2. Redes bayesianas y paradigmas generativos modernos

Dada la diversidad de paradigmas y la modularidad de los modelos actuales, resulta útil realizar el estudio bajo la perspectiva unificada de los **modelos gráficos probabilísticos**, lo que permite obtener un entendimiento más holístico de la IA generativa moderna y entender más fácilmente las similitudes y diferencias que hay entre los distintos paradigmas generativos. Por otro lado, este enfoque es natural dado que todos los modelos generativos modernos suelen ser formulados y analizados en términos probabilísticos. Más aún, todos los paradigmas que se estudiarán (a excepción de los modelos basados en score) pueden ser vistos como un tipo particular de modelo gráfico denominado **red bayesiana**. En consecuencia, este capítulo comienza estudiando algunos aspectos importantes de este marco teórico, para luego revisar el problema de inferencia en redes bayesianas, el cual puede interpretarse como el proceso de entrenamiento en la jerga usual del machine learning.

2.1. Introducción a las redes bayesianas

Un modelo gráfico consiste, a grandes rasgos, en poder representar las relaciones de dependencia e independencia entre las variables de un modelo probabilístico utilizando un grafo. Esta representación visual permite entender rápidamente cómo están relacionadas las variables, ya sean visibles u ocultas, dentro de un modelo probabilístico complejo como, por ejemplo, en un modelo de Markov oculto usado para la tarea part-of-speech tagging en texto.

Dado un modelo probabilístico con distribución conjunta $p(x_1, ..., x_N)$, el modelo gráfico asociado a este modelo probabilístico es un grafo cuyos nodos representan las variables aleatorias $x_1, ..., x_N$ del modelo (hay un nodo por cada variable), mientras que la existencia de un arco entre dos nodos indica que hay una relación de dependencia (en el sentido probabilístico) entre las variables aleatorias asociadas a esos nodos.

Los modelos gráficos se pueden clasificar en dos grandes grupos, dependiendo del tipo de grafo que utilicen para representar al modelo probabilístico:

- **Red bayesiana o belief networks:** en este caso, se utiliza un grafo dirigido (i.e., cada arco tiene una dirección) para representar la relación de dependencia entre variables. En este tipo de modelos, un arco dirigido desde (el nodo) x_i hacia (el nodo) x_j indica que hay una relación de dependencia entre (la variable) x_i y (la variable) x_j. Se verá que el grafo dirigido asociado a un modelo probabilístico siempre debe ser acíclico para que esta interpretación tenga sentido.
- **Redes de Markov o Markov random fields (MRF):** en este tipo de modelos se utiliza un grafo no dirigido (i.e., no se diferencia entre un nodo origen y un nodo destino), por lo que ya no hay una jerarquía en las relaciones de dependencia. En consecuencia, la factorización de la distribución conjunta para este tipo de modelos se vuelve un poco diferente ya que se debe realizar sobre los cliques maximales del grafo, mientras que los términos que se multiplican ya no se interpretan como distribuciones, si no que se interpretan como

potenciales. Este tipo de modelos no es muy importante para la IA generativa moderna, por lo que solo se comentará brevemente al estudiar los modelos basados en energía. Es importante mencionar que varios modelos generativos clásicos como las **máquinas de Boltzmann** o las **redes de Hopfield** (Nobel de física, 2024) son modelos de este tipo.

2.1.1. Ejemplo inicial

Para introducir la idea de red bayesiana, se comenzará con un ejemplo que modela la probabilidad de aprobar o no un curso, considerando únicamente si un estudiante estudia o no para el examen, y su rendimiento en las preguntas el examen. Para esto, se considerarán 4 variables aleatorias binarias (i.e., Bernoulli), x_1, x_2, x_3 y x_4, las cuales representarán los siguientes escenarios:

- **Variable aleatoria x_1:** el estudiante estudia para el examen.
- **Variable aleatoria x_2:** el estudiante responde bien las preguntas teóricas.
- **Variable aleatoria x_3:** el estudiante responde bien las preguntas prácticas.
- **Variable aleatoria x_4:** el estudiante aprueba el curso.

Como es usual, se asociará la respuesta «sí» al valor 1 y la respuesta «no» al valor 0. El siguiente grafo dirigido representa la relación entre las 4 variables aleatorias que considera el modelo:

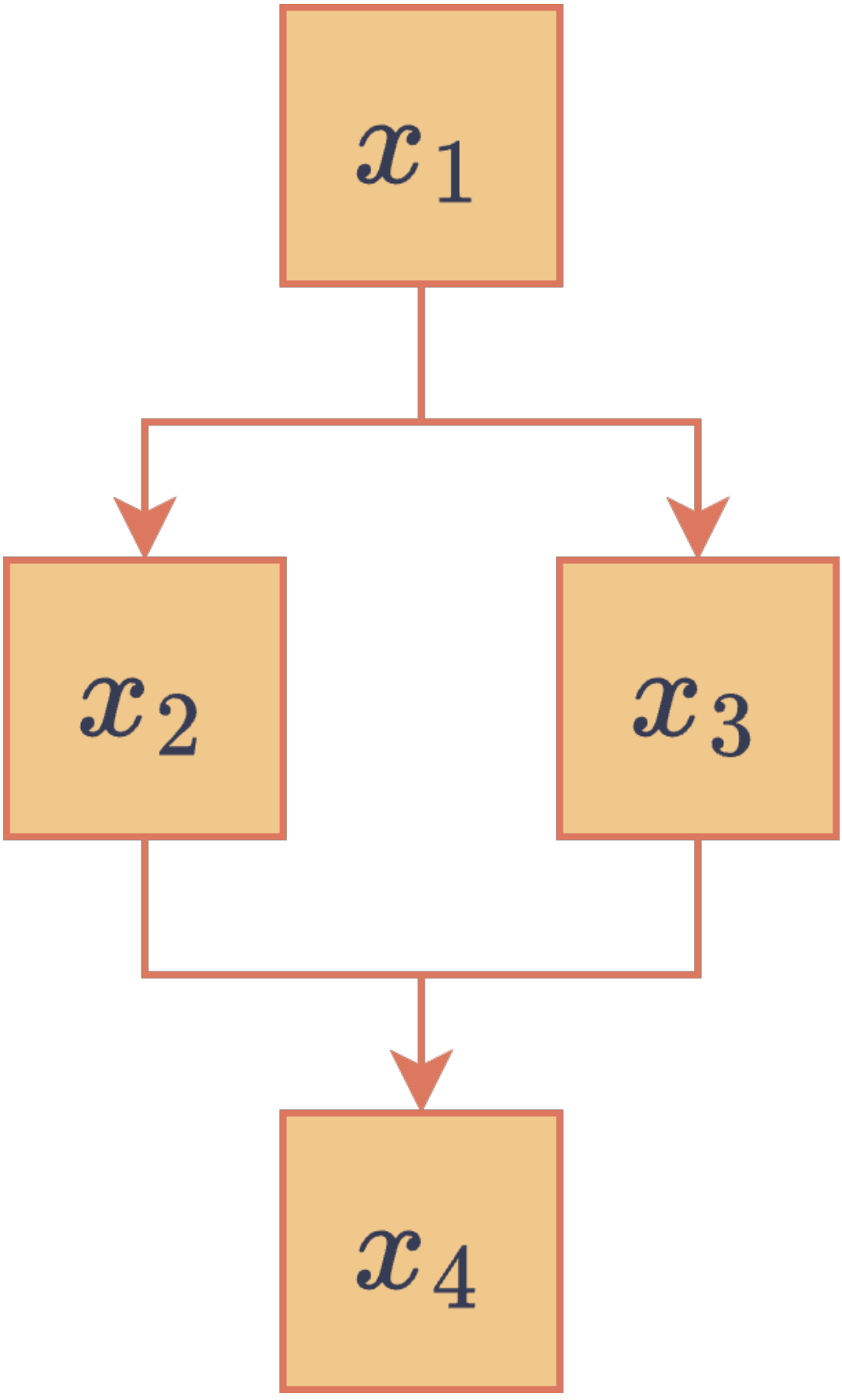

Figura 1: Red bayesiana del ejemplo del estudiante.

Este grafo indica que las variables aleatorias x_2 y x_3 dependen de x_1, mientras que la variable aleatoria x_4 depende de las variables aleatorias x_2 y x_3. Más precisamente, este grafo se interpreta factorizando la distribución conjunta de las 4 variables aleatorias como

$$p(x_1, x_2, x_3, x_4) = p(x_1)\, p(x_2 \mid x_1)\, p(x_3 \mid x_1)\, p(x_4 \mid x_2, x_3) \tag{1}$$

Es útil notar algunas independencias condicionales que induce esta factorización:

- La variable aleatoria x_3 es condicionalmente independiente, dado el valor de x_1, de la variable aleatoria x_2: $p(x_3 \mid x_1, x_2) = p(x_3 \mid x_1)$ o, equivalentemente, $p(x_2, x_3 \mid x_1) = p(x_2 \mid x_1)\, p(x_3 \mid x_1)$. Esta independencia indica que responder bien o no en las preguntas prácticas es independiente de responder bien o no en las preguntas teóricas, si es que se sabe si el estudiante estudió o no.
- La variable aleatoria x_4 es condicionalmente independiente de la variable aleatoria x_1, dados los valores de x_2 y x_3: $p(x_4 \mid x_1, x_2, x_3) = p(x_4 \mid x_2, x_3)$ o, equivalentemente, $p(x_1, x_4 \mid x_2, x_3) = p(x_1 \mid x_2, x_3)\, p(x_4 \mid x_2, x_3)$. Esta independencia indica que aprobar o no el examen no depende de si el estudiante estudió o no, si es que se sabe si respondió bien o no las preguntas teóricas y las preguntas prácticas.

Estas independencias condicionales se observan en la simplificación de la factorización de la distribución conjunta que se obtiene cuando se factoriza usando la regla de la cadena (la cual no asume ninguna independencia, solo utiliza la definición de probabilidad condicional):

$$p(x_1, x_2, x_3, x_4) = p(x_1)\, p(x_2 \mid x_1) \underbrace{p(x_3 \mid x_1, x_2)}_{p(x_3 \mid x_1)} \underbrace{p(x_4 \mid x_1, x_2, x_3)}_{p(x_4 \mid x_2, x_3)} \tag{2}$$

Notar que para definir cada una de las distribuciones condicionales, es necesario definir una distribución para cada posible valor de las variables condicionales. Por ejemplo, la distribución $p(x_4 \mid x_1, x_2, x_3)$ necesitaría definir $2 \cdot 2 \cdot 2 = 8$ distribuciones binarias sobre x_4 (una para cada posible valor de $(x_1, x_2, x_3) \in \{0, 1\}^3$), mientras que $p(x_4 \mid x_2, x_3)$ necesita definir solo $2 \cdot 2 = 4$ distribuciones binarias sobre x_4. Esto muestra cómo las hipótesis de independencia ayudan a reducir la cantidad de parámetros necesarios para definir una distribución conjunta.

Si se conocen todas las distribuciones de la factorización, es directo responder preguntas como ¿cuál es la probabilidad de responder bien las preguntas teóricas si se estudió? o ¿cuál es la probabilidad de aprobar el curso si respondió bien las preguntas prácticas y teóricas? ya que bastaría con mirar los parámetros de las distribuciones. Sin embargo, otras preguntas requieren hacer cálculos adicionales como marginalización o cálculo de posteriores. Algunas preguntas de este tipo son:

- ¿Cuál es la probabilidad de que el estudiante responda bien las preguntas teóricas?
- ¿Cuál es la probabilidad de que el estudiante haya estudiado si no respondió bien las preguntas teóricas?
- ¿Cuál es la probabilidad de responder bien tanto las preguntas teóricas como las preguntas prácticas?
- ¿Cuál es la probabilidad de responder bien las preguntas prácticas si se respondieron bien las preguntas teóricas?

Para responder las preguntas se asignarán probabilidades de juguete a cada una de las distribuciones que definen la probabilidad conjunta $p(x_1, x_2, x_3, x_4)$. Notar que la suma de las probabilidades de cada fila debe ser 1 para representar una distribución de probabilidad válida.

$p(x_1 = 0)$	$p(x_1 = 1)$
0.10	0.90

x_1	$p(x_2 = 0 \mid x_1)$	$p(x_2 = 1 \mid x_1)$
0	0.80	0.20
1	0.25	0.75

x_1	$p(x_3 = 0 \mid x_1)$	$p(x_3 = 1 \mid x_1)$
0	0.70	0.30
1	0.20	0.80

x_2	x_3	$p(x_4 = 0 \mid x_2, x_3)$	$p(x_4 = 1 \mid x_2, x_3)$
0	0	0.95	0.05
1	0	0.35	0.65
0	1	0.40	0.60
1	1	0.01	0.99

¿Cuál es la probabilidad de que el estudiante responda bien las preguntas teóricas?

Se debe calcular $p(x_2 = 1)$. Esto se puede hacer marginalizando la distribución conjunta:

$$
\begin{aligned}
p(x_2 = 1) &= \sum_{x_1, x_3, x_4 \in \{0,1\}} p(x_1, x_2 = 1, x_3, x_4) \\
&= \sum_{x_1, x_3, x_4 \in \{0,1\}} p(x_1)\, p(x_2 = 1 \mid x_1)\, p(x_3 \mid x_1)\, p(x_4 \mid x_2 = 1, x_3) \\
&= \sum_{x_1 \in \{0,1\}} \left[p(x_1)\, p(x_2 = 1 \mid x_1) \sum_{x_3 \in \{0,1\}} \left(p(x_3 \mid x_1) \sum_{x_4 \in \{0,1\}} p(x_4 \mid x_2 = 1, x_3) \right) \right] \\
&= \sum_{x_1 \in \{0,1\}} p(x_1)\, p(x_2 = 1 \mid x_1) \\
&= p(x_1 = 0)\, p(x_2 = 1 \mid x_1 = 0) + p(x_1 = 1)\, p(x_2 = 1 \mid x_1 = 1) \\
&= 0.10 \times 0.20 + 0.90 \times 0.75 \\
&= 0.695
\end{aligned}
\tag{3}
$$

Es decir, sin observar ninguna variable a priori (e.g., sin saber si el estudiante estudió o no), es cerca de un 70% probable que el estudiante responda bien las preguntas teóricas. Notar que para pasar a la cuarta igualdad se utilizó que $\sum_{k \in \text{supp}(x)} p(x = k) = 1$ para la distribución $p(x_4 \mid x_2 = 1, x_3)$ y luego para la distribución $p(x_3 \mid x_1)$.

¿Cuál es la probabilidad de que el estudiante haya estudiado si no respondió bien las preguntas teóricas?

Se debe calcular $p(x_1 = 1 \mid x_2 = 0)$. Para esto, se puede usar la definición de probabilidad condicional y luego marginalizar la distribución conjunta:

$$
\begin{aligned}
p(x_1 = 1 \mid x_2 = 0) &= \frac{p(x_1 = 1, x_2 = 0)}{p(x_2 = 0)} \\
&= \frac{1}{p(x_2 = 0)} \sum_{x_3, x_4 \in \{0,1\}} p(x_1 = 1, x_2 = 0, x_3, x_4) \\
&= \frac{1}{p(x_2 = 0)} \sum_{x_3, x_4 \in \{0,1\}} p(x_1 = 1)\, p(x_2 = 0 \mid x_1 = 1)\, p(x_3 \mid x_1 = 1)\, p(x_4 \mid x_2 = 0, x_3) \\
&= \frac{1}{p(x_2 = 0)}\, p(x_1 = 1)\, p(x_2 = 0 \mid x_1 = 1) \sum_{x_3 \in \{0,1\}} \left(p(x_3 \mid x_1 = 1) \sum_{x_4 \in \{0,1\}} p(x_4 \mid x_2 = 0, x_3) \right) \\
&= \frac{p(x_1 = 1)\, p(x_2 = 0 \mid x_1 = 1)}{p(x_2 = 0)} \\
&= \frac{0.25 \times 0.90}{0.305} \\
&\approx 0.74
\end{aligned}
\tag{4}
$$

En la penúltima igualdad se usó el resultado de la pregunta anterior para obtener $p(x_2 = 0) = 1 - p(x_2 = 1) = 0.305$.

Por otro lado, notar que esta distribución condicional no está en la factorización entregada por el grafo y, de hecho, tiene la dirección contraria al orden que induce el grafo. Por lo tanto, también se podría calcular esta probabilidad posterior utilizando la regla de Bayes:

$$
p(x_1 = 1 \mid x_2 = 0) = \frac{p(x_2 = 0 \mid x_1 = 1)\, p(x_1 = 1)}{p(x_2 = 0)} \tag{5}
$$

Llegando a la misma expresión que antes.

¿Cuál es la probabilidad de responder bien tanto las preguntas teóricas como las preguntas prácticas?

