D_{KL}(q(z|x) \,\|\, p(z|x)) &= \int q(z|x) \log\left(\frac{q(z|x)}{p(z|x)}\right) dz \\ &= \int q(z|x)\big[\log q(z|x) - \log p(z|x)\big]\,dz \end{aligned} \tag{4.3}$$

$$\begin{aligned} D_{KL}(q(z|x) \,\|\, p(z|x)) &= \int q(z|x)\big[\log q(z|x) - \log p(x,z) + \log p(x)\big]\,dz \\ &= D_{KL}(q(z|x) \,\|\, p(x,z)) + \log p(x) \end{aligned} \tag{4.4}$$

A continuación, definimos el límite inferior variacional o ELBO, (*evidence lower bound*) $\mathcal{L}$, en (4.5). Ahora podemos escribir $\log(p(x))=D_{KL}(q(z|x) \,\|\, p(z|x))+\mathcal{L}(x)$. Como la divergencia Kullback–Leibler es siempre mayor que 0, sabemos que $\log(p(x)) \leq \mathcal{L}(x)$. El ELBO es un un límite inferior a la log-verosimilitud de nuestros datos. La distancia entre el ELBO y la distribución marginal se hace cero cuando $q(z|x)=p(z|x)$. Hemos convertido nuestro problema original en uno de maximizar $\mathcal{L}(x)$. Como último paso, descomponemos el ELBO en dos componentes que nos serán útiles a la hora de crear nuestro modelo (4.6).

$$\mathcal{L}(x) \triangleq -D_{KL}(q(z|x) \,\|\, p(x,z)) = \log p(x) - D_{KL}(q(z|x) \,\|\, p(z|x)) \tag{4.5}$$

$$\begin{aligned}
\log p(x) \geq \mathcal{L}(x) &= -D_{KL}(q(z|x) \,\|\, p(x,z)) \\
&= -\int q(z|x) \log \left[\frac{q(z|x)}{p(x|z)p(z)} \right] dz \\
&= -\int q(z|x) [\log \left[\frac{q(z|x)}{p(z)} \right] - \log p(x|z)] \, dz \\
&= -D_{KL}(q(z|x) \,\|\, p(z)) + \mathop{\mathbb{E}}_{q(z|x)} [\log p(x|z)]
\end{aligned} \tag{4.6}$$

4.3. El autoencoder variacional

El autoencoder variacional (VAE), propuesto por [Kingma and Welling, 2013], es un modelo que intenta maximizar el ELBO descompuesto como en (4.6). La distribución condicional de z aproximada, $q_\phi(z|x)$, queda parametrizada por una red neuronal profunda, que hace de *encoder*. Designamos sus parámetros con la letra ϕ. Parametrizamos $p_\theta(x|z)$ con otra red neuronal profunda, el *decoder*, con parámetros θ.

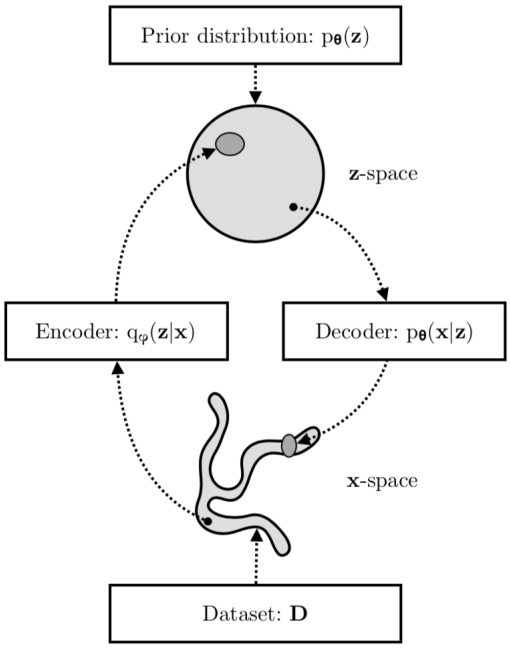


Figura 4.1: El VAE aprende una transformación desde el espacio de los datos x, cuya distribución es habitualmente compleja, hasta el espacio variables latentes z. A través de la elección de la distribución a priori $p_\theta(z)$, podemos regular la complejidad del espacio de variables latentes. Figura tomada de [Kingma, 2017].

En un VAE, el *encoder* define una transformación desde el espacio de los datos de entrada hasta el de las variables latentes. El *decoder* es un modelo generativo que toma como entrada las variables latentes $z_{1:d}$ y las utiliza para reconstruir la imagen original. Ilustramos este concepto en la Figura 4.1.

Utilizando la expresión (4.6) para maximizar $\mathcal{L}_{\phi\theta}(x)$ respecto de ϕ y θ, conseguimos dos cosas. Aproximamos la distribución marginal $p(x)$, haciendo más preciso nuestro modelo generativo o *decoder*. Minimizamos $D_{KL}(q(z|x) \,\|\, p(z|x))$, haciendo que nuestro *encoder* aproxime mejor la distribución a posteriori de z.

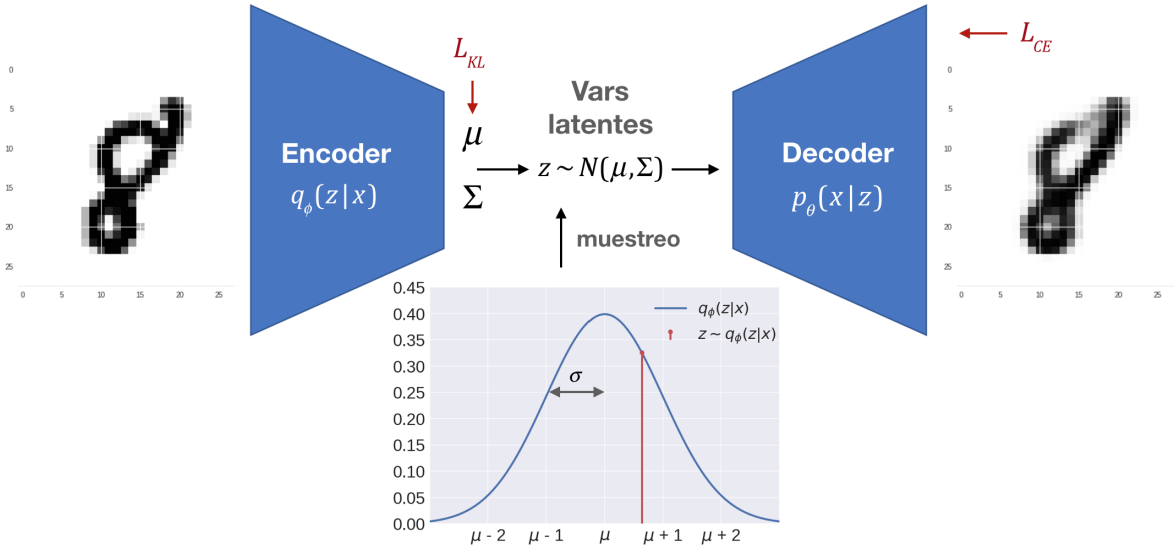


Figura 4.2: Diagrama de funcionamiento del VAE para el caso de procesado de imagen. El *encoder* tiene como salida los parámetros de una distribución normal, μ y Σ. Tomamos una muestra aleatoria de esta distribución z y se la pasamos al *decoder*. Cada dimensión de z codifica información distinta sobre la imagen original. El *decoder* deberá utilizar esta información para reconstruir la imagen de entrada. Con L_{KL}, penalizamos al encoder por devolver una distribución para las variables latentes que se aleja de la normal isotrópica. Con L_{CE}, penalizamos el error de reconstrucción.

Como vemos en (4.6), el primer término de $\mathcal{L}_{\phi\theta}(x)$ obliga a $q_\phi(z|x)$ a parecerse a una distribución a priori $p_\theta(z)$. Para esta distribución a priori, elegimos variables latentes continuas con una distribución normal multivariada de media 0 y matriz de covarianza identidad $N(0, I)$. Las ventajas de esta elección son su sencillez y que $D_{KL}(q(z|x) \,\|\, p(z))$ se puede obtener analíticamente. El segundo término de (4.6) es sencillamente el error de reconstrucción del dato generado a partir de las variables latentes respecto del original.

Para una imagen de entrada concreta $x^{(i)}$, el *encoder* nos dará los parámetros de la distribución $q(z|x^{(i)})$: $\mu^{(i)}$ y $\sigma^{(i)} \cdot I$. Tomaremos una muestra aleatoria de esta distribución, $z^{(i)}$, la cual pasaremos como entrada al *decoder*. A partir de ella, este regenerará la imagen original. Esquematizamos esto en Figura 4.2. Conociendo $\mu^{(i)}$ y $\sigma^{(i)}$, calculamos el primer término del ELBO en (4.7). Con la imagen regenerada, podemos estimar el segundo término tomando L muestras de $q(z|x^{(i)})$ (4.8). Por simplicidad fijaremos $L = 1$. Juntando los términos, obtenemos $\mathcal{L}^B$, un estimador insesgado de $\mathcal{L}$ (4.9). Esta es la función objetivo que maximizamos con el VAE.

$$-D_{KL}(q(z|x^{(i)}) \,\|\, p(z)) = \frac{1}{2}\sum_{j=1}^{d}(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) \tag{4.7}$$

$$\mathop{\mathbb{E}}_{q(z^{(i)}|x^{(i)})}[\log p(x^{(i)}|z^{(i)})] \approx \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(x_l^{(i)}) \tag{4.8}$$

$$\mathcal{L}_{\phi\,\theta}(x) \approx \mathcal{L}^B_{\phi\,\theta}(x) = \frac{1}{2}\sum_{j=1}^{d}(1 + \log((\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2) + \frac{1}{L}\sum_{l=1}^{L}\log p_\theta(x_l^{(i)}) \tag{4.9}$$

Intuitivamente, la aleatoriedad del muestreo de $z^{(i)}$ obliga al *encoder* a agrupar datos similares en el espacio de z. Podemos interpretar que $z^{(i)}$ es igual a la suma de $\mu^{(i)}$ con un término aleatorio de ruido. Si puntos cercanos de $q(z)$ representan información similar, el ruido afectará menos a la reconstrucción de la imagen original. Así, obtenemos un espacio de variables latentes estructurado.

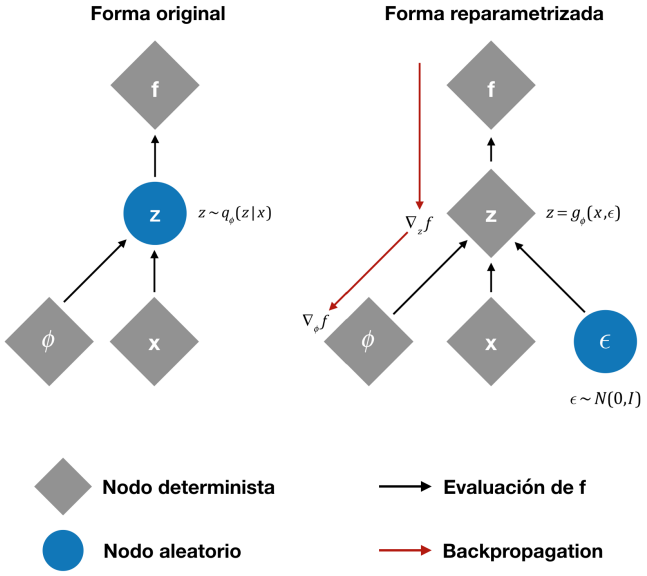


Figura 4.3: A la izquierda, el planteamiento original del espacio latente. El nodo estocástico representa una operación de muestreo aleatorio no diferenciable. A la derecha, el modelo reparametrizado. Al convertir el muestreo en un producto con una variable aleatoria, construimos un camino diferenciable. Basado en una figura de [Kingma, 2017].

$$z = g_\phi(x^{(i)}, \epsilon) = \mu^{(i)} + \sigma^{(i)} \odot \epsilon \tag{4.10}$$

$$\epsilon \sim N(0, I) \tag{4.11}$$

Buscamos optimizar los parámetros de nuestras redes con *backpropagation*. A priori, esto supone un problema ya que el muestreo aleatorio de una distribución no es una operación diferenciable. [Kingma and Welling, 2013] y [Rezende et al., 2014] proponen "el truco de la reparametrización" (*reparametrization trick*) en el que reescriben el muestreo de $z \sim N(\mu, \sigma^2 I)$ de la forma (4.10), refiriéndose $\odot$ al producto elemento a elemento y siendo ϵ una muestra de ruido Gaussiano de media nula y varianza unidad (4.11). Ilustramos este concepto en Figura 4.3.

Implementamos el VAE en Python con la librería Pytorch. Los detalles de su arquitectura se pueden encontrar en el Apéndice B. Lo entrenamos con la base de datos MNIST. Como error de reconstrucción, $-\log p(x^{(i)}|z^{(i)})$, utilizamos la entropía cruzada entre cada par de píxeles de la imagen original y la reconstruida. Designamos a este coste L_{CE}. Nos referiremos al término $D_{KL}(q(z|x^{(i)}) \,\|\, p(z))$ como L_{KL}.

En la Figura 4.4 generamos reconstrucciones para un conjunto de puntos equiespaciados en el espacio latente. La *variedad* o *manifold* es la distribución del conjunto de factores con los que podemos describir los datos completamente. Es un subespacio con menos grados de libertad que los datos originales. Explorando z a través de las reconstrucciones, podemos ver cualitativamente cómo de bien hemos modelado el *manifold* correspondiente a nuestros datos.

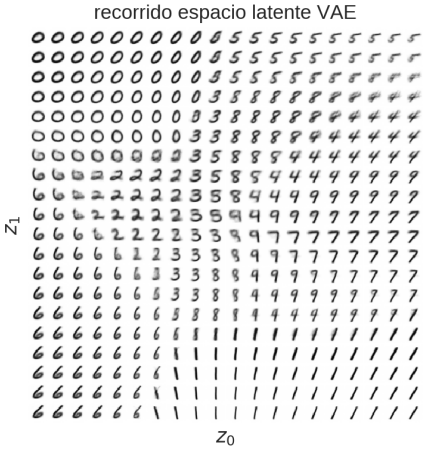


Figura 4.4: Reconstrucciones para valores de z equiespaciados en el rango $[-2, 2]$. Debido a nuestra elección de distribución a priori $N(0, I)$, sabemos que los datos se agruparán en torno al origen con varianza 1. Por comodidad, representamos un espacio latente bidimensional. En este caso, el principal factor aprendido es la distinción de clase. También se aprecian variaciones en la inclinación y grosor de los dígitos.

4.4. Distribución a priori, interpretación y desentrelazado

La introducción del VAE por [Kingma and Welling, 2013] supuso un avance importante para el campo de los modelos generativos y del *representation learning*. Desde entonces, ha surgido una subdisciplina que busca desentrelazar las representaciones aprendidas por los VAE. Es decir, busca que cada dimensión de las variables latentes controle un factor de variación distinto en la imagen. Volviendo al ejemplo de las imágenes de caras, con un espacio latente desentrelazado podríamos aumentar o disminuir la sonrisa en la cara reconstruida variando la amplitud de una dimensión de z. En un espacio entrelazado, la sonrisa dependería de combinaciones especificas de distintas dimensiones de z. Así, el entrelazado se puede entender como un acoplo entre factores, de forma que al cambiar una sola dimensión de z, veamos variaciones en múltiples factores de la imagen generada.

Posibles aplicaciones de los VAE desentrelazados son la obtención de representaciones interpretables que nos permitan aprender más sobre nuestros conjuntos de datos y la realización de modificaciones controladas sobre los datos de entrada al variar gradualmente las variables latentes. Ambas posibilidades hacen a los VAE más versátiles que los modelos

generativos adversarios (GAN) [Goodfellow et al., 2014], los cuales también han ganado mucha popularidad recientemente.

El concepto de desentrelazado es difícil de expresar matemáticamente ya que, a veces, lo asociamos a características perceptuales subjetivas. Sin embargo, sabemos que un espacio latente desentrelazado es uno en el que sus dimensiones modelan fenómenos independientes, luego son variables independientes. En esta línea, [Irina Higgins, 2017], [Burgess et al., 2018], [Chen et al., 2018], [Kim and Mnih, 2018], [Gao et al., 2018] y [Esmaeili et al., 2018] analizan el ELBO y proponen modificaciones para obtener un espacio latente más desentrelazado.