Se debe calcular $p(x_2 = 1, x_3 = 1)$. Esto se puede hacer, al igual que antes, marginalizando la distribución conjunta:

$$
\begin{aligned}
p(x_2 = 1, x_3 = 1) &= \sum_{x_1, x_4 \in \{0,1\}} p(x_1, x_2 = 1, x_3 = 1, x_4) \\
&= \sum_{x_1, x_4 \in \{0,1\}} p(x_1)\, p(x_2 = 1 \,|\, x_1)\, p(x_3 = 1 \,|\, x_1)\, p(x_4 \,|\, x_2 = 1, x_3 = 1) \\
&= \sum_{x_1 \in \{0,1\}} \left[p(x_1)\, p(x_2 = 1 \,|\, x_1)\, p(x_3 = 1 \,|\, x_1) \sum_{x_4 \in \{0,1\}} p(x_4 \,|\, x_2 = 1, x_3 = 1) \right] \\
&= \sum_{x_1 \in \{0,1\}} p(x_1)\, p(x_2 = 1 \,|\, x_1)\, p(x_3 = 1 \,|\, x_1) \\
&= p(x_1 = 0)\, p(x_2 = 1 \,|\, x_1 = 0)\, p(x_3 = 1 \,|\, x_1 = 0) + p(x_1 = 1)\, p(x_2 = 1 \,|\, x_1 = 1)\, p(x_3 = 1 \,|\, x_1 = 1) \\
&= 0.10 \times 0.20 \times 0.30 + 0.90 \times 0.75 \times 0.80 \\
&= 0.546
\end{aligned}
\tag{6}
$$

Notar que $p(x_2 = 1, x_3 = 1) \neq p(x_2 = 1)\, p(x_3 = 1)$ ya que x_2 y x_3 solo son independientes cuando se conoce el valor de x_1 (i.e., son condicionalmente independientes).

¿Cuál es la probabilidad de responder bien las preguntas prácticas si se respondieron bien las preguntas teóricas?

Se debe calcular $p(x_3 = 1 \,|\, x_2 = 1)$. Notar que $p(x_3 = 1 \,|\, x_2 = 1) \neq p(x_3 = 1)$ ya que, como se mencionó anteriormente, x_2 y x_3 solo son independientes cuando se conoce el valor de x_1. La cantidad buscada se puede obtener por definición de probabilidad condicional:

$$
\begin{aligned}
p(x_3 = 1 \,|\, x_2 = 1) &= \frac{p(x_2 = 1, x_3 = 1)}{p(x_2 = 1)} \\
&= \frac{0.546}{0.695} \\
&\approx 0.79
\end{aligned}
\tag{7}
$$

Donde se usaron los resultados de las preguntas anteriores.

Los desarrollos anteriores muestran que siempre se puede seguir el mismo procedimiento de marginalización para obtener alguna probabilidad deseada. Sin embargo, en muchos casos estos cálculos de pueden hacer de manera más eficiente utilizando la regla de Bayes cuando se busca conocer una distribución posterior.

Por otra parte, desde la perspectiva del grafo asociado al modelo probabilístico, sus nodos son $V = \{x_1, x_2, x_3, x_4\}$, mientras que sus arcos son $E = \{(x_1, x_2), (x_1, x_3), (x_2, x_4), (x_3, x_4)\} \subset V \times V$. Además, x_1 no tiene nodos padres, x_2 y x_3 tienen a x_1 como único nodo padre, y x_4 tiene a x_2 y a x_3 como nodos padres.

Notar que la estructura del grafo anterior puede ser usada para representar otros escenarios con una dinámica similar: una primera variable influye en el valor de otras dos variables condicionalmente independientes, las cuales a su vez influyen en una cuarta variable. Por otro lado, en este ejemplo se conocen las distribuciones de cada nodo (cada variable aleatoria sigue una distribución Bernoulli, cuyos parámetros se pueden deducir de las tablas), lo que permitió responder preguntas (i.e., hacer inferencia bayesiana) acerca del modelo. En los modelos generativos modernos no se conoce a priori la distribución de los datos (e.g., texto o imágenes), por lo que esta suele ser aprendida por una red neuronal utilizando muestras de entrenamiento

generadas desde la distribución desconocida. Una vez se tiene el modelo probabilístico entrenado, se pueden generar nuevas muestras a partir de él, las cuales serán similares a las muestras usadas durante el entrenamiento ya que el modelo habrá aprendido a replicar la distribución original de los datos, la cual, en un comienzo, era desconocida. De esta forma, es posible utilizar redes neuronales para generar texto, imágenes u otro tipo de dato.

2.1.2. Formulación de una red bayesiana

Para dar una definición más precisa de lo que es una red bayesiana, es importante recordar que toda la información acerca de cómo se relacionan las distintas variables aleatorias dentro de un modelo probabilístico está contenida en su distribución conjunta $p(x_1, ..., x_N)$, la cual es más informativa que el conjunto de marginales $p(x_1), ..., p(x_N)$ ya que las marginales pierden las relaciones entre las variables. Sin embargo, muchas veces los modelos suelen incluir hipótesis de independencia entre ellas para regularizar el modelo haciéndolo más simple, lo que en particular ayuda a evitar el overfitting (un modelo más débil no puede gastar capacidad en memorizar muestras de entrenamiento). Por otro lado, las hipótesis de independencia permiten disminuir la cantidad de parámetros necesarios para definir la distribución conjunta $p(x_1, ..., x_N)$. A modo de ejemplo, si las N variables $x_1, ..., x_N$ son binarias, la cantidad de combinaciones distintas entre ellas es 2^N, por lo que la distribución conjunta $p(x_1, ..., x_N)$ necesitaría una cantidad exponencial (en el número de variables) de valores para poder ser definida completamente. Con hipótesis de independencia, este costo exponencial puede ser disminuido significativamente, llegando incluso a un costo lineal en el caso extremo donde todas las variables se asumen independientes entre sí (como ocurre en los unigramas de texto). Para formular las hipótesis de independencia, es necesario recordar algunos conceptos elementales de teoría de grafo.

Un grafo dirigido es una estructura matemática compuesta por un conjunto de nodos V y por un conjunto de arcos E. Cada arco del grafo se representa como una flecha que va de un nodo a otro. Matemáticamente, un arco se suele denotar como un par ordenado de nodos, es decir, la afirmación $(u, v) \in E$ indica que el grafo posee un arco dirigido desde el nodo $u \in V$ hacia el nodo $v \in V$. En particular, esta notación indica que $E \subset V \times V$. Las siguientes definiciones son importantes para formalizar el concepto de red bayesiana:

- **Camino dirigido:** una secuencia de vértices $(w_1, ..., w_k) \in V^k$ se dice que es un camino dirigido desde $w_1 \in V$ hasta $w_k \in V$ si al recorrer los vértices de la secuencia (en el orden dado por la secuencia) se sigue el orden dado por los arcos, es decir, $(w_i, w_{i+1}) \in E$ para todo $i \in \{1, ..., k-1\}$.
- **Grafo dirigido acíclico (DAG):** un grafo dirigido se dice acíclico si para todo nodo $v \in V$, no es posible encontrar un camino dirigido que empiece y termine en v. Es decir, no se pueden formar caminos dirigidos cerrados (este tipo de caminos se llama ciclo).
- **Nodos padres:** los DAG inducen una jerarquía sobre el conjunto de los nodos del grafo. Para un nodo $v \in V$, sus nodos padres son los nodos que están directamente conectados a él, es decir, $\mathrm{Pa}(v) := \{u \in V : (u, v) \in E\}$.
- **Orden topológico:** los DAG inducen un orden (parcial) sobre V a partir de la jerarquía impuesta por la orientación de los arcos. Este orden se denomina orden topológico y siempre se asumirá, sin pérdida de generalidad, que los vértices de un DAG están enumerados de acuerdo al orden topológico del grafo, es decir, $V = \{x_1, ..., x_N\}$, donde $N = |V|$ es la cantidad de nodos, y $x_j \notin \mathrm{Pa}(x_i)$ si $j > i$.

Dado un conjunto de variables aleatorias, $x_1, ..., x_N$, la idea general de una red bayesiana es poder representar la factorización de su distribución conjunta $p(x_1, ..., x_N)$ como un DAG (V, E), donde sus nodos son las variables aleatorias del modelo probabilístico (i.e., $V = \{x_1, ..., x_N\}$), mientras que la presencia de un arco $(x_i, x_j) \in E$ dirigido desde el nodo $x_i \in V$ hacia el nodo $x_j \in V$ indica que la variable x_i influye directamente en el valor de la variable x_j (o que x_j depende directamente de x_i). Más precisamente, la noción de independencia que induce un DAG, denominada **propiedad de Markov**, es $p(x_n \,|\, x_1, ..., x_{n-1}) = p(x_n \,|\, \text{Pa}(x_n))$, donde $\text{Pa}(x_n) \subset \{x_1, ..., x_{n-1}\}$. Esta noción de independencia indica que $x_i \perp x_{\text{pred}(x_i) \setminus \text{Pa}(x_i)} \,|\, x_{\text{Pa}(x_i)}$, es decir, todo nodo es independiente de sus ancestros (nodos hacia arriba en la jerarquía) si se conoce el valor de sus nodos padres. De esta forma, sustituyendo las propiedades de independencia en la factorización que entrega la regla de la cadena, $p(x_1, ..., x_N) = \prod_{n=1}^{N} p(x_n \,|\, x_1, ..., x_{n-1})$, la distribución conjunta $p(x_1, ..., x_N)$ del modelo probabilístico asociado al DAG (V, E) se factoriza como

$$p(x_1, ..., x_N) = \prod_{n=1}^{N} p(x_n \,|\, \text{Pa}(x_n)) \tag{8}$$

Notar que los nodos raíces del grafo (nodos x_n donde $\text{Pa}(x_n) = \emptyset$) definen probabilidades incondicionales $p_\theta(x_n \,|\, \text{Pa}(x_n)) = p_\theta(x_n)$, por lo que (las distribuciones de) estas variables se suelen llamar priors, mientras que las variables donde $\text{Pa}(x_n) \neq \emptyset$ interactúan de forma condicional con sus nodos padres, permitiendo muchas veces obtener un modelo intuitivo e interpretable. Por otro lado, entendiendo $(x_i, x_j) \in E$ como que x_i causa x_j (en un sentido informal), no tendría sentido que también $(x_j, x_i) \in E$ (i.e., que x_j cause x_i). Esta observación permite entender por qué se necesita que el grafo (V, E) sea acíclico para poder ser interpretado como una red bayesiana.

En el caso de trabajar únicamente con variables discretas (e.g., categóricas), los parámetros de las distribuciones condicionales $p(x_n \,|\, \text{Pa}(x_n))$ se pueden almacenar en tablas de forma similar a las tablas construidas en el ejercicio inicial. Sin embargo, los modelos generativos modernos requieren aprender distribuciones mucho más complejas, usualmente continuas, por lo que se suele utilizar redes neuronales que se sabe que funcionan bien en tareas que requieren capturar patrones en alta dimensión. Por lo general, cada factor $p(x_n \,|\, \text{Pa}(x_n))$ de una red bayesiana suele seguir una distribución típica, donde sus parámetros (de la distribución, no de la red neuronal) son aprendidos por una red neuronal cuya entrada son los valores de las variables condicionantes (i.e., los valores de las variables en $\text{Pa}(x_n)$), mientras que la salida son los parámetros que requiere la distribución $p(x_n \,|\, \text{Pa}(x_n))$ para quedar totalmente definida. A modo de ejemplo:

- Si $x = (x_1, ..., x_N) \in \{0, 1\}^N$ es una imagen (monocromática), donde $N = \text{ancho} \times \text{alto}$, y x_n es el n-ésimo pixel de la imagen (cuyo valor es 0 o 1 por simplicidad), entonces cada pixel x_n puede ser considerado como una distribución Bernoulli, $x_n \sim \text{Bernoulli}(r_n)$, donde $r_n \in [0, 1]$ es el parámetro de la distribución, cuyo valor depende de los pixeles en $\text{Pa}(x_n)$. Este parámetro suele ser determinado por una red neuronal (con salida sigmoidal para estar en el intervalo $[0, 1]$), cuya entrada son los valores de los pixeles vecinos $\text{Pa}(x_n)$ que el modelo considere. Por ejemplo, una capa convolucional considera como $\text{Pa}(x_n)$ únicamente a los pixeles que están dentro del campo receptivo de la convolución.
- Si $x = (x_1, ..., x_N) \in \mathbb{R}^N$ es una variable continua (y cada componente x_n es un escalar irrestricto), es usual considerar $p(x_n \,|\, \text{Pa}(x_n)) \sim \mathcal{N}(\mu_n, \sigma_n^2)$, donde los parámetros $\mu_n \in \mathbb{R}$

y $\sigma_n^2 > 0$ dependen de los valores de las variables en $\text{Pa}(x_n)$. Al igual que antes, estos parámetros suelen estar determinados por redes neuronales cuyas entradas son los valores de las variables en $\text{Pa}(x_n)$.

Notar que en todos los ejemplos se ha considerado que cada x_n es una variable aleatoria unidimensional. En general, cada variable x_n por sí sola puede ser un vector aleatorio de varias dimensiones, por lo que no se diferenciará, como es usual en machine learning, entre variable aleatoria y vector aleatorio. Más aún, cada nodo $x_n \in V$ puede estar representando a un conjunto de variables aleatorias que, por sus propiedades de independencia, pueden agruparse en un mismo nodo.

Por otro lado, es importante mencionar que las redes bayesianas no se llaman así debido a que asumen la interpretación bayesiana. El nombre se debe a que utilizan la regla de Bayes para hacer inferencia sobre las variables desconocidas.

2.1.2.1. Generación de muestras

Dada una red bayesiana $p(x_1, ..., x_N) = \prod_{n=1}^{N} p(x_n \mid \text{Pa}(x_n))$, donde las distribuciones $p(x_n \mid \text{Pa}(x_n))$ son todas conocidas (y con parámetros conocidos o aprendidos con una red neuronal), entonces el algoritmo de **ancestral sampling** permite generar una nueva muestra desde la distribución conjunta $p(x_1, ..., x_N)$. Si bien esto no siempre es necesario (e.g., en el ejemplo inicial no es claro para qué sería útil generar muestras a partir de esa red bayesiana), en IA generativa el sampling es la tarea más importante luego del entrenamiento ya que permite, por ejemplo, generar nuevas respuestas si se tiene un LLM ya entrenado o generar una nueva imagen si se tiene un modelo de difusión ya entrenado.

El algoritmo de ancestral sampling comienza, naturalmente, generando muestras desde los nodos raíces (que funcionan como semillas) y luego utiliza estas muestras para generar nuevas muestras de los nodos hijos. Este procedimiento se repite jerárquicamente (i.e., con el orden de generación siguiendo el orden topológico del grafo) hasta llegar al último nodo, obteniendo así una muestra desde todos los nodos del grafo. La muestra conjunta resultante de este procedimiento, $(x_1, ..., x_N)$, es una muestra generada desde la distribución conjunta $p(x_1, ..., x_n)$.

Es importante destacar que el algoritmo de ancestral sampling asume que se sabe cómo generar nuevas muestras desde cada factor $p(x_n \mid \text{Pa}(x_n))$ de la red bayesiana. Este es otro motivo por el cual los modelos gráficos suelen elegir distribuciones condicionales simples (e.g., categóricas o gaussianas) en sus formulaciones.

En los paradigmas generativos que se revisarán, el sampling será realizado casi siempre usando ancestral sampling: en modelos de variable latente como las GANs o los VAEs, se comienza generando una muestra inicial desde $z \sim p(z)$ y luego se utiliza esta muestra para generar desde $p(x \mid z)$ usando la red neuronal entrenada (llamada generador en el caso de la GAN y decoder en el caso del VAE). En modelos secuenciales como los ARMs o los DMs, se comienza con una muestra inicial desde el nodo raíz (token inicial en ARMs y ruido inicial en DMs) y luego se decodifica iterativamente de forma secuencial (de forma causal en los ARMs y de forma anticausal en los DMs). En los modelos basados en score y EBMs en general (que no son redes bayesianas), dado que solo se suele conocer una cantidad proporcional a $p(x)$, la generación no es realizada con ancestral sampling, si no que se usan técnicas de Markov chain Monte Carlo (MCMC) ya que este tipo de algoritmos no necesita conocer la constante de normalización.

2.1.3. Algunas redes bayesianas usuales

La generalidad de las redes bayesianas permite ver varios modelos clásicos de machine learning, que usualmente se enseñan de manera independiente, como casos particulares de redes bayesianas.

2.1.3.1. Mixturas

Un modelo de mezclas (mixtura) consiste en combinar N distribuciones independientes, $p_1(x), ..., p_N(x)$, usando una variable latente $p(z) \sim \text{Categórica}(\{1, ..., N\})$ que elige al azar una de las distribuciones para generar la muestra observable x. Más precisamente, la distribución marginal asociada a la variable observable $p(x)$, es una combinación convexa de las distribuciones que se buscan mezclar:

$$p(x) = \sum_{n=1}^{N} p(z = n)\, p_n(x) \tag{9}$$

Notar que esta distribución marginal se obtiene al marginalizar x en el modelo de variable latente estándar, $p(x, z) = p(z)\, p(x \mid z)$, donde $p(z)$ es una distribución categórica y $p(x \mid z = n) = p_n(x)$. En consecuencia, el DAG asociado a una mixtura es el siguiente:

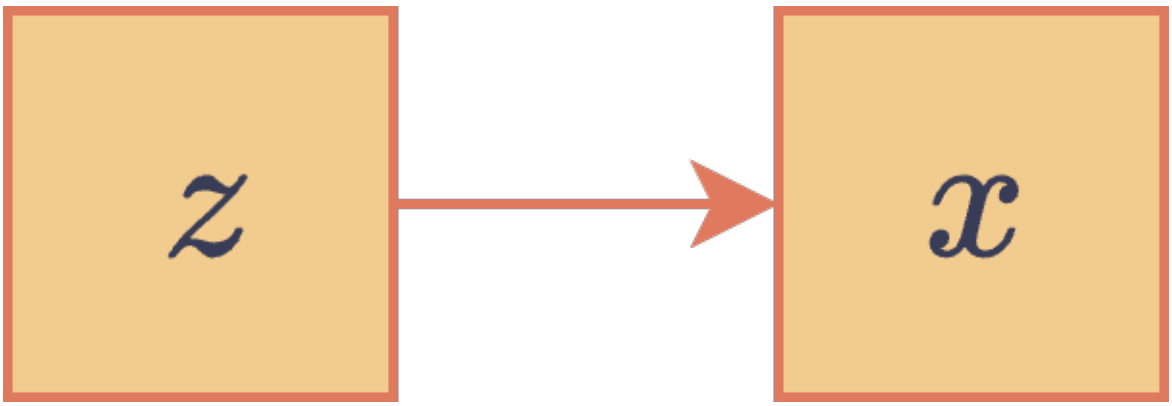

Figura 2: DAG asociado a una mixtura.

Las mixturas más usuales, llamadas **mixturas gaussianas**, son las que consideran $p_n(x) \sim \mathcal{N}(\mu_n, \Sigma_n)$. Por otro lado, es útil notar que los modelos de variable latente más generales, donde z es una variable latente continua, pueden ser vistos como una mixtura infinita de distribuciones.

2.1.3.2. Naïve Bayes

Naïve Bayes es un clasificador clásico que, a diferencia de otros clasificadores que aprenden directamente una distribución $p(y \mid x)$ (como la regresión logística o los clasificadores usuales basados en redes neuronales), utiliza un enfoque generativo (i.e., aprende una distribución conjunta $p(x, y)$ en vez de una distribución discriminativa $p(y \mid x)$). Este clasificador factoriza la distribución conjunta de una muestra $x \in \mathbb{R}^D$ y su respectiva clase, $y \in \{1, ..., C\}$, como

$$p(x, y) = p(y) \prod_{d=1}^{D} p(x_d \,|\, y) \tag{10}$$

Es decir, esta red bayesiana asume que $x_i \perp x_j \,|\, y$: las características (coordenadas) de una muestra $x \in \mathbb{R}^D$ son condicionalmente independientes si es que se conoce la etiqueta de clase $y \in \{1, ..., C\}$ (de aquí viene la parte ingenua o naïve). El DAG asociado a esta red bayesiana es el siguiente:

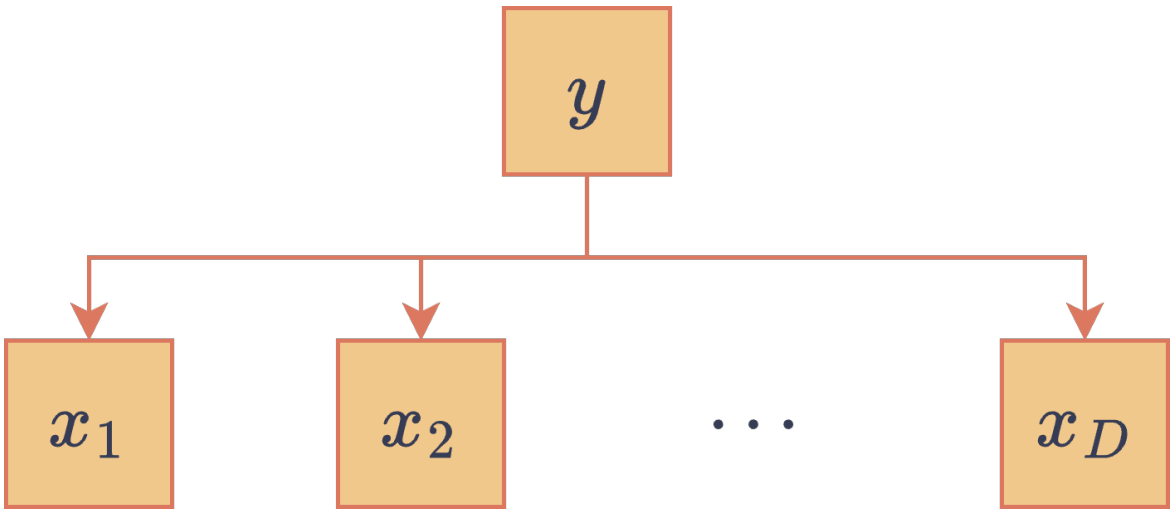

Figura 3: DAG asociado al clasificador Naïve Bayes.

Notar que en este DAG se separó la variable $x \in \mathbb{R}^D$ en sus componentes para poder separar las relaciones de independencia de cada coordenada. En otros grafos es usual agrupar todo el vector aleatorio $x \in \mathbb{R}^D$ en un mismo nodo cuando este puede ser visto como un mismo objeto en el sentido de las relaciones de dependencia e independencia.

En esta red bayesiana $p(y)$ es una distribución categórica (ya que y es una etiqueta de clase), mientras que cada distribución $p(x_d \,|\, y)$ dependerá de la naturaleza de la variable aleatoria x_d. Para realizar clasificación, este modelo calcula la posterior $p(y \,|\, x)$ mediante la regla de Bayes y elige la clase más probable.

2.1.3.3. Cadenas de Markov

Cuando las variables del modelo, $(x_1, ..., x_T)$, tienen un comportamiento o interpretación secuencial, es usual considerar ciertas hipótesis de causalidad o independencia temporal. Una familia de modelos simples pero muy útiles en la práctica son las **cadenas de Markov** (de primer orden), las cuales asumen que el futuro es independiente del pasado si se conoce el presente, es decir: $p(x_{t+1} \,|\, x_1, ..., x_t) = p(x_{t+1} \,|\, x_t)$. En consecuencia, este tipo de modelos factoriza la distribución conjunta como

$$p(x) = p(x_1) \prod_{t=1}^{T-1} p(x_{t+1} \,|\, x_t) \tag{11}$$

Dada la forma en la que se van generando las muestras según el algoritmo de ancestral sampling, a este tipo de modelos también se les dice **autorregresivos de primer orden**. Una elección

usual, y que muchas veces es asumida sin siquiera mencionarlo, es considerar que la cadena de Markov es **homogénea**, es decir, $p(x_{t+1} \mid x_t)$ es independiente de t para todo $t \in \{1, ..., T-1\}$ (lo que puede verse como una forma de parameter tying). En este caso, si además cada variable $(x_1, ..., x_T)$ es discreta, el kernel de transición $p(x_{t+1} \mid x_t)$ puede ser guardado en una matriz estocástica, llamada **matriz de transición**: $P_{ij} = p(x_2 = j \mid x_1 = i) = p(x_{t+1} = j \mid x_t = i)$, para todo $t \in \{1, ..., T-1\}$. El DAG asociado a una cadena de Markov es el siguiente:

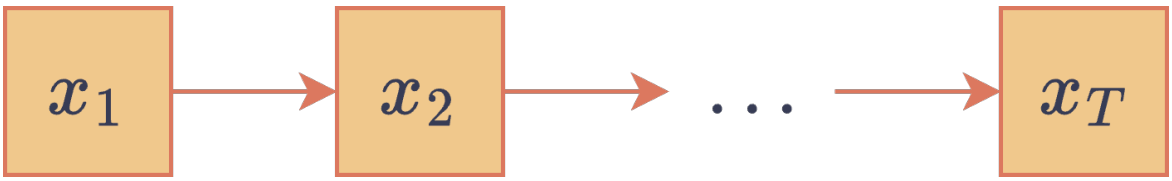

Figura 4: DAG asociado a una cadena de Markov.

La condición de Markov se puede generalizar a un orden $k \in \mathbb{N}$, donde cada variable depende de las k variables anteriores, lo cual se representa con la hipótesis de independencia $p(x_{t+1} \mid x_1, ..., x_t) = p(x_{t+1} \mid x_{t-k+1}, ..., x_t)$. En particular, si se están modelando secuencias de texto, cuando $k = 1$ al modelo se le suele llamar **bigrama** y para $k = 2$ se le llama **trigrama**. En general, se les llama $(k+1)$-grama a los modelos de Markov de orden k que modelan secuencias de texto utilizando $k+1$ gramas (unidad básica de texto, como una letra o una palabra). En el caso extremo $k = 0$, donde todos los tokens son independientes (pensar, por ahora, un token como una palabra dentro de un texto), el modelo se llama **unigrama** o **bag-of-words** (BoW). El otro caso extremo, $k = T$, corresponde a no realizar ninguna hipótesis de independencia, recuperando la regla de la cadena.

Por otro lado, los procesos forward y backward asociados a los modelos de difusión son cadenas de Markov, al igual que los flujos normalizantes. En general, la forma relativamente simple que tienen las cadenas de Markov para modelar procesos secuenciales permite poder usarlas en una amplia cantidad de casos, por lo que este tipo de procesos suelen ser estudiados de forma especializada.

2.1.3.4. Modelos de Markov ocultos

Los modelos de Markov ocultos (HMM) son, al igual que las cadenas de Markov, redes bayesianas útiles para modelar secuencias de variables $x = (x_1, ..., x_T)$. La principal diferencia es que los HMM asumen la existencia de una cadena de Markov latente, $z = (z_1, ..., z_T)$, la cual va generando las observaciones visibles $x = (x_1, ..., x_T)$, donde cada elemento x_t generado depende únicamente de la variable latente z_t. Es decir:

$$p(z, x) = p(z_1) \prod_{t=1}^{T-1} p(z_{t+1} \mid z_t) \prod_{t=1}^{T} p(x_t \mid z_t) \tag{12}$$

En esta factorización, $p(z_1)$ es la distribución inicial sobre los estados latentes, $p(z_t \mid z_{t-1})$ es el kernel de transición entre los estados latentes y $p(x_t \mid z_t)$ es la **distribución de emisión**, la cual modela cómo cada estado latente genera su respectiva observación. El DAG asociado a un HMM es el siguiente:

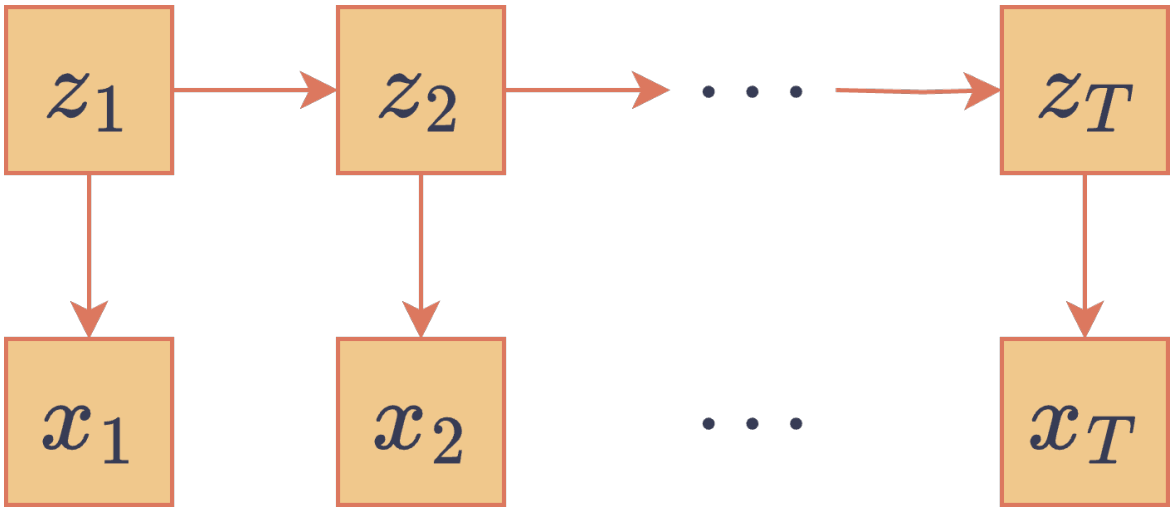

Figura 5: DAG asociado a un modelo de Markov oculto.

Si bien los HMMs se pueden usar en diversos campos, en NLP es usual ver este tipo de modelos en tareas de etiquetado de secuencias de texto como POS tagging (busca etiquetar cada palabra con su categoría gramatical como sustantivo o verbo) y NER (busca identificar palabras que son entidades como personas o lugares), donde es necesario asignarle una clase a cada uno de los tokens del texto a analizar. Por otro lado, si bien este tipo de modelos podría usarse para generación de texto, la independencia condicional entre los tokens $x_1, ..., x_T$ no permitiría generar frases con mucho sentido.

2.2. Modelos generativos modernos

2.2.1. Formulación naïve

Muchos de los modelos generativos actuales están basados en redes neuronales entrenadas para aprender a aproximar una distribución desconocida $\mathrm{p}_{\mathrm{data}}(x)$ utilizando únicamente un conjunto de entrenamiento, $\mathcal{D} = \{x^1, ..., x^K\}$, donde (se asume que) cada muestra $x^k \in \mathbb{R}^D$ fue generada de manera independiente a partir de $\mathrm{p}_{\mathrm{data}}(x)$. Una forma directa de hacer esto es considerar la **distribución empírica**, la cual le asigna una masa uniforme únicamente a las muestras observadas en el entrenamiento. Es decir, la función de masa de la distribución empírica es

$$p_{\mathcal{D}}(x) = \frac{1}{K} \sum_{k=1}^{K} \delta_{x^k}(x), \tag{13}$$

donde $\delta_{x^k}(x) := \begin{cases} 1 \text{ si } x = x^k \\ 0 \text{ si no} \end{cases}$ es la **medida de Dirac** centrada en la muestra $x^k \in \mathbb{R}^D$.

Sin embargo, esta distribución es poco útil para ser usada en un proceso generativo ya que solo asigna masa a las muestras observadas en el conjunto de entrenamiento $\mathcal{D}$, lo que no permite variabilidad en la generación más allá de este conjunto. Una posible regularización a este enfoque consiste en utilizar **kernel density estimation** (KDE), lo cual puede verse como una versión suavizada de la distribución empírica. Si $\mathcal{K} : \mathbb{R} \to \mathbb{R}_+$ es un **kernel de densidad** (i.e., una

función de densidad con $\mathbb{E}_{x\sim\mathcal{K}}[x] = 0$), entonces la distribución que entrega KDE es un promedio de estos kernels centrados en cada una de las muestras en $\mathcal{D}$:

$$\text{KDE}(x) = \frac{1}{K}\sum_{k=1}^{K}\mathcal{K}\left(x - x^k\right) \tag{14}$$

La función $x \mapsto \mathcal{K}\left(x - x^k\right)$ suele interpretarse como una versión suave de la medida de Dirac, $x \mapsto \delta_{x^k}(x)$, debido a que $\mathcal{K}\left(x - x^k\right)$ suele entregar masa a los vecinos de x^k (generalmente decayendo a medida que se alejan de x^k), mientras que $\delta_{x^k}(x)$ concentra toda su masa en x^k.

En las siguientes figuras se pueden ver ejemplos de KDE para $D = 1$ (izquierda) y $D = 2$ (derecha). En ambos casos, $\mathcal{D}$ está formado por 5 puntos generados al azar sobre el intervalo $[0, 1]$ (izquierda) y sobre el cuadrado $[0, 1] \times [0, 1]$ (derecha). El kernel usado en ambos casos es un kernel gaussiano, $\mathcal{K}(x) = \frac{1}{\sqrt{2\pi h^2}}\exp\left(-\frac{\|x\|^2}{2h^2}\right)$ (con $h > 0$ un hiperparámetro), el cual permite colocar una campana gaussiana en cada uno de los puntos de $\mathcal{D}$, de forma similar a como ocurre en una mixtura gaussiana con prior uniforme. Dado que el kernel depende únicamente de la cantidad $r = \| x \|$, este kernel también se suele llamar **kernel de base radial** (RBF kernel) ya que, como se observa en el mapa de calor, genera curvas de nivel circulares.

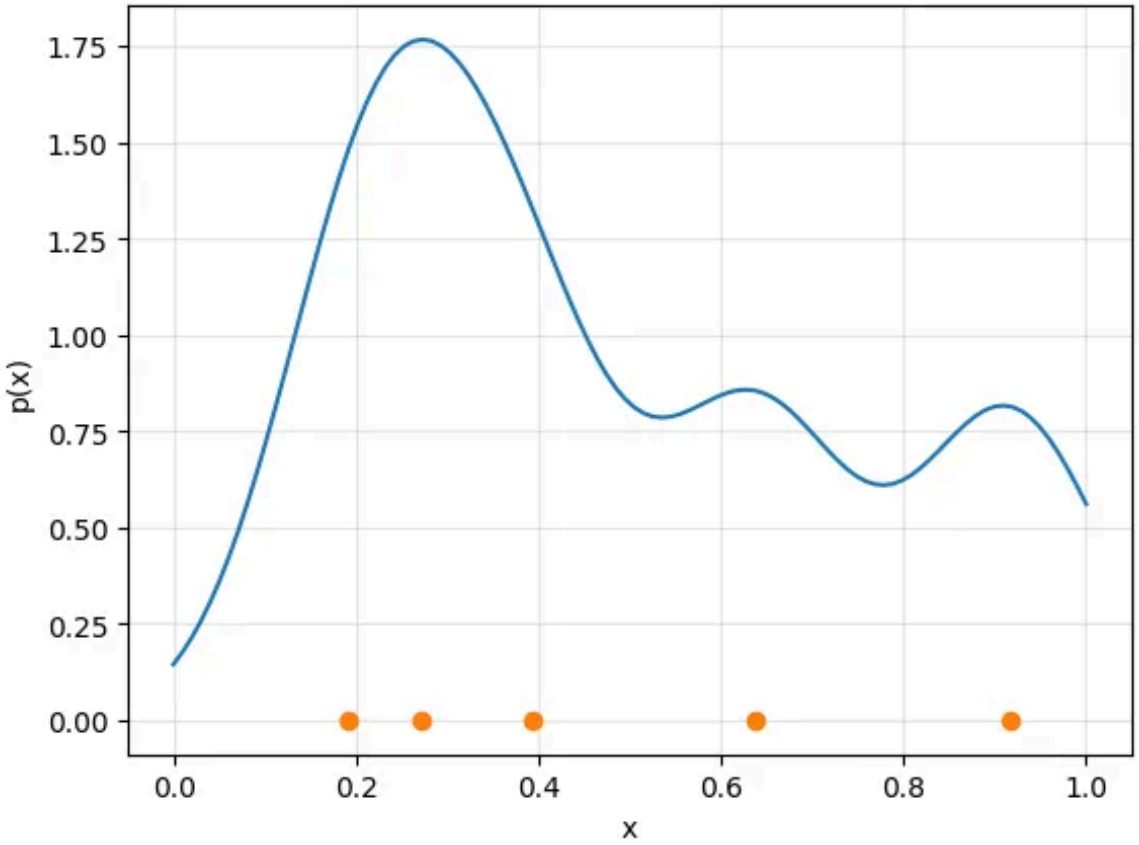

Figura 6: KDE en $D = 1$.