Tomamos la entropía de Shannon como $H(x) = -\mathbb{E}[\log p(x)]$ [Shannon, 1948]. A partir de ella definimos la información mutua entre dos variables (4.12). En (4.13), definimos *total correlation* (TC) [Watanabe, 1961]. Se trata de una medida de la dependencia entre las dimensiones de una variable. Es 0 si son independientes.

$$I(x;y) = H(x) - H(x|y) = \mathop{\mathbb{E}}_{y}[D_{KL}(p(x|y) \,\|\, p(x))] \tag{4.12}$$

$$TC(x) = \sum_{j=1}^{d} H(x_j) - H(x) = D_{KL}(p(x) \,\|\, \prod_{j=1}^{d} p(x_j)) \tag{4.13}$$

Para un conjunto de N datos y para un espacio latente de d dimensiones, en (4.14), separamos la divergencia entre nuestra distribución aproximada y distribución a priori, en tres partes [Chen et al., 2018][Esmaeili et al., 2018]. La primera (I), es la información mutua entre los datos x y las variables latentes z. Nos interesa que este término sea lo más grande posible para que nuestras variables latentes modelen nuestros datos fielmente. El segundo término (II), mide la correlación entre las dimensiones de z. Lo queremos minimizar para obtener representaciones desentrelazadas. El último término (III), regula que la distribución marginal de cada dimensión de $q_\phi(z|x)$ no difiera mucho de la distribución a priori $p(z)$. Presentamos la demostración de esta descomposición en el Apéndice D.

$$\frac{1}{N}\sum_{i=1}^{N} D_{KL}(q(z|x^{(i)}) \,\|\, p(z)) \;=\; \underbrace{I(x:z)}_{\text{(I)}} \;+\; \underbrace{TC(z)}_{\text{(II)}} \;+\; \underbrace{\sum_{j=1}^{d} D_{KL}(q(z_j) \,\|\, p(z_j))}_{\text{(III)}} \tag{4.14}$$

Analizamos las técnicas de desentrelazado propuestas anteriormente y decidimos implementar *β-VAE*, propuesto por [Irina Higgins, 2017], *$C-VAE$*, propuesto por [Burgess et al., 2018] y *anchor-VAE*, propuesto por [Gao et al., 2018]. Obviamos los demás métodos debido a que suponen una modificación importante sobre el algoritmo original del VAE y conllevan un coste computacional significativamente mayor.

La función objetivo de *β-VAE*, encontrada en (4.15), multiplica la divergencia entre la distribución aproximada y la distribución a priori por $\beta > 1$. Esto penaliza el término TC de la divergencia pero también penaliza la información mutua entre la entrada y las variables latentes. Una β grande causa que nuestro espacio latente esté desentrelazado, pero también hace peores nuestras reconstrucciones.

El coste propuesto por [Burgess et al., 2018] se encuentra en (4.16). El valor de C crece con el número de iteraciones. De esta manera, en las primeras iteraciones, las dimensiones de z estarán poco correladas pero serán poco informativas. Al crecer C, se fuerza a crecer la información de las variables latentes, a la vez que se espera que no pierdan la estructura

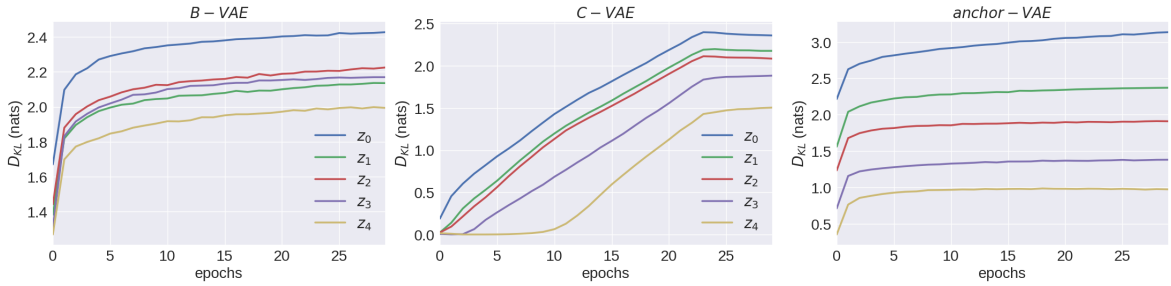


Figura 4.5: Mostramos $D_{KL}(q(z_j|x) \,\|\, p(z_j))$ para cada dimensión del espacio latente z_j, para cada método de desentrelazado expuesto. La medimos en nats ya que representa la cantidad de información codificada en cada dimensión. Tomamos valores β=2, C va de 0 a 10 tras 11000 iteraciones y λ=$[1, 2, 3, 4, 5]$. Medidas realizadas sobre datos de validación.

desentrelazada. γ se fija a un valor alto para asegurar que se cumple la restricción a la D_{KL} impuesta por C. A pesar de no tener nombre propio, nos referiremos a este modelo como "*C-VAE*".

El coste de *anchor-VAE* (4.17), es similar al de *β-VAE*, pero sustituye el factor β por un vector $\overline{\lambda}$. Así, utiliza una penalización distinta para cada dimensión de z. El hecho de que no sean igual de informativas, ayuda a prevenir la coadaptación entre dimensiones, resultando en un espacio latente desentrelazado.

$$L_{\beta-VAE} = L_{CE} + \beta D_{KL}(q_\phi(z|x) \,\|\, p_\theta(z)) \tag{4.15}$$

$$L_{C-VAE} = L_{CE} + \gamma \mid D_{KL}(q_\phi(z|x) \,\|\, p_\theta(z)) - C \mid \tag{4.16}$$

$$L_{anchor-VAE} = L_{CE} + \overline{\lambda} D_{KL}(q_\phi(z|x) \,\|\, p_\theta(z)) \tag{4.17}$$

Mostramos una comparación de la información enviada en cada dimensión del espacio latente con cada una de estas técnicas en la Figura 4.5. En el Apéndice D, incluimos mapas de reconstrucciones para distintos valores de z con cada método.

4.5. Modelo Propuesto

No todos los fenómenos que ocurren en los datos se pueden describir con variables continuas distribuidas normalmente. La separación por clase existe en muchos conjuntos de datos y es un fenómeno discreto. En esta linea, [Dupont, 2018], [Chen et al., 2016] y [Odena et al., 2017] introducen modelos generativos con variables latentes discretas. Además, [Jimenez Rezende and Mohamed, 2015] y [Dilokthanakul et al., 2016] proponen utilizar *normalizing flows* y GMMs respectivamente para conseguir distribuciones a priori $p(z)$ más flexibles.

Cuando hay un factor de variabilidad grande en el conjunto de datos que no se modela bien con la distribución a priori, suele ocupar varias de las dimensiones del espacio latente, entrelazandolo y causando un coste D_{KL} elevado. Si lo mismo ocurre con un factor que tiene poca repercusión en los datos, este suele ser ignorado por el VAE. Motivados por la red de reconstrucción dependiente de clase de las cápsulas que hemos analizado en el Capítulo 2, presentamos una solución a ambos problemas. Introducimos un modelo capaz de capturar factores de variabilidad con distribuciones tanto unimodales como multimodales en el espacio latente.

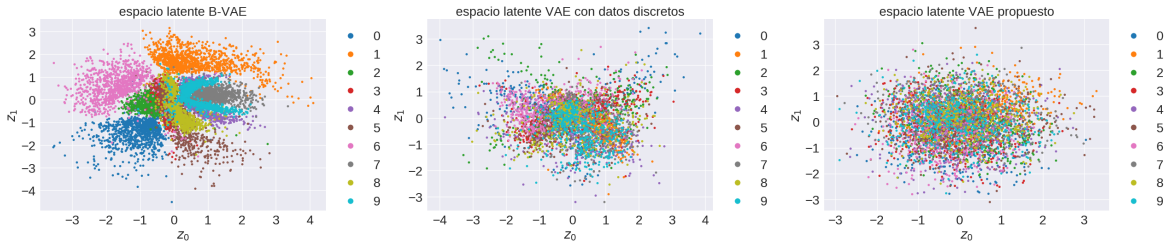


Figura 4.6: Distribuciones de variables latentes bidimensionales para datos del *test set* de MNIST. En el caso de β-VAE, observamos un espacio latente segmentado en función de la clase. Al añadir la variable discreta de clase, se rompe la multimodalidad pero las variaciones intraclase siguen condicionando la distribución de los datos. En el modelo propuesto, la distribución de las variables latentes compartidas es independiente de la clase. Para z con dimensionalidad mayor que 2, se mantiene la estructura del espacio latente representada.

Para el caso de MNIST, el factor de variación más importante, la clase, es discreto. Comenzamos diseñando un VAE con un objetivo adicional: clasificar el dígito de entrada. La clase decidida se codifica en un vector indicador o vector "*one-hot*", un vector binario en el que solo hay un elemento activo. Este, se concatena a las variables latentes para ser enviado al reconstructor. Representamos este modelo en el diagrama central de la Figura 4.7.

En la Figura 4.6 observamos cómo está técnica consigue reducir la segmentación por clase en el espacio latente. Sin embargo, al generar reconstrucciones con muestras aleatorias de z y del vector de clase, encontramos que son muy pobres. Las representamos en la Figura 4.8. Esto se debe a que sigue habiendo dependencia de clase en el espacio latente en forma de variaciones intraclase. Si combinamos un vector *one-hot* de una clase con un valor de z donde hay información de otra, generamos una aberración.

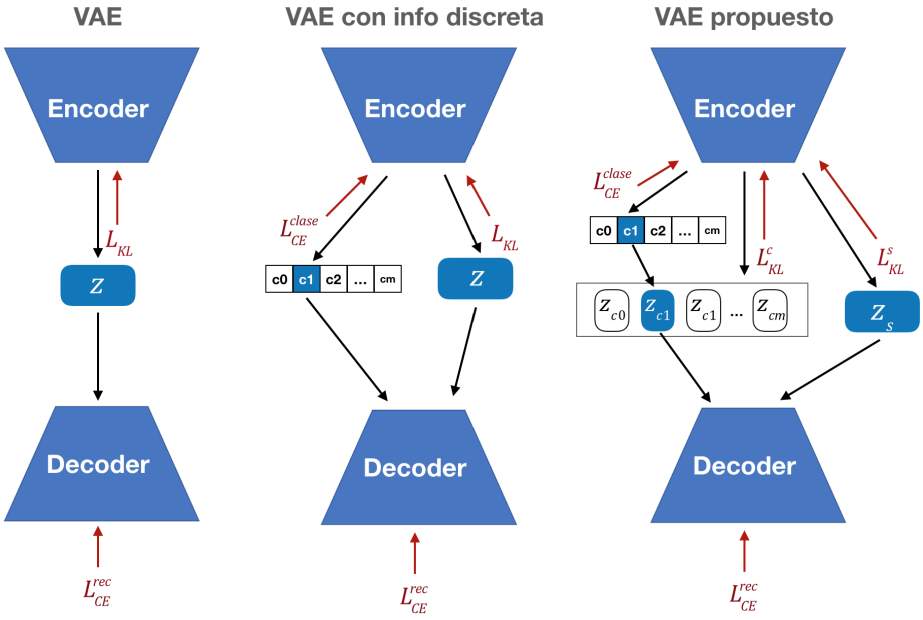


Figura 4.7: Esquema de las variables latentes de un VAE tradicional, de un VAE con información discreta de clase y del modelo propuesto en esta sección. En nuestro modelo, tras clasificar la entrada con la clase c_i, se envían al decodificador las variables latentes compartidas z_s junto a las variables latentes de la clase seleccionada z_{c_i}.

Solucionamos el problema de la dependencia entre variables continuas y discretas

Figura 4.8: Reconstrucciones para muestras aleatorias del espacio latente tomadas de $N(0, I)$. Las variables discretas se muestrean de una distribución multinomial con clases equiprobables. Observamos que la dependencia entre el espacio latente y continuo del segundo modelo causa la reconstrucción de aberraciones. El modelo propuesto tiene la flexibilidad para captar mejor el *manifold* de los datos. Genera reconstrucciones de mayor calidad que los otros dos. La independencia de las variables discretas de las continuas tanto de clase como compartidas hace que cualquier combinación regenere un dígito correctamente.

añadiendo variables latentes continuas dependientes del vector de clase. Estas modelan las variaciones intraclase de los datos. Las variables latentes originales, o compartidas, modelan los factores de variación comunes a todas las clases, como tamaño, grosor de trazo, etc. De forma más general, se trata de un modelo con un espacio latente formado por variables continuas multimodales (un modo por clase), z^c, y variables continuas unimodales (compartidas entre clases), z^s. La arquitectura del modelo propuesto queda esquematizada en la Figura 4.7.

Escribimos la función objetivo de nuestro modelo, $L_{propuesta}$, en (4.18). Respecto al VAE tradicional, añadimos una señal de clasificación a través del coste de entropía cruzada L_{CE}^{class}, y separamos z en un término dependiente de clase z_c y un término compartido z_s. Aplicamos un coste a cada uno por divergir de nuestra distribución a priori normal $p_\theta(z) = N(0, I)$. Designamos a estos costes L_{KL}^c (4.19) y L_{KL}^s (4.20) respectivamente. Mantenemos el objetivo de reconstrucción a través de la entropía cruzada de nuestros datos con la salida del decoder, L_{CE}^{rec}.

$$L_{propuesta} = L_{CE}^{class} + L_{KL}^c + L_{KL}^s + L_{CE}^{rec} \tag{4.18}$$

$$L_{KL}^c = \beta_c D_{KL}(q(z_c|x) \,\|\, p(z)) \tag{4.19}$$

$$L_{KL}^s = \beta_s D_{KL}(q(z_s|x) \,\|\, p(z)) \tag{4.20}$$

Para regular la información transmitida por cada camino del espacio latente, utilizamos el factor β propuesto en [Irina Higgins, 2017]. Probamos los tres métodos de desentrelazado expuestos anteriormente, obteniendo resultados similares con todos. Nos decidimos por este porque es el más sencillo. A diferencia de las demás técnicas de desentrelazado, nuestro modelo consigue su objetivo a través del uso de una distribución a priori más flexible para las variables latentes. No requiere factores complejos en el coste ni multitud de hiperparámetros adicionales. A su vez, no hace falta comprometer la información mutua entre datos y variables latentes para conseguir una representación desentrelazada. Esta flexibilidad permite a nuestro VAE capturar mejor la información de los datos de entrada y por tanto, generar muestras

nuevas de más calidad. Esto se aprecia en la Figura 4.8. En el Apéndice D, mostramos generaciones obtenidas con los demás métodos de desentrelazado.

El modelo propuesto combina la reconstrucción dependiente de clase vista en el Capítulo 2, la separación de la información en dos caminos vista en el Capítulo 3 y el desentrelazado visto en la sección anterior. La clave de su funcionamiento es la dependencia que se crea entre clasificar bien y reconstruir bien. No podemos reparametrizar la clasificación para diferenciar el error de reconstrucción respecto de ella. Sin embargo, sí que podemos diferenciar el error de reconstrucción respecto de las variables latentes de clase. Tras clasificar, aplicamos una máscara a z_c dejando pasar solo las variables correspondientes a la clase elegida. Como consecuencia, el coste de reconstrucción L_{CE}^{rec} refuerza la asociación entre cada variable latente de clase y su respectiva clase. A pesar de que la máscara solo deja pasar algunas de ellas, todas las variables latentes de clase contribuyen al coste L_{KL}^{c}. Este, junto con el coste de reconstrucción, se minimizan cuando solo la variable latente correspondiente a la clase correcta es informativa. Así, reforzamos aun más la asociación de cada modo con una clase.