- Código para generar la figura.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
def gaussian_kernel(x, xi, bandwidth):
    return np.exp(-0.5 * ((x - xi) / bandwidth) ** 2) / (np.sqrt(2 * np.pi) *
bandwidth)

def kde(data, x_grid, bandwidth=0.1):
    return np.mean([gaussian_kernel(x_grid, xi, bandwidth) for xi in data],
axis=0)

data = np.random.rand(5)

x = np.linspace(0, 1, 100)
density = kde(data, x)

plt.plot(x, density)
plt.plot(data, np.zeros_like(data), 'o')
plt.xlabel('x')
plt.ylabel('p(x)')
plt.show()
```

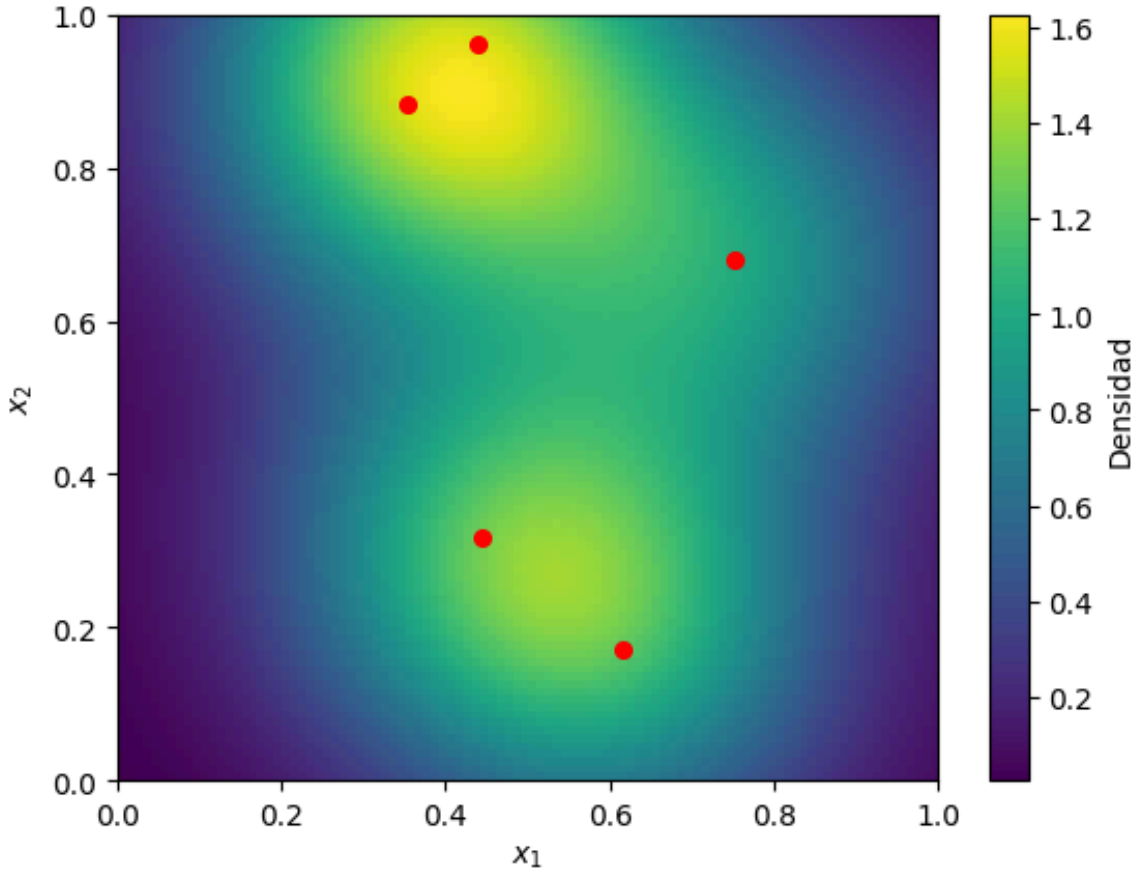

Figura 7: KDE en $D = 2$.

- Código para generar la figura.

```
import numpy as np
import matplotlib.pyplot as plt

def gaussian_kernel(x, y, xi, yi, bandwidth):
```

```
    return np.exp(-0.5 * (((x - xi) ** 2 + (y - yi) ** 2) / bandwidth ** 2)) / (2
* np.pi * bandwidth ** 2)

def kde(data, x_grid, y_grid, bandwidth=0.2):
    density = np.zeros_like(x_grid, dtype=float)
    for xi, yi in zip(data[0], data[1]):
        density += gaussian_kernel(x_grid, y_grid, xi, yi, bandwidth)
    return density / len(data[0])

data = np.random.rand(2, 5)

x_grid, y_grid = np.meshgrid(np.linspace(0, 1, 100), np.linspace(0, 1, 100))
density = kde(data, x_grid, y_grid)

plt.imshow(density, origin='lower', extent=[0, 1, 0, 1], cmap='viridis',
aspect='auto')
plt.colorbar(label='Densidad')
plt.scatter(data[0], data[1], c='red', s=30)
plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.show()
```

En ambos casos, el hiperparámetro $h > 0$ (asociado a la desviación estándar en una distribución gaussiana) determina a qué velocidad decae la densidad en $\mathbb{R}^D$ a medida que las posibles muestras se alejan de las muestras observadas en $\mathcal{D}$: un valor h pequeño puede capturar con precisión variaciones locales, pero puede producir overfitting al poder capturar también el ruido en las muestras, mientras que un h más grande, si bien genera una densidad más suave, pero puede producir underfitting al limitarse a distribuciones de baja frecuencia.

2.2.2. Paradigmas generativos actuales

Si bien KDE puede funcionar bien en baja dimensión, sufre de la maldición de la dimensionalidad, por lo que no se puede utilizar en contextos como generación de texto o imágenes. Más aún, no es claro cómo utilizar este enfoque para tareas más complejas como edición de imágenes o chatbots. En consecuencia, hay que buscar enfoques más robustos, como aquellos basados en verosimilitud o que se haya probado que, al menos en la práctica, funcionan bien. Dentro de esta familia de modelos, se encuentran los 6 paradigmas mencionados en la introducción. A excepción de los modelos basados en energía, todos estos paradigmas pueden verse como un tipo específico de red bayesiana:

2.2.2.1. Modelos autorregresivos (ARMs)

Este tipo de modelos se suele utilizar para modelar la generación de secuencias temporales. Si $x = (x_1, ..., x_T)$ es una secuencia de variables (se asumirá T fijo por simplicidad, pensado como un largo de secuencia máximo), la red bayesiana asociada a un ARM corresponde a factorizar la distribución conjunta siguiendo la regla de la cadena y no asumir, en principio, ninguna independencia (aunque impone como orden topológico del grafo al orden temporal de las variables):

$$p(x) = p(x_1) \prod_{t=1}^{T-1} p(x_{t+1} \mid x_1, ..., x_t) \tag{15}$$

El DAG que representa esta red bayesiana es el siguiente:

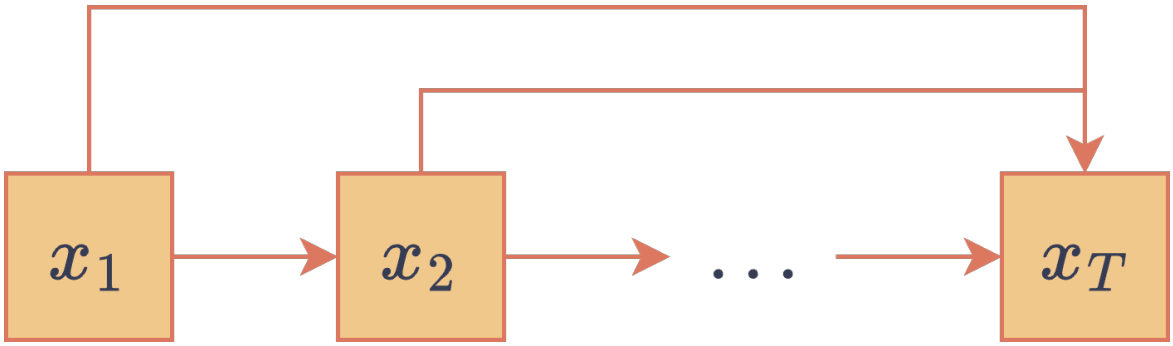

Figura 8: DAG asociado a un modelo autorregresivo.

Hasta hace no mucho tiempo, las distribuciones condicionales $p(x_{t+1} \mid x_1, ..., x_t)$ solían ser aprendidas usando redes neuronales recurrentes como la LSTM o la GRU. Sin embargo, hoy en día es más usual el uso de arquitecturas tipo Transformer debido a sus ventajas de paralelización y memoria temporal. Por otro lado, el hecho de no tener variables ocultas permite entrenar estos modelos usando el criterio de máxima verosimilitud, lo cual deja de ser posible en los siguientes paradigmas (al menos de forma exacta).

2.2.2.2. Autoencoders variacionales (VAEs)

Esta familia de modelos formula el proceso generativo como un modelo de variable latente, $p(x, z) = p(z)\, p(x \mid z)$, con la particularidad de que al mismo tiempo también aprenden otro modelo $q(z \mid x)$ que es entrenado para estimar la distribución posterior $p(z \mid x)$. Si bien estos modelos no pueden ser optimizados usando el criterio de máxima verosimilitud (debido a que se vuelve intratable en modelos complejos), el uso de estos dos modelos permite encontrar una función de costo robusta, llamada ELBO, la cual sí se puede calcular de manera eficiente para el entrenamiento. Los DAGs de un VAE son los siguientes:

Figura 9: DAGs asociados a un VAE.

Los parámetros de la distribución condicional $p(x \mid z)$ (que genera una muestra x a partir de un valor de la variable latente z) suelen ser aprendidos por una red neuronal, llamada **decoder**, la cual recibe como entrada el valor de la variable latente $z \in \mathbb{R}^L$. Por otro lado, los parámetros de la distribución condicional $q(z \mid x)$ (que recupera la variable latente z a partir de una muestra observada x) suelen ser aprendidos por otra red neuronal diferente, llamada **encoder**, la cual recibe como entrada una muestra $x \in \mathbb{R}^D$. Considerando que usualmente se utiliza $L \ll D$, este tipo de modelos tiene forma de autoencoder, de donde viene su nombre. La parte variacional

tiene que ver con su función de costo, la cual resulta ser una cota inferior de la log-verosimilitud que se obtiene usando inferencia variacional.

2.2.2.3. Redes generativas adversarias (GANs)

Las GANs también son modelos de variable latente de la forma $p(x, z) = p(z)\, p(x \mid z)$ solo que, a diferencia de los VAEs que además incluyen un encoder, este tipo de modelos incluye un clasificador externo, $q(y \mid x)$, que es desechado después del entrenamiento. La idea principal de una GAN es que el clasificador $q(y \mid x)$ aprenda a reconocer si una muestra dada, $x \in \mathbb{R}^D$, es una muestra artificial ($y = 0$) generada desde $p(x \mid z)$ o si es una muestra real ($y = 1$) generada desde la distribución desconocida $\mathrm{p}_{\mathrm{data}}(x)$. De este modo, mientras el clasificador es entrenado para esta tarea, el generador es entrenado para engañar al clasificador (i.e., busca que identifique las muestras generadas desde $p(x \mid z)$ como reales), lo que genera como consecuencia que el generador aprenda a imitar muy bien las muestras que se generan desde $\mathrm{p}_{\mathrm{data}}(x)$. Los DAGs asociados a una GAN son muy similares a los de un VAE:

Figura 10: DAGs asociados a una GAN.

En este caso, al modelo $p(x \mid z)$ se le llama **generador** (no se usa la interpretación encoder--decoder), mientras que al clasificador $q(y \mid x)$ se le suele llamar **discriminador**.

2.2.2.4. Modelos de difusión (DMs)

Este tipo de modelos está compuesto por dos distribuciones distintas. Una primera distribución (fija) inyecta ruido de manera progresiva a muestras de $\mathrm{p}_{\mathrm{data}}(x)$ hasta llegar a una imagen de puro ruido. Dado que el ruido se va inyectando directamente sobre la imagen ruidosa anterior, este proceso de destrucción de información puede ser expresado como una cadena de Markov:

$$q(x, z_1, ..., z_T) = \mathrm{p}_{\mathrm{data}}(x) \prod_{t=1}^{T} q(z_t \mid z_{t-1}), \tag{16}$$

donde $q(z_t \mid z_{t-1})$ (con $z_0 = x$) son transiciones ruidosas tales que $p(z_T) \approx \mathcal{N}(0, \mathrm{I}_D)$. Al mismo tiempo, otra red bayesiana es entrenada para aprender a deshacer el proceso de inyección de ruido. Esta red bayesiana, cuyos parámetros son aprendidos por una red neuronal, también es una cadena de Markov pero hacia atrás en el tiempo (ya que busca revertir la cadena de Markov que inyecta el ruido):

$$p(x, z_1, ..., z_T) = p(z_T) \prod_{t=1}^{T} p(z_{t-1} \mid z_t), \tag{17}$$

donde $p(z_T) \sim \mathcal{N}(0, \mathrm{I}_D)$ es el nodo raíz de esta red bayesiana. Los DAGs asociados a ambos modelos gráficos son los siguientes:

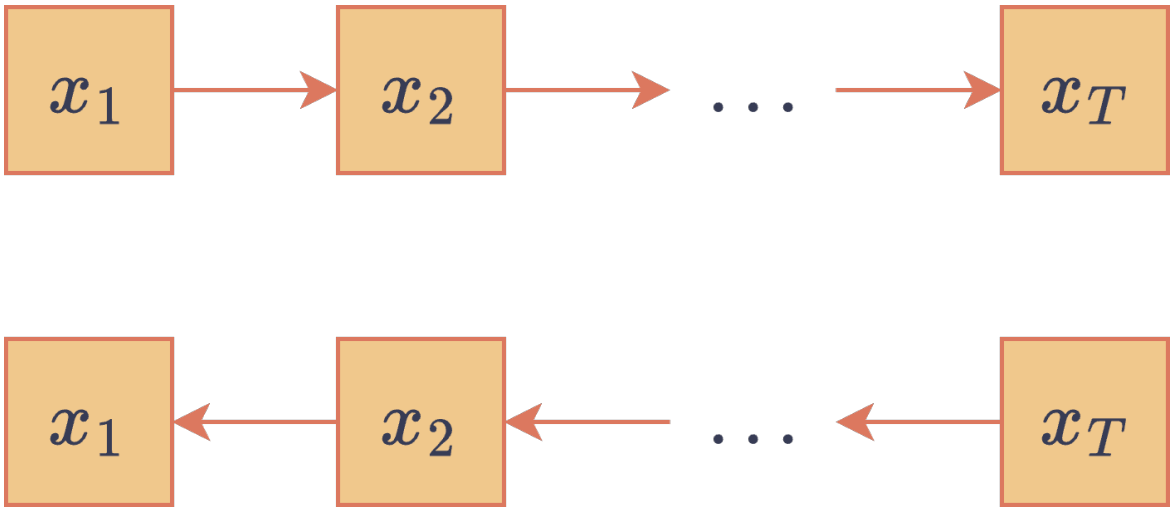

Figura 11: DAGs asociados a un modelo de difusión.

Una vez se tiene entrenada la red neuronal para $p(x, z_1, ..., z_T)$, se puede generar una nueva muestra comenzando con una muestra generada desde $p(z_T) \sim \mathcal{N}(0, \mathrm{I}_D)$ y aplicando el proceso de denoising aprendido para llegar a una muestra $x \in \mathbb{R}^D$, la cual debería parecerse a una muestra de $\mathrm{p}_{\mathrm{data}}(x)$ si el modelo de difusión fue bien entrenado.

2.2.2.5. Flujos normalizantes (NFs)

El paradigma de los flujos normalizantes modela una distribución de probabilidad $p(x \mid z)$ mediante la transformación de una variable aleatoria simple y bien conocida z a través de una serie de funciones invertibles y diferenciables. La red bayesiana asociada a estos modelos puede verse, al igual que los modelos anteriores, como un modelo de variable latente $p(x, z) = q(z)\, p(x \mid z)$, donde la distribución $p(x \mid z)$ es determinista, es decir, $x = f(z)$, cuando se conoce el valor de $z \sim q(z)$. Dado que la transformación se considera invertible y de inversa diferenciable (i.e., f es un difeomorfismo), el teorema de cambio de variable permite obtener la densidad $p(x)$ a partir de la densidad $q(z)$:

$$p(x) = q(f^{-1}(x)) \left|\det(\mathrm{D}_x\, f^{-1}(x))\right| \tag{18}$$

Por lo general, la transformación $f : \mathbb{R}^D \to \mathbb{R}^D$ se construye componiendo varias transformaciones más simples, $f(z) = (f_N \circ \cdots \circ f_1)(z)$, por lo que el jacobiano $\mathrm{D}_x\, f^{-1}(x) \in \mathcal{M}_{D,D}(\mathbb{R})$ se puede descomponer, de acuerdo a la regla de la cadena (para derivadas), como el producto de varios jacobianos individuales.

El DAG asociado a un modelo basado en flujos normalizantes se puede ver, dependiendo de la dirección temporal que se elija, de dos formas distintas:

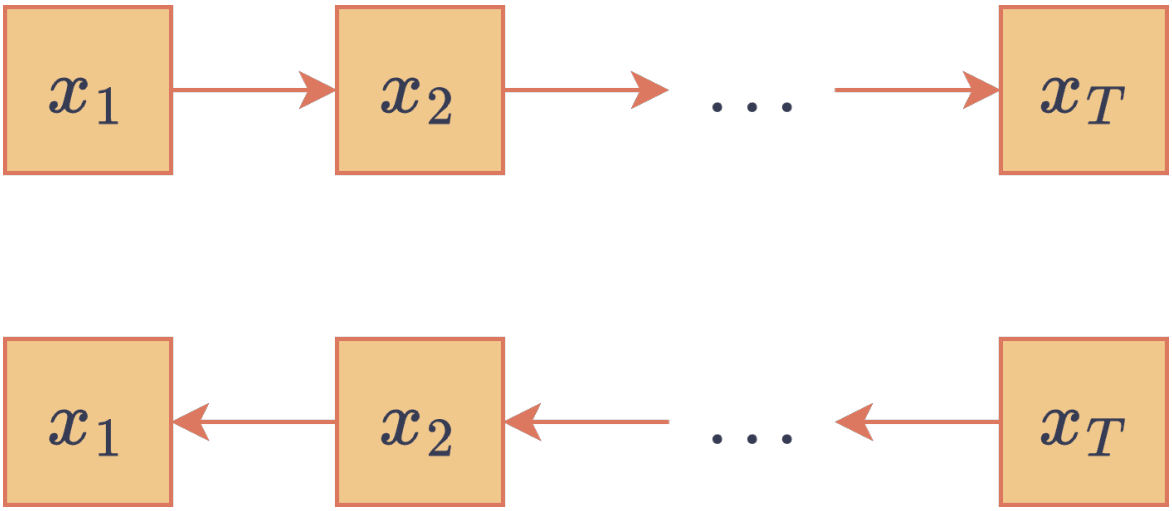

Figura 12: DAGs asociados a un modelo de flujos normalizantes.

Por último, es importante notar que cada paradigma utiliza un enfoque distinto para resolver el problema de aprender una distribución desconocida, cada uno aprovechando suposiciones e hipótesis de independencia distintas, lo cual le otorga a cada enfoque propiedades únicas, tanto en su forma de entrenar, como en el tipo de generación que realiza. Por otro lado, es interesante mencionar que todos los modelos tienen, de un modo u otro, alguna conexión con la física termodinámica: los ARMs usan la idea de temperatura para controlar la diversidad en la generación; los VAEs y los DMs entrenan una cantidad llamada ELBO, la cual puede relacionarse con la energía libre de Helmholtz; y los EBMs se basan en escribir la función de densidad como una distribución de Boltzmann y modelan directamente una función de energía.

2.2.3. Modelos generativos condicionales

Hasta el momento, todos los modelos generativos han sido formulados para aprender una distribución $p(x)$ que aproxime bien a otra distribución desconocida, $\mathrm{p}_{\mathrm{data}}(x)$. Sin embargo, en muchos casos prácticos se dispone de información adicional que se quiere que influya en la generación de los datos. Esto da lugar a los **modelos generativos condicionales**, donde en lugar de modelar una distribución incondicional $p(x)$, se modela una distribución condicional $p(x \mid y)$, con y representando una variable observada que actúa como entrada adicional al modelo y que busca modificar el comportamiento de la distribución. Este tipo de condicionamiento es el que permite generar imágenes dada una descripción o conversar con un chatbot (en vez de generar texto libre). En estos casos, a la condición y se le suele llamar **prompt**.

Los modelos condicionales, junto a la capacidad de multimodalidad (esto es, procesar varias modalidades: texto, imágenes, sonido, videos, etc.), permiten resolver algunas tareas que parecen difíciles (o hasta imposibles) de resolver con otros enfoques más clásicos, por ejemplo:

- **Traducción de texto:** la arquitectura Transformer [1] (y otras arquitecturas seq2seq [2]) son diseñadas para aprender un modelo condicional $p(x \mid y)$ que es capaz de traducir un texto de entrada y a otro idioma (texto de salida x). En la arquitectura Transformer el texto es codificado mediante un mecanismo de auto-atención, mientras que la información condicional es inyectada al modelo mediante un mecanismo de atención cruzada.

- **Generación de imágenes a partir de texto:** modelos como DALL-E [3] y Stable Diffusion [4] permiten generar imágenes a partir de descripciones textuales. En este caso, los modelos aprenden una distribución condicional $p(x \mid y)$ donde x es la imagen generada e y es el texto dado como entrada (prompt). En el caso de DALL-E, $p(x \mid y)$ es modelado por un modelo autorregresivo, mientras que en Stable Diffusion, $p(x \mid y)$ es modelado por un modelo de difusión. El éxito de los modelos de difusión llevó a OpenAI a cambiar a este paradigma para diseñar DALL-E 2 [5] y DALL-E 3 [6].
- **Traducción y modificación de imágenes:** modelos como pix2pix [7] y CycleGAN [8] (ambos modelos tipo GAN) o FLUX.1 Kontext [9] (modelo de flow matching) permiten realizar tareas sobre imágenes como colorización de imágenes en escala de grises e inpainting (imputación de píxeles, por ejemplo, para corregir detalles estéticos). En todos estos casos, x es la imagen generada e y es la imagen de entrada. De forma similar se pueden modelar otras tareas como super-resolución.
- **Automatic speech recognition (ASR):** modelos como Whisper [10] (modelo tipo Transformer que procesa secuencias de audio en vez de secuencias de texto) permiten transcribir grabaciones de voz modelando una distribución condicional $p(x \mid y)$, donde y es una pista de audio y x es una secuencia de texto. La tarea inversa, la cual consiste en modelar $p(y \mid x)$, permitiría resolver la tarea de generar un audio de voz a partir del texto que se debe hablar en el audio.

En este tipo de modelos, la generación de nuevas muestras depende explícitamente del valor del factor condicionante y, el cual es necesario para controlar y dirigir el proceso de generación dependiendo de qué valor se entregue como entrada adicional al modelo. Por otro lado, es importante notar que cualquier red bayesiana incondicional, $p(x_1, ..., x_N) = \prod_{n=1}^{N} p(x_n \mid \mathrm{Pa}(x_n))$, puede ser transformada en una red bayesiana condicional, agregando y a cada uno de los factores:

$$p(x_1, ..., x_N \mid y) = \prod_{n=1}^{N} p(x_n \mid \mathrm{Pa}(x_n), y) \tag{19}$$

Más aún, dado que cada factor $p(x_n \mid \mathrm{Pa}(x_n))$ suele ser aprendido por una red neuronal (cuya entrada son los valores de las variables aleatorias $\mathrm{Pa}(x_n)$), la misma arquitectura neuronal usada para aprender (los parámetros de) $p(x_n \mid \mathrm{Pa}(x_n))$ suele ser adaptada para aprender (los parámetros de) la distribución condicional $p(x_n \mid \mathrm{Pa}(x_n), y)$. Para esto, basta modificar la red neuronal para poder inyectar, de algún modo, el valor de y en el input. Una forma fácil y que muchas veces funciona es concatenar el valor de y al resto de entradas de la red neuronal, aunque también es posible utilizar mecanismos más complejos. Por ejemplo, Stable Diffusion inyecta el prompt al proceso de generación utilizando un mecanismo de atención cruzada, mientras que la arquitectura DiT [11] (Diffusion Transformer) inyecta la condición mediante moduladores tipo FiLM [12].

Esta facilidad para transformar una arquitectura incondicional en una arquitectura condicional ha permitido un rápido desarrollo de los modelos generativos multimodales (MLLMs). Encontrando formas eficientes de representar imágenes como vectores (o tensores de algún rango mayor), un modelo de lenguaje como GPT 4 puede ser transformado a un modelo que permita recibir adicionalmente imágenes como entrada, al igual que, por ejemplo, el modelo GPT 4o. De forma análoga se puede condicionar cualquier modelo generativo basado en redes neuronales con respecto a otras modalidades como video, temperatura, movimiento, etc., siempre y cuando

se conozca una forma eficiente de inyectar la información condicional en la red neuronal. A la representación vectorial de una entrada en alguna modalidad específica se le suele llamar **vector de embedding**, mientras que al modelo que transforma dicha modalidad en una representación vectorial se le suele llamar **modelo de embedding** o **capa de embedding** en el caso de estar dentro de un modelo más grande.

Para mayor simplicidad, por lo general se considerarán modelos generativos incondicionales al momento de formular los distintos paradigmas, pero siempre hay que tener en cuenta que el enfoque utilizado actualmente (basado en redes neuronales) permite añadir condiciones adicionales al modelo gráfico como entradas adicionales a las redes neuronales sin mayor complejidad. Más aún, por lo general uno siempre desea tener un modelo generativo condicional ya que no suele ser suficiente tener un modelo que genere objetos (e.g., imágenes o texto) de forma libre, sino que se busca poder guiar el proceso de generación (e.g., mediante un prompt) para obtener resultados útiles para el usuario final.

Por último, es importante notar que para construir un modelo generativo condicional es necesario aprender una distribución $p(x \mid y)$ (que se puede considerar como el opuesto bayesiano de un clasificador), por lo que es necesario, al igual que en el caso supervisado, tener pares condición--muestra para el entrenamiento (aunque no siempre; algunos modelos condicionales siguen un enfoque autosupervisado). Sin embargo, la obtención de estos datos etiquetados muchas veces es más fácil que en un clasificador clásico. Por ejemplo, para obtener un dataset de imágenes con texto descriptivo, se pueden obtener los captions a partir del texto alternativo de imágenes sacadas de internet.

2.2.3.1. Relación con los modelos discriminativos

Por lo general, un curso clásico de machine learning o deep learning se enfoca principalmente en estudiar tópicos de aprendizaje supervisado donde, por ejemplo, una red neuronal aprende a discriminar a partir de múltiples ejemplos etiquetados. Es decir, el modelo aprende una distribución condicional $p(y \mid x)$ que aprende a reconocer el grupo al que pertenece una muestra. Por esto, a este tipo de modelos se les llama **modelos discriminativos**.

Dado un modelo generativo $p(x, y)$, siempre es posible, al menos en teoría, obtener un modelo discriminativo considerando que $p(y \mid x) = \frac{p(x,y)}{\int p(x,y)\,\mathrm{d}\,y}$. Sin embargo, si la tarea objetivo es de tipo discriminativa (e.g., un clasificador), por lo general se obtiene un mejor desempeño entrenando directamente un modelo $p(y \mid x)$ en vez de adaptar un modelo generativo $p(x, y)$ a su versión discriminativa. Esto es esperable ya que ambos enfoques siguen paradigmas de entrenamiento distintos, donde los modelos discriminativos son entrenados precisamente para rendir bien en la tarea de clasificación. Por otro lado, un modelo discriminativo $p(y \mid x)$ solo se enfoca en aprender a clasificar objetos en categorías predefinidas, sin modelar la estructura subyacente de los datos mediante el aprendizaje de la distribución $p(x)$. En cambio, un modelo discriminativo obtenido a partir de un modelo generativo aprende ambas distribuciones al aprender la distribución conjunta $p(x, y)$. En consecuencia, los modelos generativos suelen tener un mejor entendimiento del mundo que los modelos discriminativos, ya que un modelo discriminativo no puede, por ejemplo, generar nuevas muestras a partir de una clase determinada. Además, los modelos discriminativos pueden ser vulnerables a ataques adversarios, donde se ha demostrado que pequeñas modificaciones en la entrada pueden cambiar totalmente la predicción. En cambio, un modelo discriminativo

obtenido a partir de uno generativo podría ser más robusto a estas perturbaciones. Más aún, en problemas de aprendizaje semisupervisado, donde hay pocos ejemplos etiquetados y muchos sin etiquetar, los modelos generativos pueden ser útiles para mejorar el clasificador final.

Otra observación importante es que hasta hace no muchos años, los principales avances en deep learning (e.g., AlexNet, ResNet y EfficientNet) correspondían a modelos usados para aprendizaje supervisado, ya que se daba por hecho que las tareas generativas estaban limitadas solo a los humanos. Sin embargo, en los últimos diez años, el mayor progreso ha estado dominado por los modelos generativos, los cuales son, usualmente, de naturaleza no supervisada. Parte de este rápido desarrollo es gracias a las técnicas desarrolladas en modelos discriminativos, las cuales han sido adaptadas a modelos generativos. Dentro de estas técnicas se encuentran muchas de las mejoras en arquitecturas neuronales (e.g., ReLU, bloques residuales, dropout, BatchNorm), pero también se encuentran mejoras en los frameworks y en el hardware usado.

2.2.4. Modelos de variable latente

En el ejemplo inicial, todas las variables del modelo pueden ser observadas (es decir, además de conocer sus distribuciones, se puede conocer el valor que toman), por lo que se dice que el modelo es completamente observable. Sin embargo, en modelos probabilísticos más complejos, es común que algunas variables del sistema no sean directamente observables, sino que sean **variables latentes** (también llamadas **variables ocultas**), las cuales se suelen considerar como variables ficticias que influyen sobre las variables observadas, permitiendo, además, explicar la variabilidad de los datos observados. A modo de ejemplo, las imágenes del dataset CelebA [13] (dataset de imágenes de caras) se representan como vectores de dimensión $D = 178 \times 218 \times 3 = 116\,412$. Sin embargo, uno puede hipotetizar que las imágenes fueron generadas transformando una variable aleatoria oculta (que vive en un espacio de dimensión $L < D$) al espacio ambiente $\mathbb{R}^D$. Este espacio latente podría ser un conjunto de variables más primitivas como el sexo de la persona en la fotografía, su color de pelo, el color de sus ojos, su color de piel, si usa lentes o no, si usa sombrero, etc., y las distintas imágenes que se puedan obtener una vez se definen estas variables latentes vienen dadas por la aleatoriedad del proceso de generación.

El uso de modelos de variables ocultas se suele justificar mediante la **manifold hypothesis**, la cual propone que los datos observados viven en un espacio ambiente de alta dimensión, $\mathbb{R}^D$, pero que realmente provienen de un espacio (más precisamente, una variedad diferenciable) de menor dimensión, $\mathbb{R}^L$. La hipótesis de la variedad es una hipótesis bastante aceptada (con evidencia empírica) y permite, entre otras cosas, explicar la aparente ausencia de la maldición de la dimensionalidad (a.k.a. efecto Hughes) al entrenar redes neuronales. Por otro lado, el uso de los modelos de variable latente permite construir formas ingeniosas de formular distintos enfoques generativos, los cuales muchas veces no tienen relación entre sí más que ser modelos de variable latente. Por ejemplo, las GANs y los VAEs son dos tipos de modelo generativo de variable latente que funcionan de manera muy diferente. Cada uno de estos paradigmas, al ser entrenados bajo criterios distintos, pueden aprender comportamientos y patrones distintos, incluso si son entrenados sobre un mismo conjunto de datos. Por ejemplo, en el caso de las GANs y los VAEs, se observa que las GANs son buenas generando de forma clara los bordes de los objetos, mientras que los VAEs generan bordes más borrosos. Sin embargo, las GANs tienen dificultades para aprender muestras con mayor variabilidad, mientras que los VAEs son capaces de capturar mejor el soporte de la distribución desconocida $\mathrm{p}_{\mathrm{data}}(x)$ que se busca aproximar.

En la siguiente figura se ve el dataset swiss roll [14], el cual será usado en todas las implementaciones iniciales (excepto en los ARMs) para introducir cada uno de los paradigmas generativos. En línea con la hipótesis de la variedad, se observa que, si bien las muestras viven en el espacio ambiente $\mathbb{R}^3$, estas siguen una estructura que pareciera poder explicarse, al menos de forma aproximada, con solo dos grados de libertad.

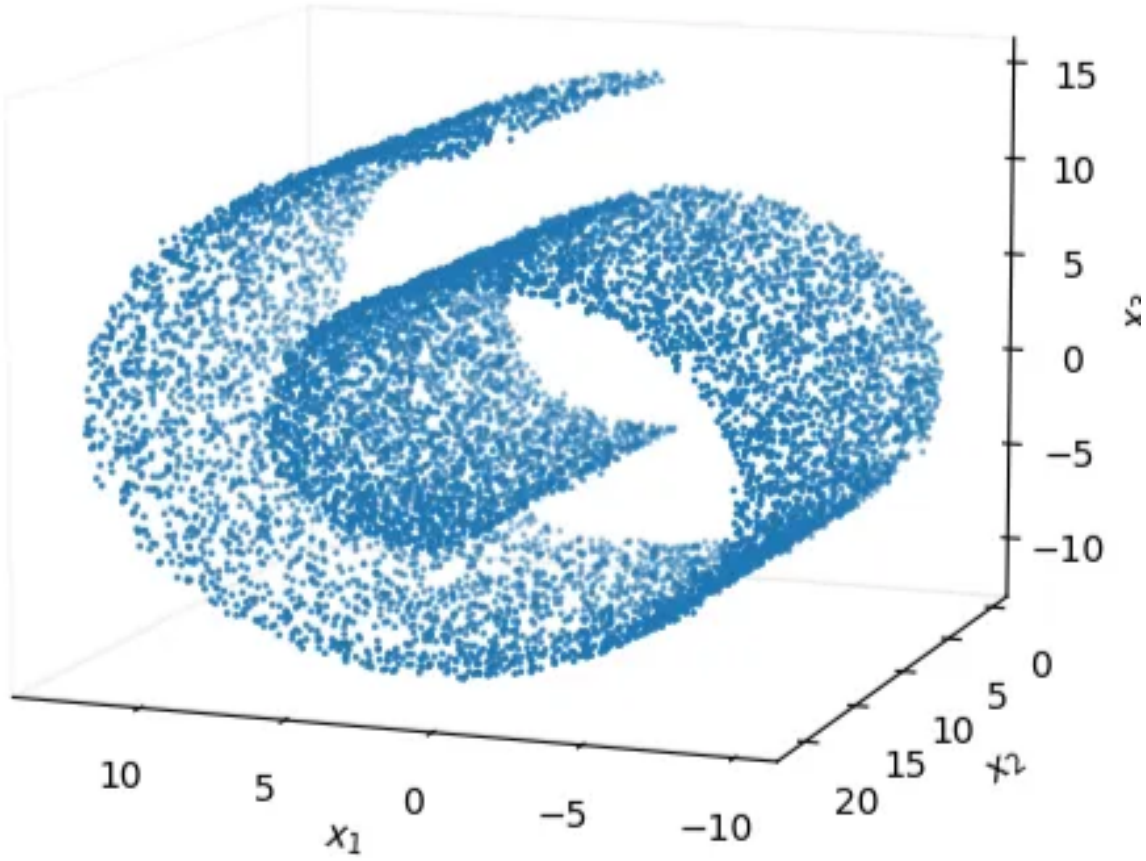

Figura 13: Swiss roll [14].

- Código para la figura.

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll

x, _ = make_swiss_roll(10000, noise=0.1)

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x[:, 0], x[:, 1], x[:, 2], s=1)
ax.view_init(elev=10, azim=110)
ax.xaxis.pane.fill = False
ax.yaxis.pane.fill = False
ax.zaxis.pane.fill = False
ax.set_xlabel(r'$x_1$')
ax.set_ylabel(r'$x_2$')
ax.set_zlabel(r'$x_3$')
ax.grid(False)
plt.show()
```