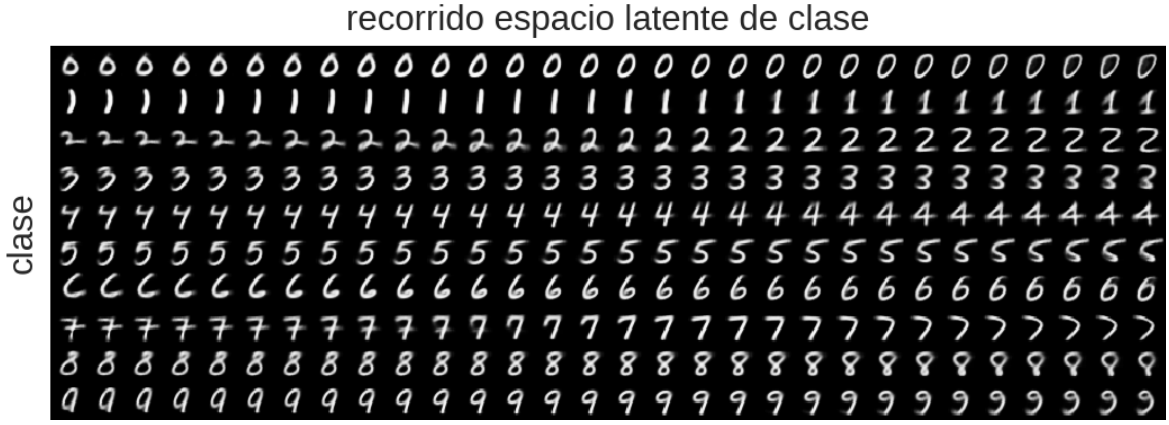


Figura 4.9: Reconstrucciones con nuestro modelo propuesto para valores de z_c equiespaciados en el rango $[-3, 3]$. Asignamos una dimensión a cada clase. Fijamos z_s=0. Observamos que cambiando el vector indicador de clase, modificamos el dígito que se regenera. El valor de la dimensión de clase regula variaciones en la forma del dígito reconstruido exclusivas a la clase seleccionada.

En contraste con los demás modelos generativos de variables latentes, el nuestro permite controlar de forma independiente la clase del dato generado, las variaciones intraclase y las variaciones interclase. En la Figura 4.9, generamos dígitos a partir de muestras equiespaciadas de z_c para cada clase. Observamos que z_c captura información propia de la forma de cada clase: si el 4 tiene forma cerrada o abierta, si el 2 tiene base recta o con lazo, si el 8 tiene uno de los círculos más grande que el otro. En las variables latentes compartidas z_s, representadas en Figura 4.10, capturamos variaciones comunes a todas las clases como: la inclinación del dígito, el grosor del trazo o la anchura del dígito. En el Apéndice D, mostramos que estas dimensiones controlan los mismos factores de variación independientemente del dígito.

En la Figura 4.11, recorremos el espacio latente de *anchor-VAE*. Observamos que las representaciones obtenidas están mucho más entrelazadas que las de nuestro modelo. En este caso, la distinción de clase es el principal factor de variación en la mayoría de las dimensiones informativas. La dificultad para obtener un espacio latente estructurado e interpretable viene de intentar modelar variables latentes multimodales con una única distribución normal. Los resultados son similares o peores para las demás técnicas de desentrelazado estudiadas.

Figura 4.10: Reconstrucciones para valores de cada dimensión de z_s equiespaciados en el rango $[-3, 3]$. En este caso, z_s es un vector de 5 dimensiones. Activamos la quinta variable discreta de clase y fijamos z_c=0. Observamos que las variables latentes compartidas de nuestro modelo aprenden representaciones desentrelazadas. La primera dimensión codifica la inclinación del dígito. La segunda dimensión codifica si el grosor de trazo es más fuerte en la parte superior o inferior del dígito. La tercera codifica la altura. La cuarta guarda la información de la anchura del dígito y la quinta la del grosor de trazo.

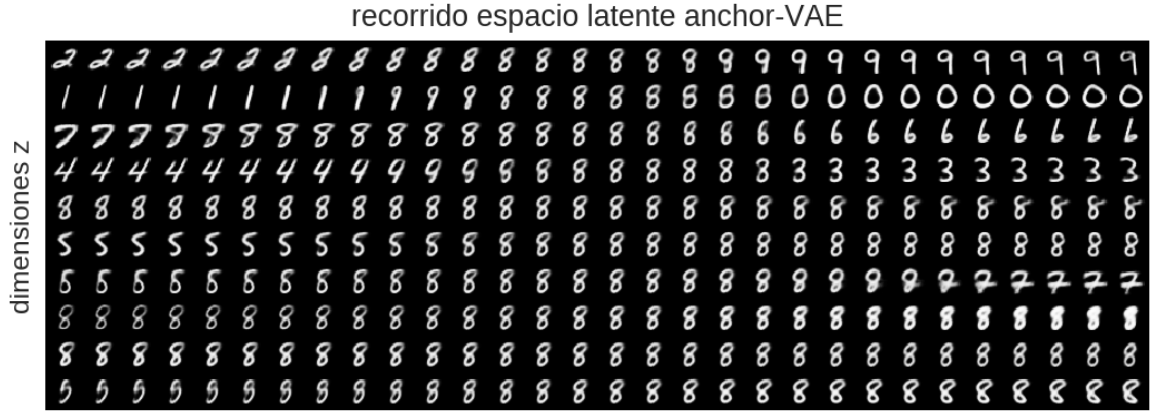


Figura 4.11: Reconstrucciones de *anchor-VAE* para valores de cada dimensión de z equiespaciados en el rango $[-3, 3]$. Observamos que, a pesar del uso de la técnica de desentrelazado, la información de clase se distribuye por múltiples dimensiones. La única dimensión completamente desentrelzada es la octava, la cual codifica el grosor de trazo.

Al no imponer una distribución a priori normal a un espacio latente en el que queremos capturar información distribuida multimodalmente, nuestro método consigue un mayor grado de desentrelazado que los demás modelos de la literatura. En el Apéndice D, mostramos mapas de reconstrucciones para *β-VAE* y para *C-VAE*.

4.6. Evaluación de modelos generativos

Hemos visto que el modelo propuesto permite más precisión y flexibilidad que un VAE tradicional a la hora de hacer *representation learning*. Aun así, queremos verificar empíricamente que capta mejor el *manifold* de los datos que los demás modelos de variables latentes existentes.

Utilizamos la técnica de evaluación de modelos generativos propuesta por [Odena et al., 2017]. Esta consiste en entrenar un clasificador exclusivamente con datos generados por un modelo generativo entrenado con el *train set*. Cuanta más información capte nuestro modelo generativo en su espacio latente, más información se transmitirá al clasificador y mejor clasificará este el *test set*. Adicionalmente, el espacio latente debe estar

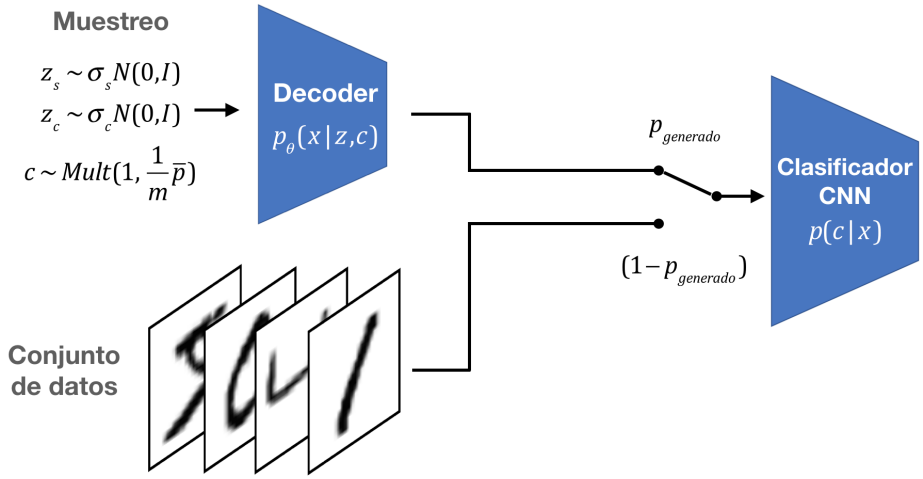


Figura 4.12: Esquema de entrenamiento del clasificador. Con una probabilidad $p_{generado}$, le enviaremos imagenes generadas en lugar de originales. El decoder mostrado es el del modelo propuesto en la sección anterior, aunque puede ser sustituido por cualquier otro modelo generador. $\overline{p}$ es un vector uniforme.

estructurado de forma que podamos tomar muestras aleatorias de él para generar datos de entrenamiento. Extendemos esta técnica evaluando el rendimiento del clasificador para distintas proporciones de datos de entrenamiento originales y generados Figura 4.12. Con esto, queremos demostrar que nuestro modelo es capaz de captar con suficiente precisión el *manifold* de los datos como para poder explorarlo por fuera de la zona donde se encuentran los ejemplos de entrenamiento originales. Esperamos poder generar datos plausibles a la vez que diferentes a los del conjunto de entrenamiento, que nos permitan aprender a clasificar mejor el *test set*.

Nuestro modelo se presta para este tipo de tarea ya que nos permite elegir la clase del dígito generado y así tener etiquetas con las que entrenar. A su vez, debemos compararlo con otros modelos generativos que también aumentan su espacio latente con información de clase. CVAE, propuesto por [Sohn et al., 2015], extiende el modelo del VAE tradicional al aportar información de clase tanto al decoder como al encoder. CGAN, propuesto por [Mirza and Osindero, 2014] condiciona tanto al modelo generador como al discriminador de una GAN con la etiqueta de clase. AC-GAN, propuesto por [Odena et al., 2017], aporta información de clase al generador y añade un objetivo de clasificación a una GAN. Los detalles del funcionamiento de estas redes y ejemplos de muestras generadas por cada una de ellas se pueden encontrar en Apéndice D. Para establecer unas condiciones justas en los experimentos, utilizamos el mismo número y tamaño de capas en todos los modelos que comparamos.

Queremos remarcar que la generación de datos nuevos es solo una de las posibles aplicaciones de un VAE. Nos centramos en ella ya que la podemos utilizar como medida de fidelidad objetiva.

Implementamos todos los modelos para nuestros experimentos en Python, utilizando la librería Pytorch. Las arquitecturas de estos se pueden ver en el Apéndice B. Entrenamos las redes en el cluster de computación del grupo de investigación ViVoLab. El optimizador utilizado es ADAM [Kingma and Ba, 2014]. Trabajamos con el conjunto da datos MNIST [LeCun and Cortes, 2010]. También realizamos una segunda tanda de experimentos sobre fashion-MNIST [Xiao et al., 2017]. Los detalles de esta se encuentran en el Apéndice D.

Figura 4.13: Dígitos regenerados con nuestro modelo a partir de muestras aleatorias del espacio latente. Para generar dígitos atípicos, que ayuden a regularizar el clasificador, muestreamos de $N(0,\, 1.4 \cdot I)$. Al comparar estos dígitos con los de la Figura 4.8, observamos trazos de mala calidad y características propias de alguien escribiendo con prisa o mala letra.

Para cada dígito generado, elegimos una de de las m clases posibles muestreando de una distribución multinomial. Las clases son equiprobables. Con σ_s y σ_c regulamos la desviación típica de la normal de la que muestreamos las variables latentes continuas. Por simplicidad, y para facilitar la comparación con los demás modelos, que no distinguen entre variables de clase y compartidas, fijamos $\sigma_s{=}\sigma_c$. Nos referiremos sencillamente a este parámetro como σ.

Todos los modelos generadores han sido entrenados con $p(z){=}N(0, I)$. Suponiendo que captamos bien el *manifold* de los datos, elegir una desviación típica mayor que 1 nos permitirá salir de las zonas del espacio latente correspondientes a datos de entrenamiento y generar datos con características no vistas antes. Hacemos esto en la Figura 4.13.

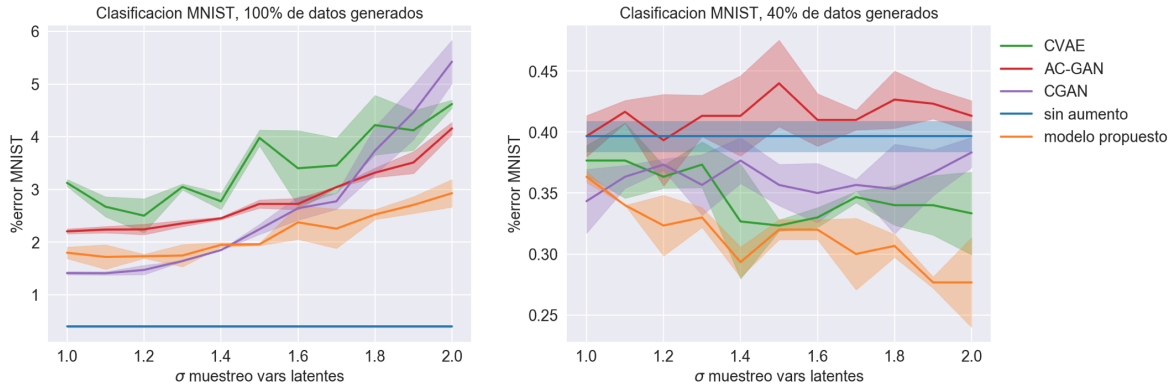


Figura 4.14: Porcentaje de error en MNIST obtenido al entrenar una red clasificadora con un 100 % (izquierda) y un 40 % (derecha) de probabilidad de utilización de datos generados. Variamos σ, la desviación típica de la distribución normal de la que muestreamos las variables latentes, entre 1 y 2. Utilizamos estas muestras para generar los datos. Los trazos centrales representan las medias de los valores de error. Los trazos transparentes representan desviaciones típicas. Repetimos cada experimento tres veces.

En la Figura 4.14, podemos observar los resultados obtenidos al entrenar una red clasificadora solo con datos generados, $p_{generado}{=}1$. Esta medida nos dice cuánta información del *train set* ha captado nuestro modelo. Los resultados son significativamente peores que al entrenar con la base de datos original. Esto se debe a que los modelos generativos no capturan

todas las variaciones o anormalidades de los datos. Nuestro modelo da resultados competitivos con las GAN para valores de σ bajos. Al aumentar este parámetro, el rendimiento de todos los modelos se deteriora porque sus generaciones se vuelven menos similares a los dígitos originales. El nuestro es el más lento en empeorar, indicando que sigue generando imágenes realistas e informativas a pesar de estar explorando una parte del espacio latente alejada de la zona de los datos de entrenamiento.

Aunque la función objetivo de las GAN las haces capaces de aprender a generar muestras más realistas que los VAE, no modelan bien el *manifold* de los datos. Las GAN no buscan obtener la probabilidad condicional de sus variables latentes, sencillamente utilizan el muestreo de estas como una fuente de variabilidad. Vemos que al aumentar σ y salirnos de la zona del espacio latente donde se encuentran los ejemplos de entrenamiento, el rendimiento de estas se mantiene o empeora en todos los experimentos. Podríamos interpretar que se sobreajustan al rango de valores visto en el entrenamiento. Las GAN no consiguen generar muestras con información nueva.

El segundo experimento de la Figura 4.14, con $p_{generado}$=0.4, es un indicador de la capacidad de los modelos de generar información nueva, distinta a la del *train set*, pero complementaria para la clasificación. Como la mayoría de los datos utilizados siguen siendo de la base de datos original, para conseguir mejoras se necesitan imágenes generadas con características no vistas antes. Nuestro modelo da los mejores resultados para todos los casos con σ>1. Además, sus resultados mejoran consistentemente con el aumento de σ. Sigue generando muestras informativas al explorar zonas del *manifold* lejanas a los datos de entrada.

Tabla 4.1: Porcentaje de error mínimo en MNIST conseguido por red clasificadora al entrenarla con una mezcla de imágenes del *train set* original y de datos generados por cada modelo tratado en esta sección. Repetimos cada experimento 3 veces.

modelo	media (%)	std (%)
sin aumento	0.3966	0.01247
modelo propuesto	**0.2766**	**0.00471**
CVAE	0.3233	0.00471
AC-GAN	0.3933	0.03771
CGAN	0.3433	0.02624

Hay un número limitado de imágenes en el dataset original. Sin embargo, podemos tomar infinitas muestras diferentes del espacio latente. Así, como cada generación es diferente a las anteriores, el entrenamiento con imágenes generadas evita el sobreajuste del clasificador a los datos de entrenamiento. Tiene un efecto regularizador. Sin embargo, no podemos atribuir toda la mejora de resultados a este efecto. El hecho de que obtengamos errores más bajos al generar datos con nuestro modelo, demuestra que este es capaz de captar más información que los demás en su espacio latente. Es un modelo más expresivo.