Más en general, se han encontrado dimensiones intrínsecas para algunos datasets clásicos, lo que valida empíricamente la hipótesis de la manifold. Por ejemplo, para el dataset MNIST se ha probado experimentalmente que su dimensión intrínseca es cercana a 12, lo cual es considerablemente menor que la cantidad de pixeles de las imágenes de este dataset, donde $D = 728$. Por otro lado, en los casos donde la hipótesis de la variedad no se cumple (o donde no existe realmente una variable latente), el considerar un modelo de variable latente no perjudica la formulación ya que si el modelo real desconocido no fuese de variable latente, siempre se puede terminar aprendiendo que $x \perp z$, es decir, $p(x \mid z) = p(x)$.

Vista como una red bayesiana, la distribución conjunta de un modelo de variable latente se factoriza de forma causal como $p(x, z) = p(z)\, p(x \mid z)$, donde $p(z)$ es el prior (se llama así ya que no está condicionado a nada) sobre la variable latente z y $p(x \mid z)$ es la distribución condicional de los datos observados dado el valor de la variable latente z (que no es lo mismo que la marginal $p(x) = \int_{\mathbb{R}^L} p(x, z)\, \mathrm{d}\ z$). En estos casos, el modelo $p(x \mid z)$ se interpreta como un modelo generador, dada una semilla inicial z, por lo que también se le conoce como **decoder**. Es importante notar que $x \in \mathbb{R}^D$ está representando todas las variables observables unidas en una única variable con el fin de simplificar la notación. Por ejemplo, para un modelo que modifica una imagen de acuerdo a una descripción, x podría ser un vector formado por algún embedding de la imagen original y del texto con las instrucciones de edición. El mismo principio aplica para $z \in \mathbb{R}^L$, por lo que a z se le puede decir la variable latente o las variables latentes.

Por otra parte, a excepción de los modelos autorregresivos (usados en texto) y los modelos basados en energía (ya no tan usados en IA generativa, al menos en su versión original), todos los paradigmas generativos modernos son modelos de variable latente, por lo que es necesario hacer hipótesis sobre la distribución prior $p(z)$. Por lo general, es usual considerar $p(z) \sim \mathcal{N}(0, \mathrm{I}_L)$ ya que funciona bien al desarrollar la matemática de los distintos modelos y, además, es una distribución desde la que es fácil generar muestras. Esto último es una propiedad importante ya que prácticamente todos los modelos de variable latente (como las GANs, los VAEs o los modelos de difusión) funcionan generando una muestra latente inicial $z \sim p(z)$ y luego generan una nueva muestra (e.g., una imagen) utilizando el valor de z en la distribución $p(x \mid z)$. Además, al estar considerando una matriz de covarianzas diagonal en $p(z) \sim \mathcal{N}(0, \mathrm{I}_L)$, se está imponiendo explícitamente que todas las coordenadas de la variable latente sean independientes entre sí, lo cual no es raro si se interpretan las variables latentes como el conjunto de características esenciales que poseerán las muestras que se generen desde $p(x \mid z)$.

Desde el punto de vista de la generación en modelos de variable latente, se observa que el algoritmo de ancestral sampling sí entrega muestras de la distribución marginal $p_\theta(x)$ ya que

$$p_\theta(x) = \int_{\mathbb{R}^L} p_\theta(x \mid z)\, p(z)\, \mathrm{d}\ z = \mathbb{E}_{p(z)}[p_\theta(x \mid z)], \tag{20}$$

por lo que se puede ver el sampling $z \sim p(z) \to x \sim p(x \mid z)$ como una aproximación de Monte Carlo (con una única muestra) de la esperanza $\mathbb{E}_{p(z)}[p_\theta(x \mid z)]$.

A continuación se revisarán algunas de las ventajas que poseen los modelos generativos basados en variable latente.

2.2.4.1. Reducción de la dimensionalidad

Si bien el proceso generativo de un modelo de variable latente es de la forma $z \to p(x \mid z)$, es posible considerar la dirección opuesta, $x \to p(z \mid x)$, con el fin de recuperar la información primitiva que generó una muestra $x \in \mathbb{R}^D$. Esta distribución posterior $p(z \mid x)$ puede interpretarse como una representación (estocástica) más compacta de x, la cual muchas veces es interpretable o útil. Por ejemplo, si $p(x)$ es una distribución que genera imágenes aleatorias de círculos de color negro con una resolución de $D = 256 \times 256 = 65536$, el proceso de generación se podría reducir a una variable latente z en $\mathbb{R}^3$ que genera ternas (r, h, k) con $r > 0$ indicando el radio del círculo y $(h, k) \in \mathbb{R}^2$ las coordenadas de su centro. Con esto, la distribución generadora $p(x \mid z)$ puede ser la función determinista que grafica el respectivo círculo y genera la imagen.

La tarea de aprender buenas representaciones latentes se denomina **representation learning** y puede ser vista como una forma robusta de hacer reducción de dimensionalidad. Por ejemplo, el modelo Stable Diffusion [4] de Stability AI entrena un modelo de variable latente (tipo VAE) para luego poder entrenar otro modelo generativo más potente (tipo difusión) en el espacio latente del modelo anterior. Esta técnica de aplicar un modelo generativo (e.g., difusión) en el espacio latente de otro modelo (e.g., un VAE) es usada prácticamente en todos los modelos generativos de variable latente que hay hoy en día ya que permite disminuir el costo y tiempo de entrenamiento considerablemente al poder trabajar con representaciones de menor dimensión.

2.2.4.2. Interpolación semántica en el espacio latente

La representación latente $p(z \mid x)$ de una muestra $x \in \mathbb{R}^D$, aparte de actuar como una representación más compacta de x, suele ser semánticamente más rica que la representación original, en el sentido que es posible desplazarse en el espacio latente para cambiar algunas propiedades semánticas de la muestra, entregando una forma de **interpolación semántica** entre dos objetos.

Si $x_0, x_1 \in \mathbb{R}^D$ son dos imágenes, una tarea natural es querer interpolar entre estas dos imágenes, realizando una transición suave de una imagen a otra. Un enfoque naïve es interpolar linealmente en el espacio de pixeles, donde se considera como imagen interpolada a $x_t = tx_1 + (1-t)x_0$, para $t \in [0, 1]$. Esta interpolación consiste en ir desplazando cada pixel a lo largo de la recta que unen los pixeles homólogos en las imágenes x_0 y x_1. Sin embargo, dado que los valores de los pixeles representan los colores de la imagen, esta interpolación es una interpolación lineal entre la intensidad de color de cada pixel individual, los cuales no contienen información semántica acerca de la imagen, por lo que la imagen interpolada, $x_t \in \mathbb{R}^D$, suele ser una imagen sin mucho sentido.

Un enfoque que funciona mejor consiste en utilizar un modelo generativo de variable latente, $p(x, z) = p(z)\, p(x \mid z)$. De este modo, si una función $x \mapsto z = \text{encoder}(x)$ genera una muestra desde la distribución posterior $p(z \mid x)$ (distribución obtenida, por ejemplo, usando la regla de Bayes), se pueden obtener representaciones latentes de x_0 y x_1 mediante $z_0 = \text{encoder}(x_0)$ y $z_1 = \text{encoder}(x_1)$ respectivamente. Con estas representaciones, se puede realizar una interpolación en el espacio latente del modelo generativo mediante $z_t = tz_1 + (1-t)z_0 \in \mathbb{R}^L$. Luego, si $z \mapsto x = \text{decoder}(z)$ genera una muestra desde la distribución $p(x \mid z)$, entonces $x_t = \text{decoder}(z_t) \in \mathbb{R}^D$ es una imagen interpolada pero en el espacio latente de las imágenes x_0 y x_1. Dado que el espacio latente suele estar asociado a atributos específicos de las muestras que se generarán, la interpolación en el espacio latente suele ser mucho más natural que la interpolación obtenida en

el espacio de pixeles debido a que resulta ser una interpolación de propiedades esenciales de las muestras a generar y no solo una interpolación pixel a pixel. Por otro lado, si bien se podrían realizar otro tipo de interpolaciones en el espacio latente (e.g., slerp), la interpolación lineal suele ser suficiente. De hecho, algunos estudios han probado empíricamente que la variedad latente aprendida por los modelos de variable latente a menudo tiene una curvatura cercana a cero, por lo que se asemeja a un espacio euclidiano, donde la interpolación lineal es el método de interpolación natural.

En la siguiente imagen se observan interpolaciones semánticas en el espacio latente de una GAN, donde cada fila realiza la interpolación entre las imágenes de la primera y última columna:

Figura 14: Interpolación latente usando una DCGAN. Imagen obtenida desde [15].

2.2.4.3. Manipulación de atributos

Otra herramienta que entrega la continuidad del espacio latente es la posibilidad de modificar la intensidad de algún atributo de una muestra. Por ejemplo, en la siguiente imagen se modifica la intensidad del atributo sonrisa de la imagen que está en la primera columna:

Figura 15: Modificación de atributos usando un VAE. Imagen obtenida desde [16].

Para lograr esto, es necesario encontrar una dirección en el espacio latente que apunte hacia donde aumenta la intensidad del atributo que se quiere destacar en la variable observable. Luego, para una muestra $x \in \mathbb{R}^D$, se puede desplazar su representación latente $z = \text{encoder}(x) \in \mathbb{R}^L$ en esta dirección para aumentar (o disminuir) la intensidad del atributo en la muestra original.

Más precisamente, si $\mathcal{X}^+ \subset \mathbb{R}^D$ es un conjunto de muestras que notoriamente poseen el atributo que se busca modificar (e.g., imágenes de personas sonriendo) y $\mathcal{X}^- \subset \mathbb{R}^D$ es un conjunto de muestras que notoriamente no poseen el atributo (e.g., imágenes de personas serias), entonces se pueden considerar los centroides latentes para ambos conjuntos:

$$z^+ = \frac{1}{|\mathcal{X}^+|} \sum_{x \in \mathcal{X}^+} \text{encoder}(x), \quad z^- = \frac{1}{|\mathcal{X}^-|} \sum_{x \in \mathcal{X}^-} \text{encoder}(x) \tag{21}$$

De este modo, se puede considerar el vector director $z_{\text{atributo}} = z^+ - z^- \in \mathbb{R}^L$ como la dirección latente asociada al atributo que se busca amplificar o aminorar. Luego, para $\lambda \in \mathbb{R}$ y para una muestra $x \in \mathbb{R}^D$ generada desde $p(x)$, se puede considerar a $x_\lambda = \text{decoder}(\text{encoder}(x) + \lambda z_{\text{atributo}})$ como la muestra que aumentó ($\lambda > 0$) o disminuyó ($\lambda < 0$) la intensidad del atributo en la muestra x. Este procedimiento permite trabajar con representaciones latentes interpretables, las cuales, además, permiten alterar propiedades de alto nivel mediante interpolación en el espacio latente. Sin embargo, hoy en día, la modificación de atributos suele realizarse mediante el uso de modelos condicionales $p(x \mid y)$ que generan imágenes de acuerdo a una instrucción contenida en y (prompt).

Por otro lado, es importante mencionar que por lo general no es posible encontrar una función $x \mapsto z = \text{encoder}(x)$ para las GANs, por lo que este procedimiento solo se puede realizar cuando se conoce de antemano la variable latente que genera a la respectiva muestra x. En cambio, los VAEs cuentan desde un comienzo con un modelo encoder, $q(z \mid x)$, por lo que sí es posible realizar este tipo de interpolaciones para muestras genéricas.

2.3. Estimación de parámetros

Dada una red bayesiana, para conocer la distribución conjunta $p(x_1, ..., x_N)$ es necesario y suficiente conocer cada uno de los factores $p(x_n \mid \mathrm{Pa}(x_n))$ que componen la factorización que asume la red bayesiana, $p(x_1, ..., x_N) = \prod_{n=1}^{N} p(x_n \mid \mathrm{Pa}(x_n))$. Si bien el desarrollo realizado hasta el momento permite considerar variables con valores desconocidos (variables latentes), siempre se ha asumido que todos los parámetros de las distribuciones $p(x_n \mid \mathrm{Pa}(x_n))$ son conocidos (e.g., si uno de los factores $p(x_n \mid \mathrm{Pa}(x_n))$ es una distribución gaussiana, se conocen los valores de su media y de su varianza para los distintos valores de las variables $\mathrm{Pa}(x_n)$). Sin embargo, en los modelos de IA generativa lo usual es no conocer dichos parámetros y solo contar con un conjunto de muestras (dataset de entrenamiento) generadas desde la distribución desconocida $\mathrm{p}_{\mathrm{data}}(x_1, ..., x_N)$ que se busca aprender. El proceso de estimar los valores de estos parámetros desconocidos (lo que usualmente se logra entrenando redes neuronales) se denomina **inferencia**, y la forma más usual de hacer inferencia (puntual) es siguiendo el criterio de máxima verosimilitud.

De manera general, inferir es la acción de obtener una conclusión lógica a partir de un conjunto de premisas que se asumen como ciertas. La inferencia estadística corresponde a realizar inferencia sobre una distribución a partir de un conjunto de muestras observadas, donde el objetivo más usual es inferir los parámetros de la distribución que (supuestamente) generó los datos. Dentro de la inferencia estadística se encuentra la inferencia bayesiana, donde se utiliza la regla de Bayes para hacer inferencia sobre variables desconocidas. De esta forma, la inferencia bayesiana evalúa la probabilidad de que una hipótesis sea cierta a partir de una creencia a priori sobre la hipótesis y de un conjunto de muestras observadas, las cuales actualizan la información asumida por el prior previo a la observación.

Sobre una red bayesiana, se pueden realizar 3 tipos de inferencia bayesiana: inferencia sobre variables no observadas (i.e., estimar el valor de variables desconocidas), inferencia sobre los parámetros de alguna de las distribuciones $p(x_n \mid \mathrm{Pa}(x_n))$ del grafo, e inferencia sobre el grafo mismo (i.e., aprender la estructura del grafo), aunque esta última tarea de inferencia no se trabajará. Para la tarea de inferencia sobre los valores de variables desconocidas, se puede considerar como ejemplo un modelo de variable latente $p(x, z) = p(z)\, p(x \mid z)$, donde se asume que se conoce el valor de la variable observable x pero no el valor de la variable latente z. De este modo, se puede inferir la distribución posterior $p(z \mid x)$ mediante la regla de Bayes:

$$p(z \mid x) = \frac{p(x \mid z)\, p(z)}{p(x)} \tag{22}$$

Notar que la distribución marginal $p(x) = \int_{\mathbb{R}^L} p(x, z)\, \mathrm{d}\, z$ (o $p(x) = \sum_z p(x, z)$ si z es variable latente discreta) es costosa de obtener ya que requiere integrar (sumar en el caso discreto) sobre todos los posibles valores de la variable latente z.