El clasificador que entrenamos es una CNN sencilla, explicada en Apéndice B. Sin embargo, como vemos en la Tabla 4.1, al entrenarla con datos aumentados por nuestro modelo, conseguimos un error medio de 0.27 % en MNIST. Se trata de un resultado muy competitivo, solo obtenido hasta la fecha con redes mucho más grandes y complejas. MNIST tiene un subgrupo de imágenes de dígitos muy difíciles de clasificar, incluso para humanos. Pasar de un error de 0.4 % a 0.27 % en MNIST, es pasar de un resultado fácilmente obtenible a uno comparable con las mejores soluciones.

En la figura Figura 4.15, observamos las tasas de error obtenidas al variar $p_{generado}$.

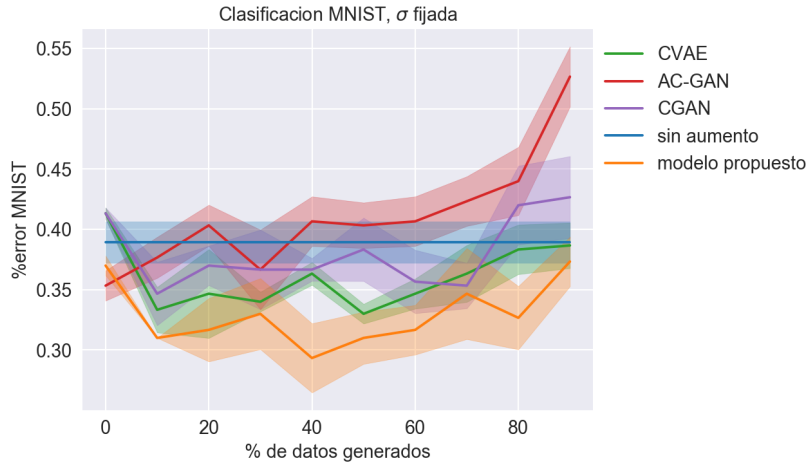


Figura 4.15: Porcentaje de error en MNIST obtenido al entrenar una red clasificadora con distintas probabilidades de utilizar datos generados. Utilizamos σ=1 para las redes GAN y σ=1.4 para CVAE y para nuestro modelo propuesto. Escogemos estos valores ya que dan los mejores resultados con cada red. Los trazos centrales representan las medias de los valores de error. Los trazos transparentes representan desviaciones típicas. Repetimos cada experimento tres veces.

Nuestro modelo consigue los mejores resultados para todos los experimentos, demostrando su consistencia.

4.7. Aprendizaje semisupervisado

Nuestro modelo crea una asociación entre sus variables discretas y la clase introduciendo un objetivo de clasificación. Utilizando el método de reparametrización de variables discretas con el estimador *Gumbel-Softmax* propuesto por [Jang et al., 2016], podemos aprender una distribución para nuestras variables discretas sin la necesidad de supervisión. El modelo *joint-VAE*, propuesto por [Dupont, 2018] de forma concurrente con nuestro trabajo, hace esto para combinar variables continuas y discretas en el espacio latente.

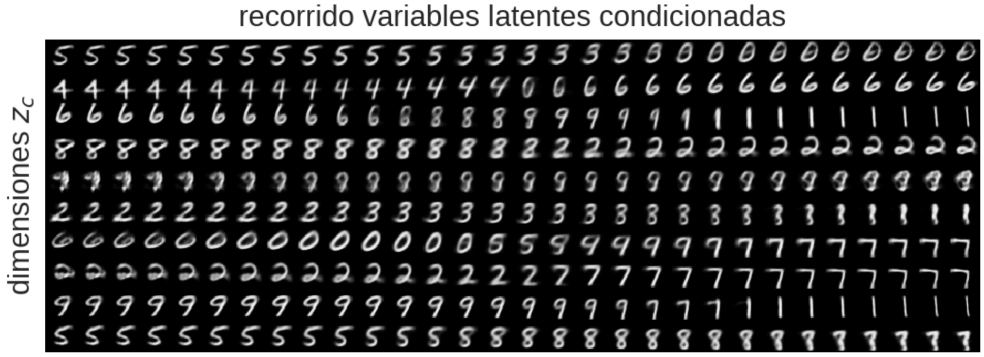


Figura 4.16: Reconstrucciones de nuestro modelo sin objetivo de clasificación para valores de cada dimensión de z_c equiespaciados en el rango $[-3, 3]$. Fijamos las variables latentes compartidas a 0, z_s=0.

Reprogramamos el modelo propuesto en la Sección 4.5, sustituyendo el clasificador por una reparametrización con *Gumbel-Softmax*. Así, podremos aprender distribuciones multimodales

en el espacio latente, con modos dependientes de factores aprendidos en lugar de la clase. Esto podría ser útil para trabajar con conjuntos de datos sin etiquetas de clase. Realizamos un experimento tentativo con este modelo y la base de datos MNIST. En la Figura 4.16, mostramos las reconstrucciones obtenidas al recorrer las dimensiones de z_c. La red codifica varias clases con la misma variable discreta. Mostramos una representación t-SNE de las activaciones tras la primera capa del reconstructor en la Figura 4.17. En ella, podemos ver cómo el *decoder* recombina la información proveniente de las variables latentes dependientes del vector discreto, con la de las compartidas. Observamos que la información se distribuye multimodalmente, separándose los datos por clase pero, a su vez, separando las clases en distintos modos.

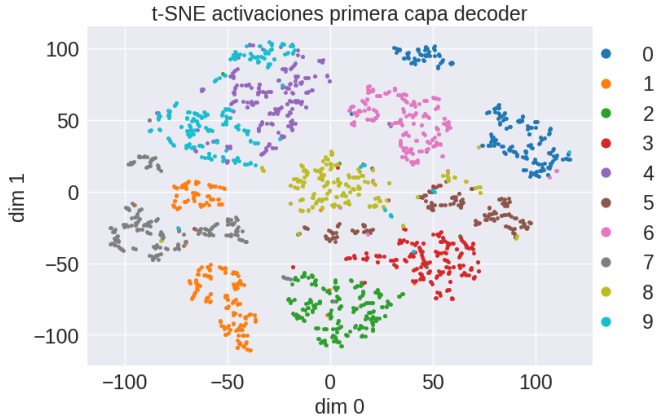


Figura 4.17: Representación t-SNE de las activaciones de la primera capa del decoder de nuestro modelo al ser entrenado sin objetivo de clasificación y con reparametrización *Gumbel-Softmax*.

A la vista de estos resultados, nos planteamos el uso de nuestro modelo para el aprendizaje semi-supervisado. Este consiste en entrenar un clasificador con una base de datos para la que solo conocemos la clase de un subconjunto de las muestras. Se trata de un campo muy interesante ya que existen grandes bases de datos que no se pueden explotar por la falta de etiquetas de clase. Habitualmente, clasificar muestras requiere a un experto humano y resulta muy caro. Es el caso de la detección de patologías en el campo biomédico.

Como demuestran [Kingma et al., 2014], el VAE presenta un buen marco para hacer aprendizaje no supervisado. Estos aprovechan la segmentación que se crea en el espacio latente de forma no supervisada para entrenar un clasificador. Introducen variables discretas para modelar la clase. La desventaja de su modelo es que, al no poder propagar gradientes por el camino discreto, se encuentran limitados a calcular $p_\theta(c|x)$ con el teorema de Bayes. Esto es una tarea muy costosa computacionalmente. Además, al no contemplar la existencia de variaciones intraclase en los datos, su modelo no puede aprovecharse de su detección para reforzar el clasificador.

Utilizando nuestro modelo, con el subconjunto de datos clasificados deberíamos poder crear una relación entre las variables discretas y la clase. Después, lo entrenaríamos con los datos sin etiquetas, sin coste de clasificación y reparametrizando las variables discretas con *Gumbel-Softmax*. Así, aprovecharíamos la dependencia entre clasificar bien y reconstruir bien de nuestro modelo para reforzar las asociaciones creadas entre las variables dependientes del vector discreto y las clases. Opinamos que el aprendizaje semisupervisado es una vía de trabajo futuro muy prometedora.

Capítulo 5

Conclusiones y Trabajo Futuro

En el *machine learning*, la regularización de modelos y el aprendizaje de características relevantes pueden entenderse como un mismo propósito. En este trabajo, exploramos nuevas técnicas de regularización y *representation learning*. Nos centramos en el uso de algoritmos de clusterizado y de modelos de variables latentes.

Realizamos un análisis e implementación de las redes de cápsulas propuestas por [Sabour et al., 2017]. Realizamos una interpretación propia de las técnicas de regularización utilizadas por este tipo de red, haciendo énfasis en cómo las podemos utilizar en otros tipos de redes neuronales.

Proponemos una técnica de visualización de información en redes neuronales convolucionales. Esta consiste en representar las activaciones espaciales de las capas convolucionales de la red, superponiéndoles sus correspondientes campos receptivos. Esta técnica nos permite ver los factores en función de los cuales las redes separan la información.

Proponemos un algoritmo de clusterizado para las activaciones de las redes neuronales clasificadoras llamado *margin-centerloss*. Este guarda un centroide por clase. Utilizamos un coste basado en un margen mínimo de distancia entre los datos, los centroides de sus clases y los centroides de las demás clases. La posición de los centroides se desplaza hacia la posición promedio de los puntos de su clase en cada iteración. Mostramos cómo, a diferencia de algoritmos existentes, el nuestro promueve tanto la separación entre clases como la compactación intraclase.

Proponemos el uso del anterior algoritmo de clusterizado para obtener medidas de incertidumbre sobre las decisiones de las redes clasificadoras. La función objetivo de entropía cruzada, junto a la activación *softmax*, causan un efecto de sobreconfianza. Es común que las redes clasificadoras, al equivocarse, den probabilidades muy altas. Nuestro método utiliza *margin-centerloss* para organizar las activaciones de la última capa de estas redes de forma que podamos medir cómo de atípico es un nuevo dato en el contexto de todos los anteriores. Se trata de un método sencillo, de aplicabilidad general a redes clasificadoras, que no hace suposiciones a priori sobre la distribución de los datos. Conseguimos obtener medidas robustas de confianza sobre las decisiones de nuestro modelo en la clasificación de la base de datos MNIST.

Los resultados obtenidos nos hacen ver como muy prometedor el campo de la obtención de medidas de incertidumbre en sistemas de *deep learning*. El método propuesto requiere guardar el valor de las activaciones del conjunto de entrenamiento para poder evaluar datos nuevos. Como trabajo futuro, nos gustaría extender nuestro método para solo guardar puntos de interés y así poder trabajar con *big data*. También nos gustaría evaluar el funcionamiento

del método con bases de datos adicionales.

Estudiamos el uso de modelos de variables latentes para *representation learning*. Exponemos el problema de modelar fenómenos, cuyas ocurrencias se distribuyen de forma multimodal, en espacios latentes con distribuciones a priori unimodales. Proponemos un nuevo algoritmo de autoencoder variacional, el cual es capaz de modelar simultáneamente fenómenos tanto unimodales como multimodales que se encuentran en los datos. Entrenamos nuestro modelo con la base de datos MNIST, condicionando los distintos modos de las variables latentes a las clases. Mostramos que las representaciones aprendidas por nuestro algoritmo son de mayor calidad y presentan un mayor grado de desentrelazado que las que se pueden conseguir con otras técnicas de desentrelazado de la literatura. Mostramos que nuestro modelo aprende a distinguir factores de variación comunes a todas las clases, como el grosor del trazo y la anchura del dígito, de factores de variación exclusivos a las clases, como si el 2 tiene una base con lazo o una base plana.

Presentamos una técnica para la evaluación de modelos generativos que extiende a la propuesta por [Odena et al., 2017]. Esta, consiste en entrenar una red clasificadora con muestras sintéticas, generadas por el modelo a evaluar. Se utiliza la tasa de error obtenida en el conjunto de validación como métrica. Nosotros proponemos entrenar el clasificador con distintas proporciones de datos originales y sintéticos. Además, utilizamos distintos valores para la desviación típica de la distribución de la que tomamos muestras para generar datos. Con esto, queremos salirnos de la zona del *manifold* de los datos donde se encuentran las muestras de entrenamiento y generar muestras plausibles con características nuevas. Así, no solo medimos cómo de bien los modelos siendo evaluados han captado los fenómenos presentes en los datos de entrenamiento, sino que también podemos medir su capacidad para la generalización.

Utilizamos esta técnica para comparar nuestro algoritmo con otros modelos generativos que utilizan información de clase de la literatura. Obtenemos el menor error de clasificación al generar datos con nuestro algoritmo. Conseguimos reducir la tasa de error en MNIST de 0.4 %, con el clasificador base, a 0.27 % al entrenar con una mezcla de datos originales y generados. Esto demuestra la utilidad del entrenamiento con muestras sintéticas como método de regularización.

Una vía prometedora de trabajo futuro es el aprendizaje semisupervisado. Se trata de un campo muy interesante ya que, en muchos casos, no se disponen etiquetas de clase para las bases de datos y es caro obtenerlas. Aprovechando la dependencia que se crea entre clasificar imágenes y reconstruirlas en el modelo propuesto, creemos que será posible reforzar el clasificador con imágenes sin etiquetas de clase.

Implementamos todos los modelos en Python, utilizando la librería de código abierto de *deep learning* Pytorch. El entrenamiento de redes neuronales profundas conlleva un elevado coste computacional. Programamos todas las operaciones de forma matricial para que puedan ser realizadas eficientemente en GPUs con la plataforma de cómputo paralelo CUDA. Entrenamos nuestros modelos en el cluster de computación del grupo de investigación ViVoLab y en la plataforma Google Colaboratory.

Anexo A

Introducción a las redes neuronales y su terminología

A.1. Cómo funciona una red neuronal

Las redes neuronales son modelos gráficos computacionales, es decir, modelos gráficos dirigidos donde cada nodo representa una función cuyas entradas son las salidas de otros nodos. Como demuestran [Hornik et al., 1989], las redes neuronales multicapa son aproximadores universales. En este anexo, plantearemos una breve introducción a las redes neuronales basada en los libros de [Bishop, 2006] y [Goodfellow et al., 2016].

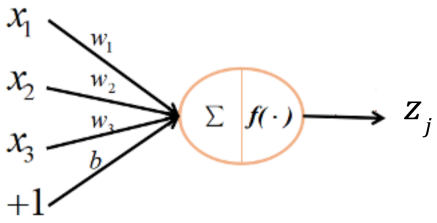


Figura A.1: Esquema de una neurona o nodo. Este computa una suma de las activaciones de los nodos de la anterior capa de la red, ponderada por los pesos w. Al resultado se le aplica la función de activación f.

Planteamos un ejemplo sencillo. Supongamos que tenemos la función $y(x)$, cuya entrada es el vector $x = (x_1, ..., x_d)$. $y(x)$ viene dada por una red neuronal como la de la Figura A.2. En este caso, se trata de una red con una única capa oculta. Cada nodo de la red realiza las operaciones representadas en Figura A.1. Nuestro conjunto de datos viene dado por N muestras, x^n, con sus correspondientes objetivos t^n, $D = \{x^n, t^n\}$.

$$z_j = h(\sum_{i=0}^{d} w_{ji}^1 x_i), \quad x_0{=}1 \tag{A.1}$$

$$y_k = \sum_{j=0}^{M} w_{kj}^2 z_j, \quad z_0{=}1 \tag{A.2}$$

Descomponemos y en las operaciones realizadas por cada capa. (A.1) describe las operaciones de la primera capa, siendo h la función de activación utilizada. Esta última, es una función no lineal, que normalmente se aplica elemento a elemento. Llamamos a los

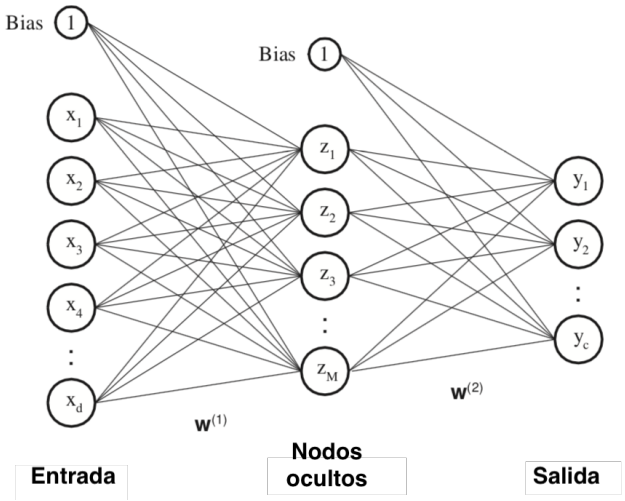


Figura A.2: Red neuronal con una capa oculta.

valores del vector z, las activaciones de la primera capa. Las operaciones de la capa de salida vienen dadas por (A.2). w^1 son los pesos de la primera capa y w^2 los de la segunda. El proceso de evaluación de $y(x)$ se denomina *forward propagation*. Es importante remarcar que todas las funciones que se aplican en la evaluación de una red neuronal deben ser diferenciables.