En las tareas de IA generativa, la red bayesiana de cada paradigma generativo suele ser conocida, por lo que no se debe hacer inferencia sobre el grafo. En cambio, el foco suele estar en hacer inferencia sobre los parámetros de las distribuciones del modelo gráfico, los cuales permitirán, posteriormente, generar nuevos datos utilizando las distribuciones aprendidas. Los modelos modernos consideran, por lo general, que cada uno de los factores $p(x_n \mid \mathrm{Pa}(x_n))$ sigue una distribución simple (e.g., una distribución gaussiana si x_n es una variable continua

o una distribución Bernoulli si x_n es binario), lo que facilita el desarrollo de la matemática al construir las funciones de pérdida o demostrar propiedades necesarias para la formulación de los modelos. Dada la simpleza de las distribuciones comúnmente utilizadas, la complejidad de las distribuciones $p(x_n \,|\, \mathrm{Pa}(x_n))$ debe ser trasladada a los parámetros que las definen, lo cual se suele conseguir utilizando redes neuronales que estimen dichos parámetros. Por ejemplo, en el caso continuo donde se suele elegir $p(x_n \,|\, \mathrm{Pa}(x_n)) \sim \mathcal{N}(\mu_n, \sigma_n^2)$, los parámetros $\mu_n \in \mathbb{R}$ y $\sigma_n^2 > 0$ suelen ser las salidas de redes neuronales cuyas entradas son los valores de las variables en $\mathrm{Pa}(x_n)$. Sin embargo, también es usual fijar algunos parámetros menos relevantes como las varianzas de las distribuciones gaussianas para poder disminuir la complejidad del modelo.

Si $\theta_n \in \mathbb{R}^{P_n}$ denota el vector de parámetros desconocidos de la distribución $p(x_n \,|\, \mathrm{Pa}(x_n))$ y $\theta = (\theta_1, ..., \theta_N) \in \mathbb{R}^P$ (con $P = \sum_{n=1}^N P_n$) denota el vector de todos los parámetros desconocidos de la red bayesiana, la forma usual de hacer inferencia sobre θ es considerar que estos parámetros también son variables del modelo cuyo valor no se conoce. De esta forma, se puede hacer inferencia sobre los parámetros haciendo inferencia sobre estas nuevas variables aleatorias. La distribución conjunta de este nuevo modelo extendido se factoriza como:

$$\begin{aligned} p(x, \theta) &= p(\theta)\, p(x \,|\, \theta) \\ &= p(\theta) \prod_{n=1}^{N} p(x_n \,|\, \mathrm{Pa}(x_n), \theta_n) \end{aligned} \tag{23}$$

La distribución $p(\theta)$ es un **prior** sobre los parámetros (recordar que se están tratando los parámetros como variables aleatorias), el cual se puede considerar uniforme ($p(\theta) \propto 1$) si no se quiere sesgar la estimación de los parámetros hacia ningún valor. La cantidad $p(x \,|\, \theta)$ se llama **verosimilitud** (de los parámetros) e indica la probabilidad de generar la muestra observada x cuando se considera el vector $\theta = (\theta_1, ..., \theta_N)$ como los parámetros de la red bayesiana. De esta forma, se puede obtener la distribución posterior $p(\theta \,|\, x)$ para los parámetros $\theta \in \mathbb{R}^P$ luego de conocer el valor de la variable conocida x. Por regla de Bayes:

$$p(\theta \,|\, x) = \frac{p(x \,|\, \theta)\, p(\theta)}{p(x)}, \tag{24}$$

donde $p(x \,|\, \theta) = \prod_{n=1}^N p(x_n \,|\, \mathrm{Pa}(x_n), \theta_n)$ es la verosimilitud y la constante de normalización $p(x) = \int_{\mathbb{R}^P} p(x, \theta) \,\mathrm{d}\, \theta$ es una cantidad independiente de θ, la cual representa la probabilidad media de observar el valor conocido x a lo largo de todos los posibles modelos (cada uno determinado por su respectivo valor de θ).

De aquí en adelante se asumirá siempre que cada vector $\theta_n \in \mathbb{R}^{P_n}$ (parámetros de la distribución $p(x_n \,|\, \mathrm{Pa}(x_n))$) será estimado usando una red neuronal cuyas entradas serán los valores de las variables en $\mathrm{Pa}(x_n)$ y la salida será la estimación de la red neuronal para el vector de parámetros $\theta_n \in \mathbb{R}^{P_n}$. Como es usual, si existe alguna restricción para $\theta_n \in \mathbb{R}^{P_n}$, esta será impuesta en la última capa de la red neuronal. Por ejemplo, si θ_n debe ser un vector de probabilidad, es usual utilizar la función $\mathtt{softmax}$ en la salida, mientras que si solo es un escalar que debe estar en el intervalo $[0, 1]$ (e.g., para el parámetro de una distribución Bernoulli), es usual utilizar la función logística (llamada sigmoide en deep learning), $\sigma(x) \coloneqq \frac{1}{1+e^{-x}}$. También es usual utilizar otras funciones de activación en la salida como por ejemplo $\tanh(x) \coloneqq \frac{e^x - e^{-x}}{e^x + e^{-x}}$, la cual restringe

la salida al intervalo $[-1, 1]$ (notar que $\tanh(x) = 2\sigma(2x) - 1$), o $\mathrm{ReLU}(x) \coloneqq \max\{0, x\}$, la cual filtra los valores negativos.

2.3.1. Máximo a posteriori

Cuando se realiza inferencia bayesiana sobre el vector de parámetros $\theta \in \mathbb{R}^P$, la regla de Bayes entrega una distribución posterior $p(\theta \mid x)$, la cual corrige la distribución que asume el prior $p(\theta)$ (recordar que ahora se está tratando a θ como variable aleatoria) teniendo en consideración el valor que tomó la variable x. Sin embargo, muchas veces lo que realmente se busca es una estimación puntual $\hat{\theta} \in \mathbb{R}^P$ de los parámetros y no una distribución completa (aunque esto último es más informativo que una estimación puntual). Más aún, la estimación usualmente no se realiza utilizando una única observación $x \in \mathbb{R}^D$, sino que se utiliza un conjunto de observaciones, $\mathcal{D} = \{x^1, ..., x^K\} \subset \mathbb{R}^D$, las que se asumen como muestras provenientes desde $p(x)$. En consecuencia, la distribución posterior que se obtiene para los parámetros $\theta \in \mathbb{R}^P$ es $p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)\, p(\theta)}{p(\mathcal{D})}$, donde $p(\mathcal{D} \mid \theta)$ es la verosimilitud de θ una vez se conoce el conjunto de observaciones $\mathcal{D}$.

Un enfoque natural para elegir una estimación puntual de los parámetros θ a partir de la posterior $p(\theta \mid \mathcal{D})$ es el de **máximo a posteriori** (MAP), el cual consiste en elegir a θ como el vector de parámetros más probable luego de conocer las observaciones en $\mathcal{D}$ (i.e., como la moda de $p(\theta \mid \mathcal{D})$):

$$\begin{aligned} \hat{\theta}_{\mathrm{MAP}} &= \arg\max_{\theta} p(\theta \mid \mathcal{D}) \\ &= \arg\max_{\theta} p(\mathcal{D} \mid \theta)\, p(\theta) \end{aligned} \tag{25}$$

Es importante notar que este enfoque entrega, por primera vez, una función objetivo con la cual se pueden entrenar las redes neuronales que estiman los parámetros $\theta_n \in \mathbb{R}^{P_n}$ de cada distribución $p(x_n \mid \mathrm{Pa}(x_n), \theta_n)$ de la red bayesiana.

Por otro lado, en vez de maximizar directamente la posterior $p(\theta \mid \mathcal{D})$, es más usual maximizar $\log p(\theta \mid \mathcal{D})$. Si bien ambos problemas son equivalentes (ya que el logaritmo es estrictamente creciente), en la siguiente sección se verá que el segundo problema es más estable numéricamente, lo cual es importante para un entrenamiento estable de las redes neuronales. De esta forma, el problema de optimización para el enfoque MAP es el siguiente:

$$\hat{\theta}_{\mathrm{MAP}} = \arg\max_{\theta} \log p(\mathcal{D} \mid \theta) + \log p(\theta) \tag{26}$$

Por simplicidad, de aquí en adelante se usará la notación estándar $p_\theta(x) \coloneqq p(x \mid \theta)$ para indicar la distribución que sigue x cuando se fijan los parámetros al valor $\theta \in \mathbb{R}^P$. Del mismo modo, para cada factor de la red bayesiana, se escribirá $p_{\theta_n}(x_n \mid \mathrm{Pa}(x_n)) \coloneqq p(x_n \mid \mathrm{Pa}(x_n), \theta_n)$. Además, si una distribución no lleva un símbolo de parámetro como subíndice, se asumirá que todos sus parámetros son conocidos (i.e., son distribuciones fijas). Esto último será usual al fijar, por ejemplo, distribuciones priors como una distribución estándar (e.g., $p_\theta(z) = p(z) \sim \mathcal{N}(0, \mathrm{I}_L)$ en un modelo de variable latente).

2.3.1.1. Relación con weight decay

Reescribir la maximización de $p(\theta \mid x)$ como la maximización de $\log p(\theta \mid x)$ permite ver que algunas elecciones para el prior $p(\theta)$ son equivalentes a aplicar técnicas de regularización clásicas durante la maximización de la verosimilitud. Por ejemplo, si se considera un prior gaussiano $p(\theta) \sim \mathcal{N}\left(0, \frac{1}{\lambda}\mathrm{I}_P\right)$, con $\lambda > 0$ un hiperparámetro fijo, MAP resulta ser equivalente a maximizar la log-verosimilitud $\log p_\theta(\mathcal{D})$ incluyendo regularización l^2 sobre los parámetros (llamado **weight decay** en PyTorch), lo cual es directo al ver que los problemas de optimización son equivalentes:

$$\begin{aligned} \log p(\theta \mid \mathcal{D}) &= \log p_\theta(\mathcal{D}) + \log p(\theta) \\ &= \log p_\theta(\mathcal{D}) - \frac{\lambda}{2} \| \theta \|^2 + \text{constante}, \end{aligned} \tag{27}$$

donde se usó que $p(\theta) \propto \exp\left(-\frac{\lambda}{2} \| \theta \|^2\right)$. Se observa que el parámetro de varianza inversa (precisión), $\lambda > 0$, define la importancia que se le da al término regularizador, lo cual es esperable, ya que una varianza pequeña ($\lambda \gg 0$) concentra fuertemente su masa alrededor de la media $0 \in \mathbb{R}^P$ de la distribución prior $p(\theta)$, mientras que una varianza alta ($\lambda \approx 0$) distribuye más uniformemente la masa sobre el soporte. De forma análoga, considerar un prior de Laplace equivale a utilizar LASSO (regularización l^1) sobre los parámetros. Notar que, en ambos casos, los priors están centrados alrededor del origen sesgando al modelo a usar parámetros pequeños, lo cual se espera que disminuya tanto la complejidad del modelo (para evitar overfitting) como su varianza (para un entrenamiento más estable).

También es posible hacer otras elecciones de priors convenientes para la optimización. Una alternativa usual en machine learning clásico es el uso de priors conjugados, donde se elige un prior $p(\theta)$ con una forma específica tal que la posterior $p(\theta \mid \mathcal{D}) \propto p_\theta(\mathcal{D})\, p(\theta)$ pertenezca a la misma familia, lo que resulta conveniente para el cálculo de productos que entrega la regla de Bayes. Sin embargo, si bien este enfoque suele entregar interpretabilidad al modelo, no se suele usar en modelos de IA generativa debido a que se prefiere elegir, implícitamente, un prior gaussiano al regularizar el entrenamiento usando weight decay.

Por último, es importante mencionar que algunas veces, sobre todo en la comunidad de machine learning, la expresión hacer inferencia se usa para referirse a la acción de realizar una predicción con un modelo ya entrenado y no a ajustar los parámetros de un modelo, a lo cual se le suele decir entrenar el modelo. Estos dos usos para el término inferencia no son contradictorios entre sí ya que realizar una predicción con un modelo puede verse como hacer inferencia sobre variables no conocidas. Sin embargo, es importante tener presente que, al menos desde una perspectiva bayesiana, entrenar un modelo también es una forma de realizar inferencia.

2.4. Criterio de máxima verosimilitud

Cuando se utiliza un prior uniforme sobre los parámetros, $p(\theta) \propto 1$, el criterio de MAP se reduce al **criterio de máxima verosimilitud**, el cual es el enfoque estándar para la optimización de modelos probabilísticos en machine learning:

$$\hat{\theta}_{\text{MLE}} = \arg\max_\theta \log p_\theta(\mathcal{D}) \tag{28}$$

En IA generativa, todos los modelos que se estudiarán, a excepción de las GANs, se suelen entrenar siguiendo un enfoque relacionado, en algún sentido, con el enfoque de máxima verosimilitud.

Por otro lado, es usual asumir que todas las muestras en $\mathcal{D} = \{x^1, ..., x^K\}$ fueron generadas de forma independiente a partir de una distribución fija y desconocida, $\mathrm{p}_{\mathrm{data}}(x)$, por lo que se dice que las muestras en $\mathcal{D}$ son independientes e idénticamente distribuidas (i.i.d.). Esta suposición motiva a considerar siempre una hipótesis de independencia condicional entre las muestras, $x^i \perp x^j \,|\, \theta$, lo cual permite escribir $p_\theta(x^i, x^j) = p_\theta(x^i)\, p_\theta(x^j)$. Más aún, esta independencia condicional permite descomponer la log-verosimilitud sobre todo el dataset de entrenamiento, $\log p_\theta(\mathcal{D})$, como una suma de log-verosimilitudes individuales:

$$\begin{aligned} \log p_\theta(\mathcal{D}) &= \log \prod_{k=1}^{K} p_\theta\big(x^k\big) \\ &= \sum_{k=1}^{K} \log p_\theta\big(x^k\big) \end{aligned} \tag{29}$$

Notar que esta separación en una suma es uno de los principales beneficios de optimizar la log-verosimilitud en vez de la verosimilitud directamente (donde quedaría un producto de verosimilitudes). Por otro lado, la mayoría de las distribuciones comúnmente usadas en machine learning (e.g., gaussiana o exponencial) son log-cóncavas (i.e., el logaritmo de su función de densidad es una función cóncava), por lo que para maximizar la log-verosimilitud $\log p_\theta(\mathcal{D})$ basta derivar e igualar a cero (i.e., la condición de 1º orden necesaria para la optimalidad también es suficiente). Además, dado que las probabilidades están siempre en el intervalo $[0, 1]$, el producto de muchas probabilidades (e.g., en $\prod_{k=1}^{K} p_\theta\big(x^k\big)$) puede provocar problemas numéricos (y, por lo tanto, un entrenamiento inestable) al estar trabajando con números muy pequeños. Usando logaritmos, el producto se transforma en suma y el intervalo de probabilidades $(0, 1]$ se estira a todo el semieje real $(-\infty, 0]$.

Para fijar conceptos, dado un dataset $\mathcal{D}$, la función $l_{\mathcal{D}}(\theta) \coloneqq \log p_\theta(\mathcal{D})$ se denomina función de log-verosimilitud, y es una función de los parámetros $\theta \in \mathbb{R}^P$ y no de los datos observados (el conjunto de muestras $\mathcal{D}$ se considera fijo, y cada conjunto $\mathcal{D}$ define una función de log--verosimilitud distinta). Notar que la función de log-verosimilitud no es una (log-)distribución ya que, si bien $p_\theta(\mathcal{D}) \coloneqq p(\mathcal{D} \,|\, \theta)$ es una distribución sobre $\mathcal{D}$, no lo es sobre θ debido a que $\int_{\mathbb{R}^P} p_\theta(\mathcal{D})\, \mathrm{d}\, \theta \neq 1$. Por otro lado, el cálculo de la log-verosimilitud $\log p_\theta(\mathcal{D})$ puede realizarse de forma secuencial (i.e., una muestra a la vez) o de forma paralela en batches, ya que la descomposición $\log p_\theta(\mathcal{D}) = \sum_{k=1}^{K} \log p_\theta\big(x^k\big)$ permite evaluar la log-verosimilitud de forma individual en cada muestra y luego sumar las salidas. Más aún, cuando el modelo no es de variable latente (i.e., todas las variables, a excepción de los parámetros, son observadas durante el entrenamiento), la optimización de los parámetros $\theta = (\theta_1, ..., \theta_N)$ puede realizarse de forma independiente en cada nodo de la red bayesiana $p(x) = \prod_{n=1}^{N} p_{\theta_n}(x_n \,|\, \mathrm{Pa}(x_n))$, lo que permite paralelizar el proceso de entrenamiento por nodos (e.g., se podría computar cada nodo en una GPU distinta si se están usando redes neuronales). En efecto, para una muestra de entrenamiento $x = (x_1, ..., x_N) \in \mathcal{D}$, su log-verosimilitud se descompone como:

$$\begin{aligned}\log p_\theta(x) &= \log\left(\prod_{n=1}^{N} p_{\theta_n}(x_n \,|\, \mathrm{Pa}(x_n))\right) \\ &= \sum_{n=1}^{N} \log p_{\theta_n}(x_n \,|\, \mathrm{Pa}(x_n)),\end{aligned} \tag{30}$$

por lo que, para obtener el estimador de máxima verosimilitud (MLE), se puede calcular individualmente el MLE para cada parámetro $\theta_n \in \mathbb{R}^{P_n}$ mediante $\arg\max_{\theta_n} \log p_{\theta_n}(x_n \,|\, \mathrm{Pa}(x_n))$ ya que θ_n no influye en ningún otro sumando de la log-verosimilitud $\log p_\theta(x)$. Del mismo modo, si se realiza inferencia usando el conjunto de observaciones $\mathcal{D} = \{x^1, ..., x^K\}$, el MLE para el parámetro θ_n se obtiene resolviendo el problema de optimización

$$\arg\max_{\theta_n} \sum_{k=1}^{K} \log p_{\theta_n}\left(x_n^k \,\middle|\, \mathrm{Pa}(x_n)\right) \tag{31}$$

La familia de modelos descrita anteriormente se denominan **modelos totalmente observados** ya que todas las variables del modelo, $x = (x_1, ..., x_N)$, son observadas en cada una de las muestras contenidas en el dataset de entrenamiento $\mathcal{D}$. Este tipo de modelos permite obtener la densidad de una muestra de forma exacta y tratable (solo se deben sumar N términos). Más aún, el gradiente de la log-verosimilitud, $\nabla_{\theta_n} \log p_\theta(x) = \nabla_{\theta_n} \log p_{\theta_n}(x_n \,|\, \mathrm{Pa}(x_n)) \in \mathbb{R}^{P_n}$, se puede obtener fácilmente mediante backpropagation, por lo que es fácil hacer inferencia sobre los parámetros de un modelo totalmente observado usando una red neuronal entrenada mediante el criterio de máxima verosimilitud. Más aún, si se trabaja con mini-batches de entrenamiento, $\mathcal{B} \subset \mathcal{D}$, entonces la cantidad $\frac{1}{|\mathcal{B}|} \log p_\theta(\mathcal{B})$ es un estimador insesgado de la verosimilitud media, $\frac{1}{|\mathcal{D}|} \log p_\theta(\mathcal{D})$, mientras que $\frac{1}{|\mathcal{B}|} \nabla_\theta \log p_\theta(\mathcal{B})$ es un estimador insesgado del gradiente $\frac{1}{|\mathcal{D}|} \nabla_\theta \log p_\theta(\mathcal{D})$. Esta observación permite, por ejemplo, interpretar a $\nabla_\theta \log p_\theta(\mathcal{B})$ como un gradiente estocástico que le entrega aleatoriedad al modelo (donde la aleatoriedad ocurre en la elección de las muestras que forman el batch $\mathcal{B} \subset \mathcal{D}$), lo cual muchas veces ayuda a evitar que el modelo se quede atrapado en un óptimo local de la función objetivo a optimizar. Por otro lado, el entrenamiento usando batches muchas veces es impuesto por restricciones de hardware, ya que para grandes modelos es usual no poder cargar todos los datos disponibles en la GPU para realizar el entrenamiento.