Las dos tareas más comunes realizadas por redes neuronales son la regresión y la clasificación. En ambas, el objetivo es encontrar los pesos óptimos de forma que se minimice una función de coste E, también llamada función objetivo.

En una regresión, la red predice un valor de $y \in \mathbb{R}$. Planteamos la ecuación $t = y(x) + \epsilon$, siendo ϵ el error o residuo que queremos minimizar. Suponiendo que los residuos se distribuyen normalmente, podemos escribir (A.3), donde β es la precisión del ruido. Para nuestro conjunto de datos formado por N muestras independientes e idénticamente distribuidas, nos proponemos maximizar la función de verosimilitud (A.4). Operando, llegamos a que esto es lo mismo que minimizar el error cuadrático (A.5). Esta será nuestra función objetivo.

$$p(t|x, w) = N(t|y(x, w), \beta^{-1}) \tag{A.3}$$

$$p(\boldsymbol{t}|\boldsymbol{x}, w) = \prod_{n=1}^{N} p(t^n|x^n, w) \tag{A.4}$$

$$E_{mse}(w) = \frac{1}{2} \sum_{n=1}^{N} \|y(x^n, w) - t^n\|_2^2 \tag{A.5}$$

En el caso de la clasificación, la red neuronal parametriza la distribución multinomial condicional $p(c|x)$, siendo x el dato a clasificar y c el vector de clases. Podemos expresar la distribución condicional de los objetivos como (A.6). Tomando como función de error la log-verosimilitud negativa, obtenemos la entropía cruzada entre la salida de la red y, y el valor objetivo t, (A.7).

$$p(\boldsymbol{t}|\boldsymbol{x}, w) = \prod_{c=1}^{C} y_c(\boldsymbol{x}, w)^{t_c} \tag{A.6}$$

$$E_{CE}(w) = -\sum_{n=1}^{N}\sum_{c}^{C} t_c^n \log y_c(x^n, w) \tag{A.7}$$

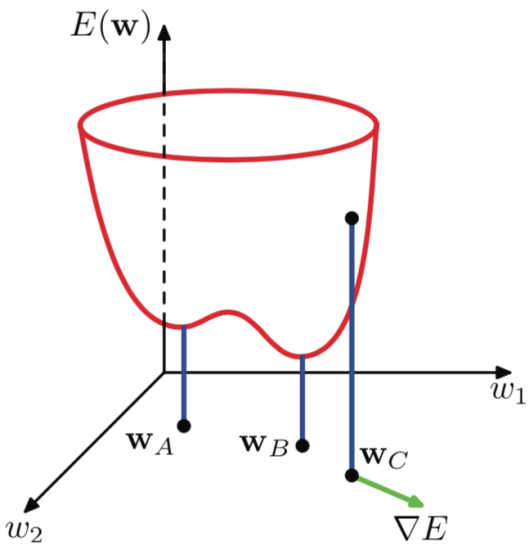


Figura A.3: La actualización de los pesos se hace por descenso de gradiente, tomando pasos en la dirección opuesta al gradiente de la función de error. El proceso de optimización puede entenderse como la exploración de una superficie, buscando su mínimo. Figura tomada de [Bishop, 2006].

Nos disponemos a optimizar los parámetros de la red para minimizar la función de coste. Podemos interpretar el problema de maximizar la verosimilitud, o minimizar E, respecto de los parámetros del modelo como la búsqueda de un mínimo en una superficie de error. Este concepto queda ilustrado en Figura A.3. Las relaciones no lineales entre E y los parámetros de la red hacen que la superficie del error no sea convexa. Habrá múltiples puntos estacionarios en el espacio de w, con distintos valores de $E(w)$. Por consiguiente, habrá múltiples mínimos locales. Convenientemente, no nos es necesario encontrar el mínimo global, solo uno local que nos dé resultados suficientemente buenos.

No existe una solución analítica al problema de encontrar w tal que $\nabla_w E$=0. Recurrimos a procedimientos iterativos. Existe una literatura extensa sobre la optimización de funciones continuas no lineales. La mayoría de las técnicas se basan en escoger un valor inicial w_0, a partir del cual podemos explorar el espacio de parámetros en sucesivos pasos con la expresión (A.8).

La técnica más común para optimizar redes neuronales es el descenso de gradiente. En esta, obtenemos Δw como (A.9), siendo η una constante conocida como la tasa de aprendizaje. En cada iteración, estamos evaluando la dirección de máxima pendiente para nuestro punto w_τ, y dando un paso en esa dirección. η regula el tamaño del paso.

$$w_{\tau+1} = w_\tau + \Delta w \tag{A.8}$$

$$\Delta w = -\eta \frac{\partial E(w)}{\partial w} \tag{A.9}$$

Como la función de error se define con respecto a todo D, calcular ∇E requiere evaluar todos los ejemplos de entrenamiento. Esto se llaman método de lote completo o *batch*. Sin embargo, existen métodos más eficientes conocidos como métodos de *mini batch*. En ellos dividimos el conjunto de entrenamiento en subconjuntos de muestras más pequeños. En cada iteración, calculamos ∇E respecto del *minibatch* y lo utilizamos para dar un paso en el espacio de parámetros. De esta manera, estamos utilizando un estimador insesgado del gradiente respecto al conjunto de datos completo. Este algoritmo se conoce como descenso de gradiente estocástico. Tiene la ventaja de ser menos intensivo computacionalmente, converger más rápido y habitualmente llegar a mínimos más profundos que el descenso de gradiente tradicional. La razón para esto último es que el estimador introduce varianza a nuestro cálculo del gradiente. Esta ayuda al modelo a salir de mínimos locales poco profundos en la superficie del error, algo que habría sido muy difícil siguiendo la dirección del gradiente exacto. Explicado de otra manera, es improbable que un punto que es estacionario respecto a todo el conjunto de datos lo sea también respecto a un pequeño subconjunto de ellos.

Para calcular el gradiente del error respecto de cada parámetro de la red se utiliza la técnica de *backpropagation*. Al ser diferenciables todas las funciones que aplica una red neuronal, podemos calcular $\frac{\partial E}{\partial w}$ para todos los valores de w utilizando la regla de la cadena. De esta manera, podemos obtener los gradientes capa a capa, apoyándonos en los cálculos de la anterior capa para calcular los gradientes de la siguiente.

Aunque hay muchos tipos, en general, las funciones de activación, o no linealidades, suelen presentar un comportamiento lineal para valores de entrada pequeños y comportamientos menos lineales conforme crecen las entradas. Normalmente, inicializamos los pesos de las capas con valores pequeños. Así, nos aseguramos de que, al principio, las señales que se propagan por la red son de pequeña amplitud y esta presenta un comportamiento lineal. Podemos agregar múltiples operaciones lineales encadenadas en una sola. Esto hace que una red multicapa, operando en régimen lineal, sea un modelo muy sencillo, equivalente a una sola capa. Conforme se entrena la red, algunos pesos van creciendo. Esto hace que se comiencen a utilizar más las zonas no lineales de las funciones de activación, permitiendo el modelado de funciones más complejas. Así, las redes neuronales son modelos que comienzan siendo poco flexibles y ganan flexibilidad al entrenarse. Existe el riesgo de que el modelo se vuelva demasiado flexible y aprenda a modelar el ruido en los datos. Por ello, son comunes técnicas de regularización que penalizan el uso de pesos muy grandes o que paran el entrenamiento antes de llegar a la convergencia para evitar el sobreajuste.

A.2. Autoencoders y GANs

Un modelo generativo adversario, o GAN, está compuesto por dos redes neuronales, un generador y un discriminador. Las entradas al generador son un conjunto de variables aleatorias, generalmente muestreadas de una distribución normal. Estas hacen de variables latentes. A partir de ellas, el generador construye una muestra sintética. La entrada al discriminador puede ser una muestra del conjunto de datos original o una muestra sintética del generador. Su objetivo es clasificarlas en función de su origen. Se entrena como un

clasificador normal. El objetivo del generador es devolver muestras que hagan equivocarse al discriminador. Para entrenarlo, calculamos el gradiente de sus parámetros respecto del error de clasificación, utilizando la regla de la cadena para propagarlo a través del discriminador. Actualizamos los parámetros del generador invirtiendo el gradiente, de forma que damos pasos cuesta arriba, maximizando el error de clasificación. De esta manera, una GAN es un juego de minimax en el que las dos redes compiten. El objetivo es que el generador aprenda una transformación desde el espacio de las variables latentes, hasta el de los datos de entrada. Esto nos permite generar muestras nuevas realistas.

Un *autoencoder* es un modelo compuesto por dos redes, un *encoder* y un *decoder*. El *encoder* define una transformación desde el espacio de los datos de entrada hasta un espacio de menos dimensiones. El decoder define una transformación desde el espacio de bajas dimensiones hasta los datos de entrada. La función objetivo es una medida de distancia entre los datos originales y los reconstruidos por el *decoder*. De esta forma, la red aprende a generar una representación comprimida de los datos de entrada, manteniendo la mayor cantidad de información posible para poder reconstruirlos.

A.3. Terminología adicional

A continuación, explicamos en más detalle algunos términos adicionales, propios del *machine learning*, que utilizamos en este trabajo.

- *Aprendizaje supervisado*: Las tareas de *machine learning* en las que los datos de entrenamiento están formados por vectores de entrada junto a sus correspondientes vectores objetivo se conocen como aprendizaje supervisado. En estos casos, el objetivo es que el modelo aprenda a predecir los objetivos a partir de las entradas.

- *Aprendizaje no supervisado*: Las aplicaciones de *machine learning* en las que los datos de entrenamiento no van acompañados de objetivos se conocen como aprendizaje no supervisado. Generalmente, el objetivo del aprendizaje no supervisado es conseguir modelar la distribución de los datos.

- *Dropout*: Esta técnica, propuesta por [Hinton et al., 2012], consiste en anular aleatoriamente algunas de las activaciones internas de una red en cada evaluación. Al limitar los caminos por los que puede propagarse la información dentro de la red, se evita la creación de dependencias mutuas entre distintos filtros, las cuales hacen el modelo demasiado flexible. Los autores demuestran que, al reducir la capacidad de representación de la red, cada filtro es obligado a aprender a capturar información útil por su cuenta para resolver la tarea.

- *EM*: El algoritmo de maximización de expectación o EM permite encontrar soluciones de máxima verosimilitud para modelos con variables latentes. Es comúnmente utilizado para optimizar los parámetros de modelos de mezclas de Gaussianas (GMM).

- *One-hot*: Un vector indicador o *one-hot* es un vector de datos binarios en el que solo uno de los elementos está activo. En ML, es común codificar datos de carácter discreto con vectores indicadores.

- *ReLU*: Función de activación rectificadora lineal. Devuelve el máximo entre la entrada y 0, $ReLU(x) = \max(0, x)$. Es la no linealidad más utilizada en las redes neuronales

actualmente ya que, al no saturar, favorece la propagación de gradientes. Además, su coste computacional es muy bajo comparado con otras funciones de activación.

– *Softmax*: Función de activación que convierte un vector de valores reales en otro de valores pertenecientes al intervalo $[0, 1]$ cuya suma es 1. Así, la salida de esta función se puede interpretar como el vector de probabilidades para una distribución multinomial. Dado un vector x de D dimensiones, expresamos la función *softmax* en (A.10).

$$Softmax(x_i) = \frac{\exp x_i}{\sum_{d=1}^{D} \exp x_d} \tag{A.10}$$

– *Transfer learning*: Subdisciplina del machine learning que busca reutilizar el conocimiento adquirido al realizar una tarea para poder aplicarlo a tareas nuevas.

– *Ejemplos adversarios*: Entradas a una red neuronal especialmente diseñadas por un atacante para que la red cometa una decisión equivocada. La técnica más común para la creación de ejemplos adversarios es añadir a una imagen de entrada ruido de baja amplitud distribuido de forma que se maximice la probabilidad de error. Se busca que el cambio en la imagen sea imperceptible para un humano pero sea suficiente para que el dato cruce un umbral en el espacio de decisión.

– *Epoch*: Las técnicas de descenso de gradiente estocástico dividen el conjunto de entrenamiento en varios *mini batch*. En cada iteración se procesa un minibatch. Cada vez que se procesa el conjunto de datos enteros entero, se dice que ha transcurrido una *epoch*.

– *Hiperparámetros*: Parámetro de un modelo de ML que no se aprende. Debe ser ajustado antes del comienzo del entrenamiento. En general, es preferible que un sistema de ML tenga pocos hiperparámetros. Ejemplos de hiperparámetros son la tasa de aprendizaje η, el tamaño del *batch* y el número de *epochs* de entrenamiento.

– *Modelo discriminativo*: En ML, un modelo discriminativo es aquel que modela la dependencia entre las variables objetivo, t, y las entradas, x. A su salida, los modelos discriminativos devuelven la probabilidad condicional $p(t|x)$ directamente.

– *Modelo generativo*: Los modelos generativos estiman la distribución conjunta de las entradas y salidas $p(t, x)$. Así, son modelos que permiten tomar muestras de esta distribución para generar datos nuevos.

– *Test set*: Conjunto de datos de prueba. Se utilizan para validar el rendimiento de un modelo una vez que ha sido entrenado. Es importante evaluar nuestros modelos con un *test set*, ya que el rendimiento conseguido con el conjunto de entrenamiento no es fiable; puede ser que estemos obteniendo resultados muy buenos debido al sobreajuste.

– *Train set*: Conjunto de datos de entrenamiento.

– *Regularización*: Conjunto de técnicas que buscan asegurar que los modelos de ML aprenden a modelar la estructura subyacente en los datos y no el ruido. Generalmente, esto se consigue limitando su flexibilidad.

Anexo B

Arquitectura de las redes implementadas

En este anexo, describimos las arquitecturas de las redes neuronales utilizadas en este trabajo. Utilizamos la siguiente notación para las capas lineales: FC(dimensión entrada, dimensión salida). Las convolucionales: Conv2d(canales entrada, canales salida, tamaño del filtro, avance del filtro, padding). Las capas convolucionales traspuestas tienen la misma estructura que las capas convolucionales pero se designan: ConvT2d. Las capas de *max pooling*: Maxpool(tamaño filtro). Junto a cada capa, indicamos el uso de diezmado y la función de activación que aplicamos a su salida. Nos referimos a la operación de *batch normalization* por: BN.

Red clasificadora utilizada en los experimentos del Capítulo 3
Conv2d(1, 32, 5×5, 1, 2), Maxpool(2×2), ReLU
Conv2d(32, 64, 5×5, 1, 2), Maxpool(2×2), ReLU, BN
FC(64*7*7, 10), Softmax

Tabla B.1: Arquitectura utilizada en experimentos de clusterizado y visualización. Para la aplicación de *centerloss* y *margin-centerloss* añadimos una capa lineal adicional, sin función de activación, antes de la última capa.

Autoencoder utilizado en experimentos del Capítulo 4	
Encoder q_ϕ	Decoder p_θ
Conv2d(1, 64, 5×5, 1, 2), Maxpool(2×2), ReLU	FC(variables latentes, 1024), ReLU
Conv2d(64, 128, 5×5, 1, 2), Maxpool(2×2), ReLU	FC(1024, 7*7*128), ReLU
FC(64*7*7, 1024), ReLU	ConvT2d(128, 64, 4×4, 2, 1), ReLU
FC(1024, variables latentes)	ConvT2d(64, 1, 4×4, 2, 1), Sigmoid

Tabla B.2: Arquitectura utilizada para implementar técnicas de desentrelazado (β-VAE, anchor-VAE, C-VAE). También utilizada para implementar el algoritmo de variables latentes propuesto en la Sección 4.5. También utilizado para generar datos con el modelo propuesto en los experimentos de la Sección 4.6.

Red clasificadora utilizada en los experimentos del Capítulo 4
Conv2d(1, 256, 5×5, 1, 2), Maxpool(2×2), ReLU, BN
Conv2d(256, 128, 5×5, 1, 2), ReLU, BN
Conv2d(128, 128, 5×5, 1, 2), Maxpool(2×2), ReLU, BN
FC(128*7*7, 1024), ReLU
FC(1024, 10), Softmax

Tabla B.3: Arquitectura de la red clasificadora utilizada en los experimentos de la Sección 4.6.

Autoencoder utilizado en experimentos con Fashion-MNIST del Apéndice D	
Encoder q_ϕ	Decoder p_θ
Conv2d(1, 256, 5×5, 1, 2), Maxpool(2×2), ReLU	FC(variables latentes, 1024), ReLU
Conv2d(256, 128, 5×5, 1, 2), ReLU	FC(1024, 7*7*128), ReLU
Conv2d(128, 128, 5×5, 1, 2), Maxpool(2×2), ReLU	ConvT2d(128, 128, 4×4, 2, 1), ReLU
FC(128*7*7, 1024), ReLU	Conv2d(128, 256, 5×5, 1, 2), ReLU
FC(1024, variables latentes)	ConvT2d(256, 1, 4×4, 2, 1), Sigmoid

Tabla B.4: Autoencoder utilizado para experimentos de generación de datos con el conjunto Fashion-MNIST. Debido a que es un conjunto de datos es más complejo, utilizamos una red ligeramente más grande que las utilizadas con MNIST.

CGAN	
Generador	Discriminador
FC(variables latentes + 10, 1024), ReLU, BN	Conv2d(1, 64, 5×5, 1, 2), Maxpool(2×2), LeakyReLU(0.2)
FC(1024, 128*7*7), ReLU, BN	Conv2d(64, 128, 5×5, 1, 2), Maxpool(2×2), LeakyReLU(0.2), BN
ConvT2d(128, 64, 4×4, 2, 1), ReLU, BN	FC(128*7*7, 1024), LeakyReLU(0.2), BN
ConvT2d(64, 1, 4×4, 2, 1), Sigmoid	FC(1024, 1), Sigmoid

Tabla B.5: Arquitectura de la red CGAN utilizada en los experimentos de generación de datos.

AC-GAN	
Generador	Discriminador
FC(variables latentes + 10, 1024), ReLU, BN	Conv2d(1, 64, 5×5, 1, 2), Maxpool(2×2), LeakyReLU(0.2)
FC(1024, 128*7*7), ReLU, BN	Conv2d(64, 128, 5×5, 1, 2), Maxpool(2×2), LeakyReLU(0.2), BN
ConvT2d(128, 64, 4×4, 2, 1), ReLU, BN	FC(128*7*7, 1024), LeakyReLU(0.2), BN
ConvT2d(64, 1, 4×4, 2, 1), Sigmoid	FC(1024, 1), Sigmoid — FC(1024, 10), Softmax

Tabla B.6: Arquitectura de la red AC-GAN utilizada en los experimentos de generación de datos.

CVAE	
Encoder	Decoder
FC(28*28 + 10, 1024), ReLU	FC(variables latentes + 10, 1024), ReLU
FC(1024, 1024), ReLU	FC(1024, 128*7*7), ReLU
FC(1024, variables latentes)	ConvT2d(128, 64, 4×4, 2, 1), ReLU
	ConvT2d(64, 1, 4×4, 2, 1), Sigmoid

Tabla B.7: Arquitectura de la red CVAE utilizada en los experimentos de generación de datos.

Anexo C

Clusterizado y visualización

C.1. Técnicas de reducción de dimensionalidad

El método t-SNE, propuesto por [van der Maaten and Hinton, 2008], busca proyectar conjuntos de datos multidimensionales a dos dimensiones, manteniendo las relaciones de distancia entre puntos. Las distancias relativas entre cada par de puntos se calculan en el espacio de los datos originales, utilizando una función de base radial con la forma de la distribución *student-t*. Se define una función de coste como la diferencia entre las distancias en el espacio original y las del espacio bidimensional. Se optimizan los parámetros de la proyección respecto de este coste con un algoritmo de descenso de gradiente. Debido a su carácter iterativo y su dependencia en hiperparámetros para elegir la anchura de la función de base radial, emplear t-SNE resulta costoso.

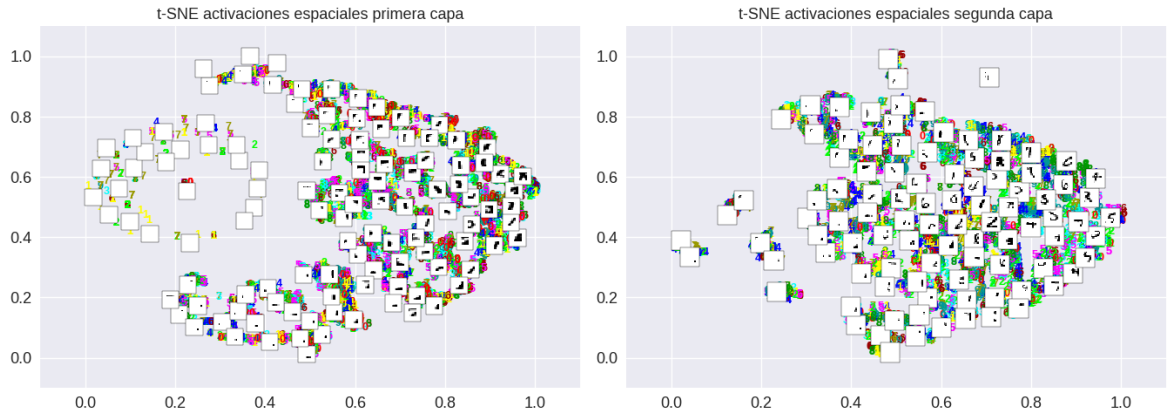


Figura C.1: Activaciones espaciales de la primera (derecha) y segunda (izquierda) capa de la CNN descrita en la Sección 3.1. Proyectamos los datos a dos dimensiones con la técnica t-SNE. Superponemos los campos receptivos de algunos puntos.

Se trata de una herramienta muy potente para representar separaciones de subgrupos de datos de muchas dimensiones, pero suele encontrar problemas con distribuciones continuas. Además, para minimizar la función de coste, t-SNE deforma las distribuciones originales de los datos en su proyección.

En la Figura C.1, observamos cómo t-SNE representa todos los puntos con campo receptivo vacío en un círculo, a pesar de que el valor de su vector de activación espacial es el mismo. Esta solución viene dada un un mínimo local en su función de coste. Estas deformaciones hacen que t-SNE no sea muy útil para visualizar la información capturada

por las capas convolucionales. La distribución de los puntos es información que no queremos perder.

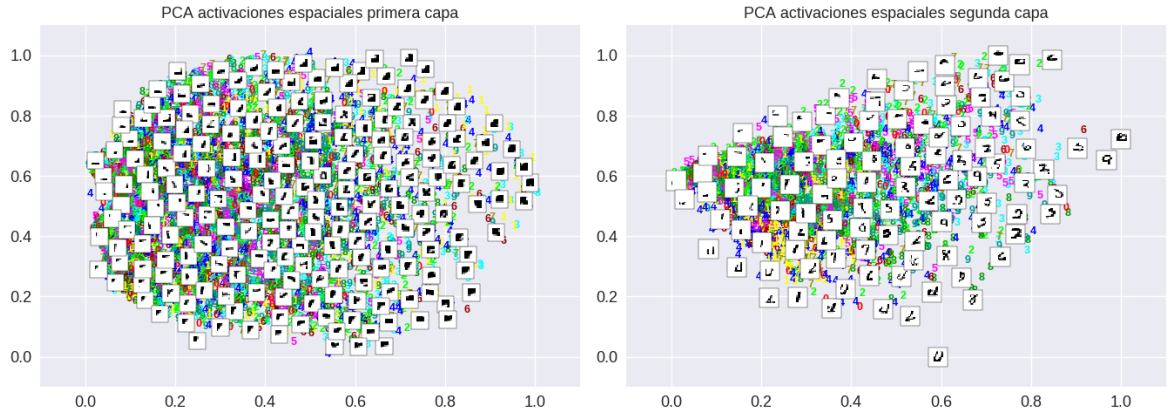


Figura C.2: Activaciones espaciales de la primera (derecha) y segunda (izquierda) capa de la CNN descrita en la Sección 3.1. Proyectamos los datos a dos dimensiones con la técnica PCA. Superponemos los campos receptivos de algunos puntos.

La técnica PCA consiste en realizar una proyección ortogonal de los datos sobre sus dimensiones de máxima varianza. Como vemos en la Figura C.2, esto tiene el inconveniente de que casi exclusivamente captura variaciones en el grosor de trazo, ya que esta es la característica que causa la mayor variación en la amplitud de las activaciones.

C.2. Cálculo de campos receptivos

Las no linealidades de las redes neuronales se aplican elemento a elemento. Las operaciones de filtrado y *pooling* son lineales. Estas se aplican sobre una porción local, del tamaño de la ventana utilizada, de los datos o activaciones de entrada. Como consecuencia, podemos agregar estas operaciones para calcular el campo receptivo de cada punto de una capa interna como si se tratase de una red de una capa.

$$lado_{CR} = l_1 + \sum_{k=2}^{K} (l_i - 1) \cdot s_{i-1} \tag{C.1}$$

$$avance_{CR} = \prod_{k=1}^{K} s_i \tag{C.2}$$

$$padding_{CR} = pad_1 + \sum_{k=2}^{K} pad_i \cdot s_{i-1} \tag{C.3}$$

Tomamos l_i como el lado del filtro de la capa i, s_i como su avance y pad_i como el tamaño de su padding. Para cada activación a la salida de la capa K-ésima de una red, podemos calcular el tamaño de su campo receptivo como (C.1). Podemos calcular el avance entre campos receptivos de puntos adyacentes en el espacio de activaciones como (C.2) y el padding equivalente sobre la imagen de entrada como (C.3).

C.3. Optimización CUDA de algoritmos de clusterizado

La arquitectura de computación paralela CUDA nos permite entrenar redes neuronales en tarjetas gráficas con gran rapidez, ya que las operaciones realizadas por estos modelos se pueden expresar como productos matriciales. El cómputo de estos es fácilmente paralelizable en GPUs.

La implementación original de *centerloss* de [Wen et al., 2016] no es óptima. Utiliza un bucle para actualizar las posiciones de los centroides y hace llamadas a la CPU para ordenar vectores. Como consecuencia, el coste del algoritmo crece linealmente con el tamaño del *batch*. A pesar de estar programada para funcionar con CUDA, no aprovecha bien la capacidad de cómputo paralelo de las GPU. El cálculo de este algoritmo y su diferenciación suponen un cuello de botella en el entrenamiento de nuestros modelos. El problema se agrava al trabajar con activaciones espaciales. Por cada imagen, tendremos tantas activaciones espaciales como píxeles. En nuestro caso, esto supone un tamaño de *batch* efectivo de 128×14×14=25088. Sin optimizar el algoritmo de clusterizado, clusterizar activaciones espaciales es una tarea intratable.

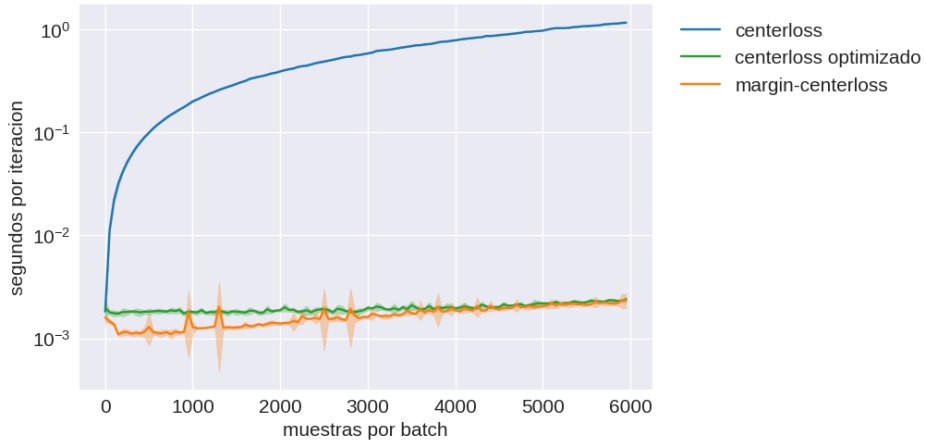


Figura C.3: Evolución del tiempo de cómputo por iteración, en segundos, para los distintos algoritmos de clusterización utilizados, al incrementar el tamaño del *batch*. Experimento realizado con 10 centroides en un espacio de 16 dimensiones. Tenemos en cuenta el tiempo de evaluación de la función de coste y del cálculo de su gradiente respecto a los parámetros de la capa inmediatamente anterior. Repetimos cada medida 10 veces. Las líneas continuas representan valores medios y los trazos transparentes representan las desviaciones típicas. Medidas tomadas en una GPU GTX 1060.

Reprogramamos *centerloss* de forma completamente matricial. Llamamos a esta implementación del algoritmo: centerloss optimizado. Programamos nuestro algoritmo propuesto *margin-centerloss* de la misma manera. En la Figura C.3, observamos cómo el coste de los algoritmos optimizados crece de forma logarítmica con el tamaño de *batch*.

Anexo D

Modelos de variables latentes

D.1. Demostración de la descomposición de $D_{KL}(q(z|x) \,\|\, p(z))$

Demostramos la igualdad (4.14) en (D.1).

$$
\begin{aligned}
&\frac{1}{N}\sum_{i=1}^{N} D_{KL}(q(z|x^{(i)}) \,\|\, p(z)) = \mathop{\mathbb{E}}_{p(x)}[D_{KL}(q(z|x) \,\|\, p(z))] \\
&= \mathop{\mathbb{E}}_{p(x)} \mathop{\mathbb{E}}_{q(z|x)}[\log \frac{q(z|x)}{p(z)}] \\
&= \mathop{\mathbb{E}}_{p(x)} \mathop{\mathbb{E}}_{q(z|x)}[\log \frac{q(z|x)q(z)\prod_{j=1}^{d} q(z_j)}{p(z)q(z)\prod_{j=1}^{d} q(z_j)}] \\
&= \mathop{\mathbb{E}}_{p(x)} \mathop{\mathbb{E}}_{q(z|x)}[\log \frac{q(z|x)}{q(z)}] + \mathop{\mathbb{E}}_{q(z)}[\log \frac{q(z)}{\prod_{j=1}^{d} q(z_j)} + \mathop{\mathbb{E}}_{q(z)}[\sum_{j=1}^{d} \log \frac{q(z_j)}{p(z_j)}] \\
&= \mathop{\mathbb{E}}_{p(x)} \mathop{\mathbb{E}}_{q(z|x)}[\log \frac{q(z|x)}{q(z)}] + \mathop{\mathbb{E}}_{q(z)}[\log \frac{q(z)}{\prod_{j=1}^{d} q(z_j)} + \sum_{j=1}^{d} \mathop{\mathbb{E}}_{q(z_j)}[\log \frac{q(z_j)}{p(z_j)}] \\
&= \mathop{\mathbb{E}}_{p(x)}[D_{KL}(q(z|x) \,\|\, q(z))] + D_{KL}(q(z) \,\|\, \prod_{j=1}^{d} q(z_j)) + \sum_{j=1}^{d} D_{KL}(q(z_j) \,\|\, p(z_j)) \\
&= I(x:z) \quad + \quad TC(z) \quad + \quad \sum_{j=1}^{d} D_{KL}(q(z_j) \,\|\, p(z_j))
\end{aligned} \tag{D.1}
$$

Formulamos esta demostración basándonos en las descomposiciones de $D_{KL}(q(z|x) \,\|\, p(z))$ realizadas por [Kim and Mnih, 2018] y [Chen et al., 2018]. Utilizamos componentes de ambas para llegar a nuestro resultado. Cabe destacar que el segundo y el tercer término se pueden separar solo si imponemos una distribución a priori cuyas dimensiones no están correladas. A su vez, esto es un requisito que debemos cumplir si queremos un espacio latente desentrelazado. $N(0, I)$ cumple este requisito. Estos dos últimos términos se podrían recombinar como mostramos en (D.2).

$$
D_{KL}(q(z) \,\|\, p(z)) = D_{KL}(q(z) \,\|\, \prod_{j=1}^{d} q(z_j)) + \sum_{j=1}^{d} D_{KL}(q(z_j) \,\|\, p(z_j)) \tag{D.2}
$$

D.2. Reconstrucciones con métodos de desentrelazado

En la Figura D.1, Figura D.2 y Figura D.3 mostramos reconstrucciones para valores equiespaciados de cada una de las dimensiones del espacio latente de β-VAE, C-VAE y anchor-VAE respectivamente. Utilizamos espacios latentes de 10 dimensiones en todos los modelos.

Figura D.1: Reconstrucciones de β-VAE para valores de cada dimensión de z equiespaciados en el rango $[-3, 3]$.

Figura D.2: Reconstrucciones de *C-VAE* para valores de cada dimensión de z equiespaciados en el rango $[-3, 3]$.

Para todos los casos, observamos que, a pesar del uso de las técnicas de desentrelazado, la información de clase se distribuye por múltiples dimensiones. Además, la variación de la amplitud de algunas de las dimensiones no causa variaciones en la reconstrucción. Son dimensiones no informativas, con D_{KL}=0.

Es difícil obtener representaciones desentrelazadas de los dígitos MNIST debido a la fuerte multimodalidad que exhiben los datos. Hay 10 clases, una por dígito, muy distintas. En este caso, $q(z, x)$ y $p(z)$ son muy distintos. Al tener que codificar la información de clase con variables continuas distribuidas normalmente, incurrimos un coste $D_{KL}(q(z|x) \,\|\, p(z))$ muy elevado. Para minimizar la función de error, el VAE busca un equilibrio entre error de reconstrucción y D_{KL}. Si D_{KL} es muy grande, porque el factor de variación más grande de los datos no encaja con la distribución a priori, la red utilizará muchas dimensiones para guardar esta información, pero no aprenderá a codificar otros factores que tengan menos repercusión en el error de reconstrucción. Así, obtenemos espacios latentes menos informativos

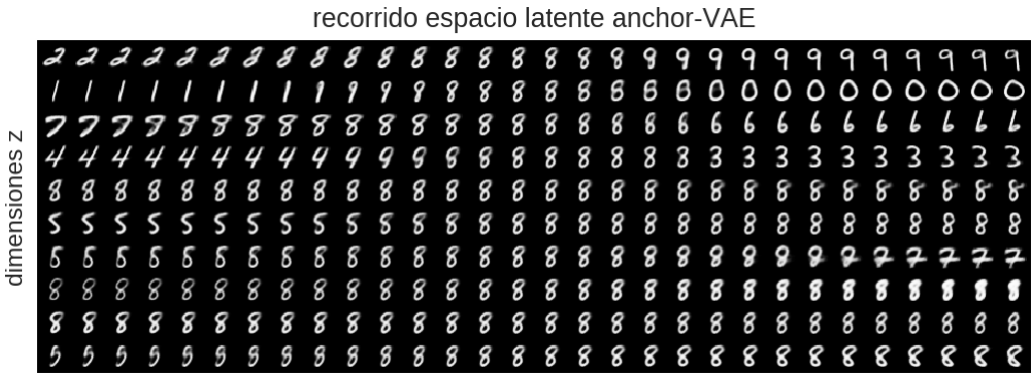


Figura D.3: Reconstrucciones de *anchor-VAE* para valores de cada dimensión de z equiespaciados en el rango $[-3, 3]$.

y más entrelazados que con un modelo que pueda captar bien los fenómenos distribuidos multimodalmente.

D.3. Distribución de la información en el espacio latente del VAE

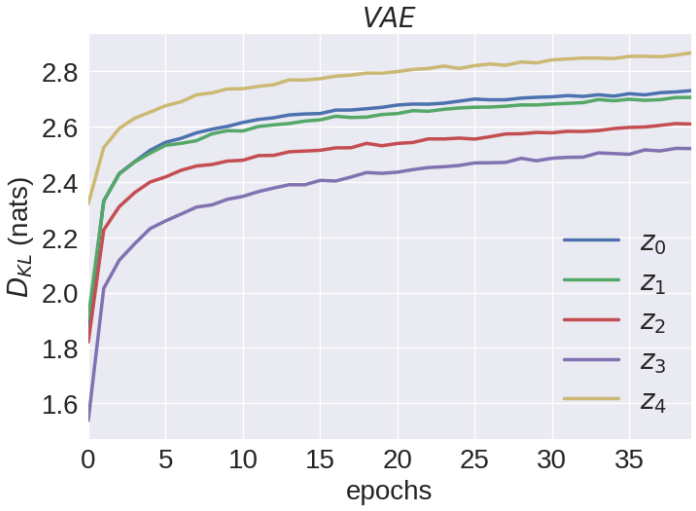


Figura D.4: .Mostramos $D_{KL}(q(z|x) \,\|\, p(z))$ para cada dimensión del espacio latente del VAE original.

En la Figura D.4, mostramos la cantidad de información codificada en cada dimensión del VAE original propuesto por [Kingma and Welling, 2013]. Obtenemos dimensiones más informativas que con los métodos de desentralazado de la Figura 4.5. Esto es debido a que estos métodos utilizan parámetros que aumentan el peso de $D_{KL}(q(z|x) \,\|\, p(z))$ en la función objetivo del modelo.

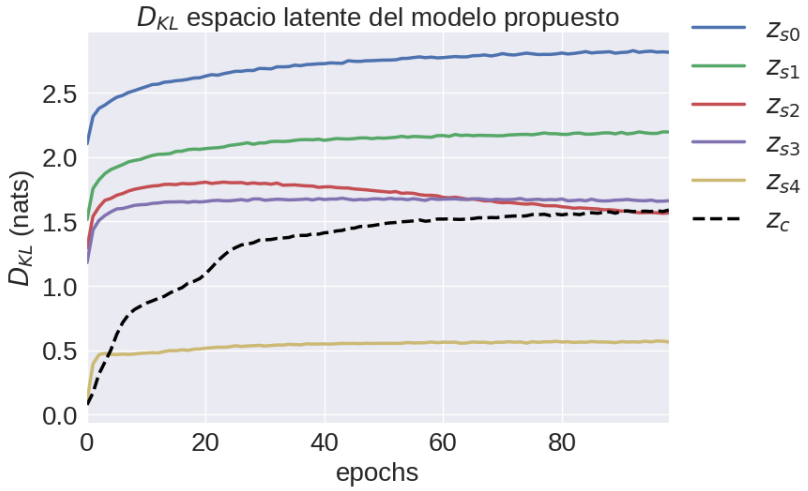


Figura D.5: Mostramos $D_{KL}(q(z|x) \,\|\, p(z))$ para cada dimensión del espacio latente de nuestro modelo. Utilizamos 5 variables latentes compartidas, z_s y una variable latente condicionada a cada clase z_c. Hay 10 clases. Al solo activarse una variable latente condicionada a la vez, sumamos el coste incurrido por todas y lo representamos con la línea negra discontinua.

D.4. Distribución de la información en el espacio latente del modelo propuesto

En la figura Figura D.5, mostramos la cantidad de información codificada en cada dimensión del modelo propuesto en la Sección 4.5. En este caso, utilizamos 5 dimensiones compartidas z_s y una dimensión condicionada a cada clase z_c. Observamos que z_c no codifica mucha información comparada con las dimensiones compartidas. Solo capta las variaciones intraclase. Esto es útil para que estas no sesguen las variables compartidas con información de clase y así podamos conseguir una representación desentrelazada. La distinción de clase viene dada por la dimensión de z_c que se activa en cada caso.

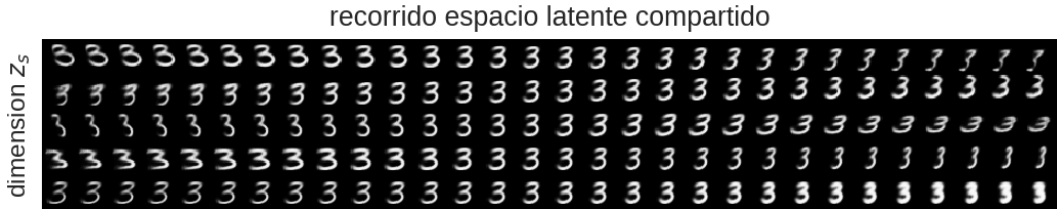


Figura D.6: Reconstrucciones para valores de cada dimension de z_s equiespaciados en el rango $[-3, 3]$. En este caso, z_s es un vector de 5 dimensiones. Activamos la tercera variable discreta de clase y fijamos z_c=0. Observamos que las variables latentes compartidas de nuestro modelo aprenden representaciones desentrelazadas. La primera dimensión codifica la inclinación del dígito. La segunda dimensión codifica si el grosor de trazo es más fuerte en la parte superior o inferior del dígito. La tercera codifica la altura. La cuarta guarda la información de la anchura del dígito y la quita la del grosor de trazo.

Comparado con el VAE expuesto en la Figura D.4, vemos que el modelo propuesto codifica menos información por dimensión, pero tiene una dimensión más. La cantidad de información total transmitida al *decoder* es aproximadamente la misma. A diferencia de los demás métodos de desentrelazado vistos en la Figura 4.5, el nuestro no limita lo informativas que son las variables latentes. Conseguimos representaciones desentrelazadas sin comprometer la calidad

de las reconstrucciones.

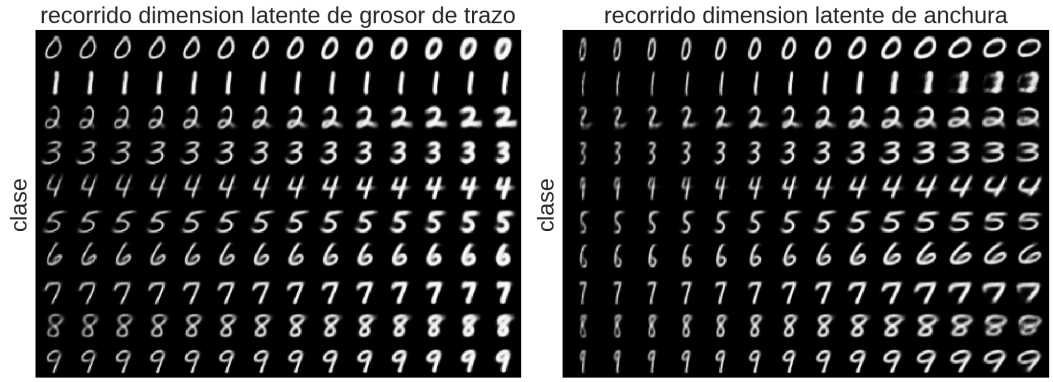


Figura D.7: Reconstrucciones generadas por el modelo propuesto en la Sección 4.5 para valores de las dimensiones de z_s correspondientes al grosor del trazo (izquierda) y la anchura del dígito (derecha) equiespaciados en el rango $[-3, 3]$. Fijamos las demás dimensiones de z_s a 0 y z_c=0.

En la Figura D.6, observamos que las dimensiones del espacio latente compartido, z_s, codifican los mismos factores que los encontrados en la Figura 4.10. La información codificada es independiente de la clase y de las variaciones intraclase. A su vez, las dimensiones de z_s codifican factores distintos, están desentrelazadas. Esto nos permite analizar datos nuevos fácilmente y generar muestras con características elegidas arbitrariamente.

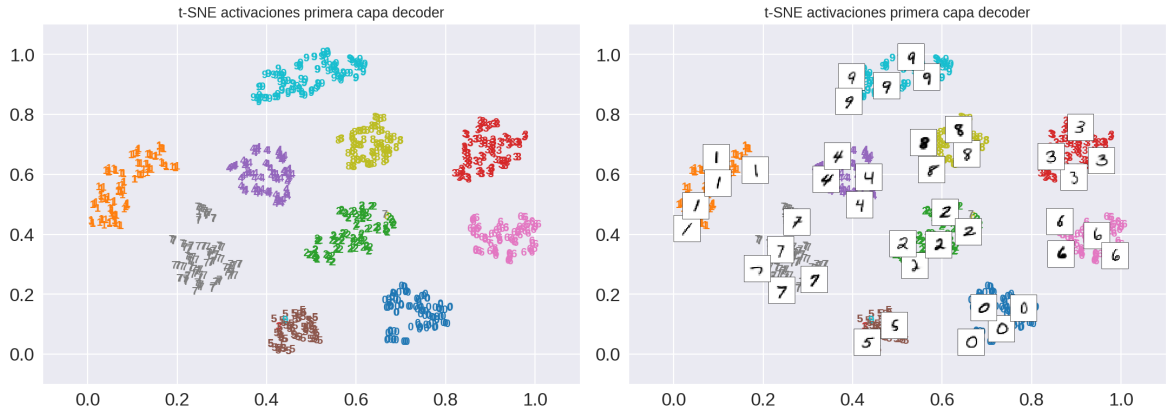


Figura D.8: Activaciones tras la primera capa del *decoder* del modelo propuesto en la Sección 4.5. En la imagen de la derecha, superponemos algunas de las reconstrucciones correspondientes a los puntos.

Elegimos dos factores de variación compartidos por todos los dígitos: el grosor del trazo y la anchura del dígito. Recorremos las dimensiones del espacio latente correspondientes a estos factores en la Figura D.7. Así, mostramos cómo podemos utilizar un modelo de variables latentes desentrelazadas para generar datos con características elegidas arbitrariamente.

En la Figura D.8, observamos cómo el *decoder* de nuestro modelo reorganiza la información de las variables latentes. Podemos interpretar que este espacio está generado por un modelo de mezcla. Es multimodal respecto a las variables discretas y continuo respecto a las demás. La información de clase determina al cluster al que pertenece un dato. La información de las variaciones intraclase y variaciones compartidas determina la posición del dato dentro del cluster.

D.5. Modelos generativos con información de clase

Para poder realizar los experimentos de la Sección 4.6, necesitamos utilizar modelos generativos en los que podamos especificar la clase del dato generado. Encontramos tres modelos en la literatura que cumplen este requisito. Representamos dígitos generados con cada uno de ellos en la Figura D.9.

Figura D.9: A la izquierda, dígitos generados con *CGAN*. En el centro, *CVAE*. A la derecha, *AC-GAN*.

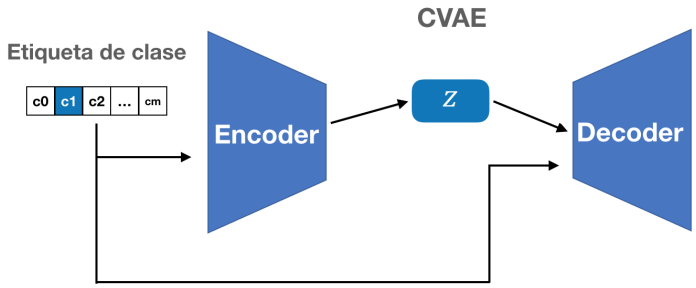


Figura D.10: Diagrama de funcionamiento de *CVAE*.

El primero es *CVAE*, propuesto en [Sohn et al., 2015]. Como podemos ver en la Figura D.10, este tiene la estructura de un autoencoder variacional estándar pero aporta información de clase tanto al encoder como al decoder. Con esto, condiciona su espacio latente entero a la clase del dígito en cuestión. Así, encontramos que la información guardada en cada dimensión de su espacio latente es distinta en función de la clase del dígito de entrada. Es un modelo que permite mucha flexibilidad a la hora de modelar datos ya que soluciona el problema de la multimodalidad. Genera reconstrucciones de mucha calidad pero genera representaciones internas muy entrelazadas. Como esto no es un problema para el entrenamiento con datos generados, es un buen candidato para nuestros experimentos.

AC-GAN condiciona al generador de una GAN con la etiqueta de clase y añade un objetivo de clasificación al discriminador. De esta manera, se minimiza el coste cuando el generador devuelve datos de la clase indicada a la vez que el discriminador los clasifica correctamente.

CGAN condiciona tanto al generador como al discriminador con la información de clase. De esta manera, el discriminador aprende, de forma implícita, a clasificar los dígitos. El generador debe producir muestras de la clase correcta para poder engañar al discriminador. Los diagramas de flujo de ambos modelos se pueden encontrar en la Figura D.11.

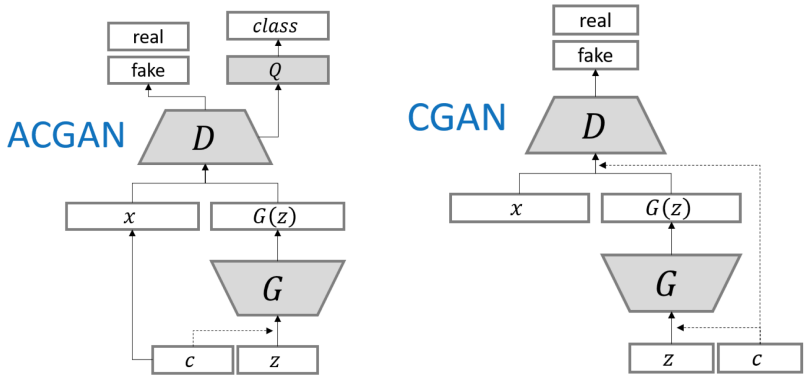


Figura D.11: A la izquierda, el diagrama de funcionamiento de *AC-GAN*. A la derecha el de *CGAN*. G se refiere a una red generadora. D se refiere a una red discriminadora. z son las variables latentes y c es la etiqueta de clase.

D.6. Experimentos con Fashion-MNIST

El conjunto de datos Fashion-MNIST [Xiao et al., 2017] contiene imágenes de distintas prendas de ropa. Hay 10 clases, una por cada tipo de prenda. Se trata de un conjunto de datos más complicado de tratar que MNIST ya que hay más variabilidad entre ejemplos de la misma clase que en MNIST pero menos entre ejemplos de clases distintas. Fashion-MNIST cuenta con 60000 ejemplos de entrenamiento y 10000 de validación.

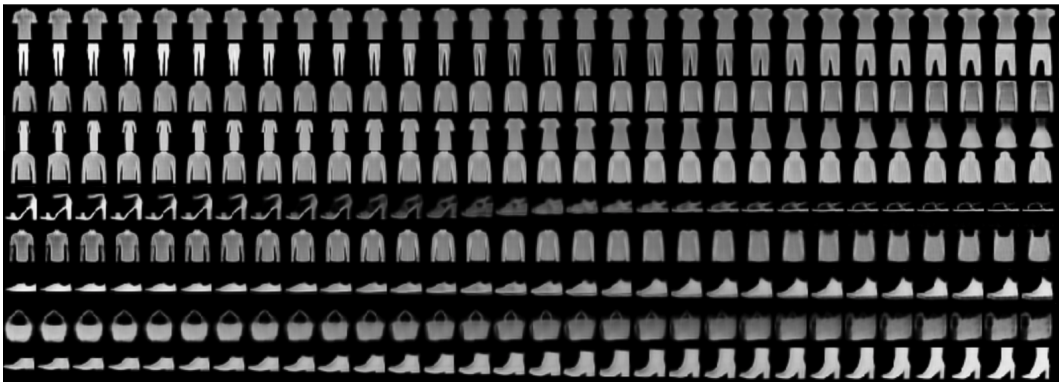


Figura D.12: Reconstrucciones de prendas generadas por el modelo propuesto en la Sección 4.5 para valores de z_c equiespaciados en el rango $[-3, 3]$. Asignamos una dimensión a cada clase. Fijamos z_s=0. Observamos que cambiando el vector indicador de clase, modificamos el tipo de prenda que se regenera. El valor de la dimensión de clase regula variaciones en la forma de la prenda reconstruida exclusivas a la clase seleccionada.

En la figura Figura D.12, mostramos reconstrucciones generadas por el modelo propuesto en la Sección 4.5 al variar las variables latentes dependientes de clase. Vemos que algunas de las clases de Fashion-MNIST exhiben una gran cantidad de variabilidad intraclase. Es el caso de las sandalias, las botas, o los vestidos representados en la sexta, décima y cuarta fila respectivamente. Por otro lado, las camisetas, las cuales se muestran en la primera fila, solo varían en la forma que distingue las de hombre de las de mujer.

En muchos casos, la diferencia entre dos prendas de ropa viene dada por una textura o patrón decorativo. Como los patrones son distintos para cada ejemplo del conjunto de datos, es difícil modelarlos con variables latentes. En la Figura D.13, mostramos algunos ejemplos

Figura D.13: En la fila superior, mostramos algunas muestras de imágenes de fashion-MNIST. En la inferior, mostramos las reconstrucciones que genera el modelo propuesto en la Sección 4.5 de dichas prendas. Observamos que el modelo capta bien la forma de las prendas pero no sus patrones y dibujos.

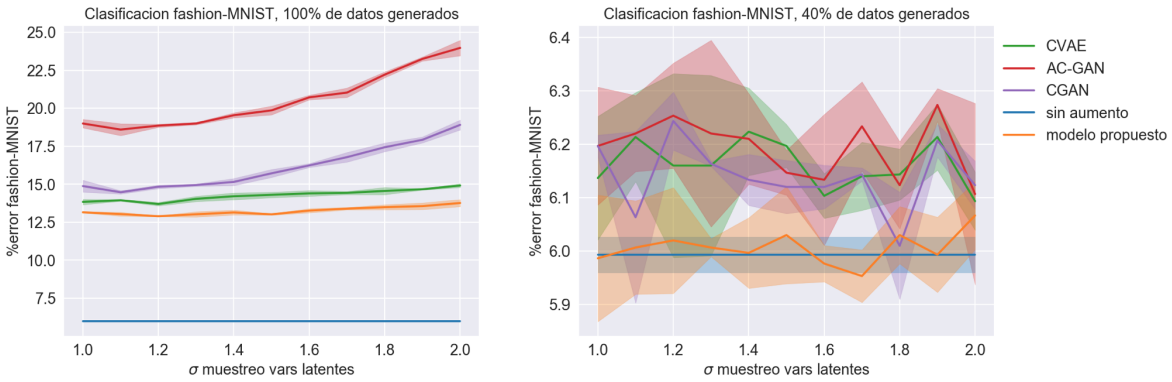


Figura D.14: Porcentaje de error en fashion-MNIST obtenido al entrenar una red clasificadora con una probabilidad del 100 % (izquierda) y del 40 % (derecha) de utilizar datos generados. Variamos σ, la desviación típica de la distribución normal de la que muestreamos las variables latentes, entre 1 y 2. Utilizamos estas muestras para generar los datos. Los trazos centrales representan las medias de los valores de error. Los trazos transparentes representan desviaciones típicas. Repetimos cada experimento tres veces.

de prendas de fashion-MNIST, con sus respectivas reconstrucciones generadas por nuestro modelo. Este, aprende bien las distintas formas de las prendas. Sin embargo, no captura los dibujos y diseños individuales de cada una. Para simular los detalles de las distintas prendas, superponemos ruido gausiano de baja energía a las prendas generadas por todos los modelos que evaluamos. En nuestros experimentos, esto mejora ligeramente los resultados a la hora de entrenar un clasificador. Empleamos la misma red clasificadora que en la Sección 4.6.

Realizamos los mismos experimentos que en la Sección 4.6. Como podemos ver en la Figura D.14, de nuevo, al aumentar los conjuntos de datos con las generaciones del modelo propuesto, obtenemos los mejores resultados. A pesar de ello, solo mejoramos marginalmente los resultados conseguidos con la base de datos original. Esto se debe a la complejidad de este conjunto de datos. En la Figura D.15, observamos que para todos los valores de $p_{generado}$, nuestro modelo da los mejores resultados.

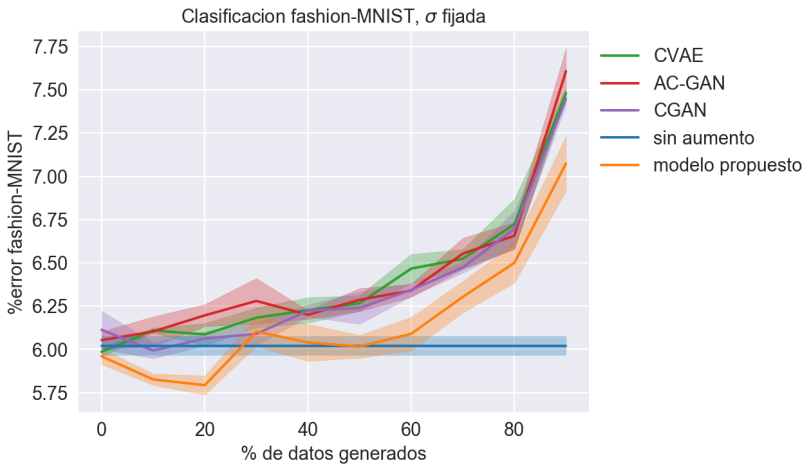


Figura D.15: Porcentaje de error en fashion-MNIST obtenido al entrenar una red clasificadora con distintas probabilidades de utilizar datos generados. Utilizamos σ=1 para las redes GAN y σ=1.4 para CVAE y para nuestro modelo propuesto. Escogemos estos valores ya que dan los mejores resultados con cada red. Los trazos centrales representan las medias de los valores de error. Los trazos transparentes representan desviaciones típicas. Repetimos cada experimento tres veces.

Bibliografía

[Aytar et al., 2016] Aytar, Y., Castrejon, L., Vondrick, C., Pirsiavash, H., and Torralba, A. (2016). Cross-modal scene networks. *CoRR*, abs/1610.09003.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Bengio, 2009] Bengio, Y. (2009). Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[Burgess et al., 2018] Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., and Lerchner, A. (2018). Understanding disentangling in β-VAE. *ArXiv e-prints*.

[Chen et al., 2018] Chen, T. Q., Li, X., Grosse, R. B., and Duvenaud, D. K. (2018). Isolating sources of disentanglement in variational autoencoders. *CoRR*, abs/1802.04942.

[Chen et al., 2016] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657.

[Dilokthanakul et al., 2016] Dilokthanakul, N., Mediano, P. A. M., Garnelo, M., Lee, M. C. H., Salimbeni, H., Arulkumaran, K., and Shanahan, M. (2016). Deep unsupervised clustering with gaussian mixture variational autoencoders. *CoRR*, abs/1611.02648.

[Dupont, 2018] Dupont, E. (2018). Learning Disentangled Joint Continuous and Discrete Representations. *ArXiv e-prints*.

[Esmaeili et al., 2018] Esmaeili, B., Wu, H., Jain, S., Siddharth, N., Paige, B., and van de Meent, J.-W. (2018). Hierarchical Disentangled Representations. *ArXiv e-prints*.

[Gal, 2016] Gal, Y. (2016). *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge.

[Gao et al., 2018] Gao, S., Brekelmans, R., Steeg, G. V., and Galstyan, A. (2018). Auto-encoding total correlation explanation. *CoRR*, abs/1802.05822.

[Gastaldi, 2017] Gastaldi, X. (2017). Shake-shake regularization. *CoRR*, abs/1705.07485.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *ArXiv e-prints*.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[He et al., 2018] He, X., Zhou, Y., Zhou, Z., Bai, S., and Bai, X. (2018). Triplet-center loss for multi-view 3d object retrieval. *CoRR*, abs/1803.06189.

[Hinton et al., 2011] Hinton, G. E., Krizhevsky, A., and Wang, S. D. (2011). Transforming auto-encoders. In Honkela, T., Duch, W., Girolami, M., and Kaski, S., editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 44–51, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Hinton et al., 2018] Hinton, G. E., Sabour, S., and Frosst, N. (2018). Matrix capsules with EM routing. In *International Conference on Learning Representations*.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[Irina Higgins, 2017] Irina Higgins, Loic Matthey, A. P. C. B. X. G. M. B. S. M. A. L. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework.

[Jang et al., 2016] Jang, E., Gu, S., and Poole, B. (2016). Categorical Reparameterization with Gumbel-Softmax. *ArXiv e-prints*.

[Jimenez Rezende and Mohamed, 2015] Jimenez Rezende, D. and Mohamed, S. (2015). Variational Inference with Normalizing Flows. *ArXiv e-prints*.

[Kim and Mnih, 2018] Kim, H. and Mnih, A. (2018). Disentangling by Factorising. *ArXiv e-prints*.

[Kingma, 2017] Kingma, D. (2017). *Variational inference & deep learning, A new synthesis*. PhD thesis, University of Amsterdam.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kingma et al., 2014] Kingma, D. P., Rezende, D. J., Mohamed, S., and Welling, M. (2014). Semi-supervised learning with deep generative models. *CoRR*, abs/1406.5298.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *ArXiv e-prints*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436 EP –.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun and Cortes, 2010] LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

[Liu et al., 2016] Liu, W., Wen, Y., Yu, Z., and Yang, M. (2016). Large-Margin Softmax Loss for Convolutional Neural Networks. *ArXiv e-prints*.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *CoRR*, abs/1411.1784.

[Odena et al., 2017] Odena, A., Olah, C., and Shlens, J. (2017). Conditional image synthesis with auxiliary classifier GANs. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2642–2651, International Convention Centre, Sydney, Australia. PMLR.

[Olah et al., 2018] Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*. https://distill.pub/2018/building-blocks.

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China. PMLR.

[Rippel et al., 2015] Rippel, O., Paluri, M., Dollar, P., and Bourdev, L. (2015). Metric Learning with Adaptive Density Discrimination. *ArXiv e-prints*.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

[Ruder, 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

[Sabour et al., 2017] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. *CoRR*, abs/1710.09829.

[Schroff et al., 2015] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832.

[Shannon, 1948] Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550:354 EP –. Article.

[Sohn et al., 2015] Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3483–3491. Curran Associates, Inc.

[van den Oord et al., 2016] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499.

[van der Maaten and Hinton, 2008] van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

[Van Der Malsburg, 1986] Van Der Malsburg, C. (1986). Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. In Palm, G. and Aertsen, A., editors, *Brain Theory*, pages 245–248, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Variani et al., 2015] Variani, E., McDermott, E., and Heigold, G. (2015). A gaussian mixture model layer jointly optimized with discriminative features within a deep neural network architecture. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4270–4274.

[Wan et al., 2018] Wan, W., Zhong, Y., Li, T., and Chen, J. (2018). Rethinking feature distribution for loss functions in image classification. *CoRR*, abs/1803.02988.

[Watanabe, 1961] Watanabe, S. (1961). A note on the formation of concept and of association by information-theoretical correlation analysis. *Information and Control*, 4(2):291 – 296.

[Weinberger et al., 2006] Weinberger, K., Blitzer, J., and K. Saul, L. (2006). Distance metric learning for large margin nearest neighbor classification. 10.

[Wen et al., 2016] Wen, Y., Zhang, K., Li, Z., and Qiao, Y. (2016). A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision*, pages 499–515. Springer.

[Xiao et al., 2017] Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747.

[Xie et al., 2016] Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. (2016). Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431.

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *CoRR*, abs/1411.1792.

[Zagoruyko and Komodakis, 2016] Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *CoRR*, abs/1605.07146.

[Zeghidour et al., 2018] Zeghidour, N., Usunier, N., Synnaeve, G., Collobert, R., and Dupoux, E. (2018). End-to-End Speech Recognition From the Raw Waveform. *ArXiv e-prints*.

[Zhang et al., 2017] Zhang, X., Zhou, X., Lin, M., and Sun, J. (2017). Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083.

[Zheng et al., 2018] Zheng, Y., Pal, D. K., and Savvides, M. (2018). Ring loss: Convex feature normalization for face recognition. *CoRR*, abs/1803.00130.

Lista de Figuras

Lista de Tablas