En conclusión, el criterio de máxima verosimilitud, $\hat{\theta} = \arg\max_\theta \log p_\theta(\mathcal{D})$, es el enfoque más común en modelos totalmente observados como los modelos autorregresivos (usados para generar texto) o los modelos basados en flujos (usados para generar imágenes). Esta función de pérdida permite optimizar los modelos de manera estable (dentro de lo posible; los Transformers a gran escala muchas veces presentan inestabilidades durante el entrenamiento aunque se entrenen usando verosimilitud), lo cual no siempre es así. Por ejemplo, las GANs, que entrenan un objetivo que no está basado en verosimilitud, sufren de muchas inestabilidades durante su entrenamiento, principalmente por la naturaleza adversativa de su entrenamiento.

2.4.1. Ejemplos

En algunos casos simples es posible encontrar el estimador de máxima verosimilitud de forma cerrada, el cual, aparte de ser único, también suele ser interpretable y natural. En esta subsección se revisarán algunos de estos casos.

2.4.1.1. MLE para la distribución gaussiana

Si $\mathcal{D} = \{x^1, ..., x^K\} \subset \mathbb{R}$ es un dataset de muestras independientes, se puede considerar el modelo $p_\theta(x) \sim \mathcal{N}(\mu_\theta, \sigma^2)$, donde $\mu_\theta \in \mathbb{R}$ es su media (desconocida) y $\sigma^2 > 0$ es su varianza, la cual, por simplicidad, se asumirá conocida. Para buscar el estimador de máxima verosimilitud para el parámetro μ_θ notar que

$$\begin{aligned} \log p_\theta(\mathcal{D}) &= \sum_{k=1}^{K} \log p\big(x^k\big) \\ &= \sum_{k=1}^{K} \frac{-1}{2\sigma^2}\big(x^k - \mu_\theta\big)^2 + \text{constante} \end{aligned} \tag{32}$$

Para obtener $\arg\max_\theta \log p_\theta(\mathcal{D})$ se puede derivar la log-verosimilitud anterior e igualarla a 0 (condición de 1º orden):

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\ \mu_\theta} \log p_\theta(\mathcal{D}) &= \sum_{k=1}^{K} \frac{-1}{\sigma^2}\big(x^k - \mu_\theta\big) \\ &= \frac{-1}{\sigma^2}\left(\sum_{k=1}^{K} x^k - K \cdot \mu_\theta\right) \\ &= 0 \end{aligned} \tag{33}$$

Por lo que basta despejar $\mu_\theta \in \mathbb{R}$ para obtener el estimador de máxima verosimilitud:

$$\mu_\theta = \frac{1}{K}\sum_{k=1}^{K} x^k \tag{34}$$

Es decir, el MLE para la media de un modelo gaussiano es simplemente la media empírica de las muestras. Si bien se podría verificar que esta cantidad es un máximo mirando la segunda derivada (o el Hessiano en más dimensiones), aquí no es necesario porque la función a maximizar es cóncava (ya que la suma de funciones cóncavas es cóncava). Por otro lado, si también se quisiera estimar el parámetro $\sigma^2 > 0$, se puede realizar el mismo procedimiento para llegar a que el MLE de la varianza es simplemente la varianza empírica de las muestras en $\mathcal{D}$.

2.4.1.2. MLE para la distribución de Bernoulli

Si $\mathcal{D} = \{x^1, ..., x^K\} \subset \{0, 1\}$ es un dataset de muestras binarias independientes, se puede considerar el modelo $p_\theta(x) \sim \text{Bernoulli}(r_\theta)$, donde $r_\theta \in [0, 1]$ es el parámetro (desconocido) de la distribución. Al igual que antes, este parámetro puede ser estimado usando el criterio de máxima verosimilitud. Para esto, notar que:

$$\begin{aligned}\log p_\theta(\mathcal{D}) &= \sum_{k=1}^{K} \log p\big(x^k\big) \\ &= \sum_{k=1}^{K} \big(x^k \log(r_\theta) + \big(1 - x^k\big) \log(1 - r_\theta)\big)\end{aligned} \tag{35}$$

Al igual que antes, para obtener $\arg\max_\theta \log p_\theta(\mathcal{D})$ se derivará la log-verosimilitud anterior y se igualará a cero:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\, r_\theta} \log p_\theta(\mathcal{D}) &= \sum_{k=1}^{K} \left(\frac{x^k}{r_\theta} - \frac{1 - x^k}{1 - r_\theta} \right) \\ &= \frac{1}{r_\theta} \sum_{k=1}^{K} x^k - \frac{1}{1 - r_\theta} \sum_{k=1}^{K} \big(1 - x^k\big) \\ &= 0\end{aligned} \tag{36}$$

Por lo que basta despejar $r_\theta \in [0, 1]$ para obtener el estimador de máxima verosimilitud del parámetro desconocido:

$$\begin{aligned}&(1 - r_\theta) \sum_{k=1}^{K} x^k - r_\theta \sum_{k=1}^{K} \big(1 - x^k\big) = 0 \\ &\implies \sum_{k=1}^{K} x^k = r_\theta \left(\sum_{k=1}^{K} x^k + \sum_{k=1}^{K} \big(1 - x^k\big) \right) = r_\theta \cdot K \\ &\implies r_\theta = \frac{1}{K} \sum_{k=1}^{K} x^k\end{aligned} \tag{37}$$

Es decir, el MLE para una distribución de Bernoulli corresponde a la fracción de muestras con valor 1 con respecto al total de muestras en $\mathcal{D}$, lo cual tiene sentido, por ejemplo, al interpretar el experimento como lanzamiento de monedas, donde la probabilidad de obtener cara, $p(x = 1) = r_\theta$, es (cuando $K \to \infty$) la cantidad de caras que se obtuvieron sobre el total de lanzamientos.

Por otro lado, al igual que en el caso gaussiano, este MLE para r_θ se puede ver como la media empírica de $\mathcal{D}$, la cual, a su vez, puede interpretarse como un estimador de la media real $\mathbb{E}_{\mathrm{p}_{\mathrm{data}}(x)}[x] = r_{\mathrm{data}}$ (asumiendo que $\mathrm{p}_{\mathrm{data}}(x) \sim \mathrm{Bernoulli}(r_{\mathrm{data}})$, con $r_{\mathrm{data}} \in [0, 1]$ desconocido).

2.4.1.3. MLE para la regresión lineal

Para un dataset supervisado $\mathcal{D} = \{(x^1, y^1), ..., (x^K, y^K)\} \subset \mathbb{R}^D \times \mathbb{R}$ (donde se asume que $y^k \sim \mathrm{p}_{\mathrm{data}}\big(y^k \,|\, x^k\big)$ para una entrada $x^k \in \mathbb{R}^D$ dada), se puede considerar el modelo gaussiano lineal, $p(y \,|\, x) \sim \mathcal{N}(\theta^\top x, \sigma^2)$ (i.e., $y = \theta^\top x + \epsilon$, con $\epsilon \sim \mathcal{N}(0, \sigma^2)$ un error gaussiano aditivo), donde $\theta \in \mathbb{R}^D$ es el parámetro desconocido a estimar y $\sigma^2 > 0$ es un hiperparámetro fijo (aunque también podría ser estimado por máxima verosimilitud). Al igual que antes, se puede obtener el estimador de máxima verosimilitud para $\theta \in \mathbb{R}^D$, el cual coincide con el regresor lineal óptimo para mínimos cuadrados, $\hat{\theta} = (X^\top X)^{-1} X^\top Y \in \mathbb{R}^D$, donde $X \in \mathcal{M}_{K,D}(\mathbb{R})$ es la matriz cuya k-ésima fila corresponde a la muestra $x^k \in \mathbb{R}^D$, mientras que $Y \in \mathbb{R}^K$ es el vector cuya k-ésima coordenada es $y^k \in \mathbb{R}$. Notar que, al igual que antes, este estimador de máxima verosimilitud también es natural ya

que la matriz $X^\dagger := (X^\top X)^{-1} X^\top \in \mathcal{M}_{D,K}(\mathbb{R})$ corresponde a la pseudoinversa de Moore-Penrose de la matriz $X \in \mathcal{M}_{K,D}(\mathbb{R})$, por lo que $\hat{\theta} = X^\dagger Y$ puede verse como un intento de despejar $\theta \in \mathbb{R}^D$ en un sistema lineal sobredeterminado, $Y = X\theta$. Más aún, en línea con lo revisado en la sección anterior, si se agrega un prior gaussiano al parámetro desconocido $\theta \in \mathbb{R}^D$, el estimador MAP coincide con el regresor lineal de mínimos cuadrados con regularización cuadrática (i.e., ridge regression).

2.4.2. Verosimilitud en modelos de variable latente

La principal dificultad de entrenar un modelo de variable latente por máxima verosimilitud es que la cantidad $\log p_\theta(x)$ es costosa de computar. En efecto, para una red bayesiana de variable latente, $p_\theta(x, z) = p_\theta(z)\, p_\theta(x \mid z)$, el cálculo de la log-verosimilitud $\log p_\theta(x)$ requiere marginalizar sobre la variable latente z:

$$\begin{aligned} p_\theta(x) &= \int_{\mathbb{R}^L} p_\theta(x, z)\, \mathrm{d}\ z \\ &= \int_{\mathbb{R}^L} p_\theta(z)\, p_\theta(x \mid z)\, \mathrm{d}\ z \end{aligned} \tag{38}$$

Esta integral (o suma si z es una variable latente discreta) es muy costosa de calcular, incluso aproximándola numéricamente (hay varias formas de hacerlo; una usual es escribir la igualdad anterior como $p_\theta(x) = \mathbb{E}_{p_\theta(z)}[p_\theta(x \mid z)]$, lo que permite usar una aproximación de Monte Carlo usando muestras desde $p_\theta(z)$), por lo que la verosimilitud para este tipo de modelos se considera intratable en la práctica. Esta dificultad aumenta aún más si se considera que, por lo general, la verosimilitud se debe calcular para un dataset de muestras, $\mathcal{D} = \{x^1, ..., x^K\}$, y en cada iteración del algoritmo de gradiente (asumiendo que los parámetros están siendo aprendidos por redes neuronales).

Por otro lado, la dificultad de calcular $p_\theta(x)$ se hereda al querer calcular el gradiente $\nabla_\theta \log p_\theta(x)$ (necesario para el entrenamiento de las redes neuronales) y al querer hacer inferencia sobre z mediante $p_\theta(z \mid x) = p_\theta \frac{x,z}{p_\theta}(x)$. Además, la marginalización necesaria para calcular $p_\theta(x)$ elimina la posibilidad de optimizar cada nodo del DAG de forma individual como sí ocurre en los modelos completamente observables. En efecto:

$$\begin{aligned} \log p_\theta(\mathcal{D}) &= \sum_{k=1}^{K} \log p_\theta\left(x^k\right) \\ &= \sum_{k=1}^{K} \log \left(\int_{\mathbb{R}^L} p_\theta(z)\, p_\theta\left(x^k \mid z\right) \mathrm{d}\ z \right) \\ &= \sum_{k=1}^{K} \log \left(\int_{\mathbb{R}^L} p_\theta(z) \prod_{n=1}^{N} p_{\theta_n}\left(x_n \mid \mathrm{Pa}\left(x_n^k\right)\right) \mathrm{d}\ z \right) \end{aligned} \tag{39}$$

Dado que el logaritmo no distribuye sobre la integral, la log-verosimilitud ya no se puede descomponer en una suma que separe los parámetros a optimizar, $\theta = (\theta_1, ..., \theta_N)$. Más aún, la log-verosimilitud que se obtiene es difícil de optimizar debido a su estructura, la cual además provoca que se pierda la log-concavidad que se suele tener para distribuciones sencillas. En

consecuencia, los modelos de variable latente por lo general son entrenados utilizando enfoques alternativos al de máxima verosimilitud. Por ejemplo, los VAEs y los modelos de difusión utilizan un enfoque variacional para optimizar una cota inferior de la log-verosimilitud (llamada ELBO), mientras que las GANs utilizan un entrenamiento alternado entre dos modelos que compiten entre sí (de hecho, este tipo de modelos no modela una función de verosimilitud de forma explícita).

Por otro lado, un algoritmo clásico para entrenar un modelo de variable latente genérico es el algoritmo de expectation-maximization (EM), el cual estima las variables latentes a partir de los datos (paso E) y luego realiza inferencia sobre los parámetros usando los valores estimados para las variables latentes (paso M). Otro tipo de modelos, como los modelos basados en energía, reparametrizan la verosimilitud de forma conveniente para luego poder optimizarla sin tener que calcularla de forma explícita, y luego utilizan algoritmos tipo Markov chain Monte Carlo (MCMC) para poder generar muestras.

2.4.3. Relación con teoría de la información

En esta sección se revisará la conexión y equivalencia entre el enfoque de máxima verosimilitud y algunos conceptos principales de la teoría de la información. Para esto, se comenzará construyendo la función de información de Shannon.

2.4.3.1. Información de Shannon y entropía

De manera intuitiva, se puede definir la información de un evento aleatorio como la cantidad de sorpresa o conocimiento no trivial que reporta el saber que el evento ocurrió. De esta forma, la ocurrencia de un evento poco probable (e.g., hoy hubo un terremoto) es muy informativo, mientras que eventos más comunes o de ocurrencia más esperable (e.g., mañana saldrá el sol) entregan poca información. Para definir formalmente este concepto, se construirá una función $\mathrm{I} : \mathcal{B} \to \mathbb{R}$ (con $\mathcal{B}$ el conjunto de todos los eventos posibles) que mida la cantidad de información que contiene un evento de acuerdo a su probabilidad de ocurrencia. Para esto, se considerarán los siguientes requisitos naturales sobre la función de información:

1. La información de un evento solo debe depender de su probabilidad. Es decir, para todo evento $A \in \mathcal{B}$, $\mathrm{I}(A) = f(\mathrm{Prob}(A))$ para alguna función continua $f : [0, 1] \to \mathbb{R}$.
2. La información debe ser no negativa. Además, es nula solo si el evento es completamente seguro. Es decir, para todo evento $A \in \mathcal{B}$, $\mathrm{I}(A) \geq 0$, con igualdad si y solo si $\mathrm{Prob}(A) = 1$.
3. La información que entrega el saber la ocurrencia de dos eventos independientes es la suma de las informaciones individuales. Es decir, para $A \perp B$ eventos independientes, se debe cumplir que $\mathrm{I}(A \cap B) = \mathrm{I}(A) + \mathrm{I}(B)$, donde $A \cap B \in \mathcal{B}$ es el evento de que ocurran los eventos A y B.

Para encontrar la función $f : [0, 1] \to \mathbb{R}$ correcta se puede comenzar considerando dos eventos independientes, $A \perp B$. Por los requisitos 3 y 1:

$$\begin{aligned} \mathrm{I}(A \cap B) &= \mathrm{I}(A) + \mathrm{I}(B) \\ &= f(\mathrm{Prob}(A)) + f(\mathrm{Prob}(B)) \end{aligned} \tag{40}$$

Por otra parte, como $A, B \in \mathcal{B}$ son eventos independientes, también se tiene que $\text{Prob}(A \cap B) = \text{Prob}(A)\,\text{Prob}(B)$. Por lo tanto:

$$\begin{aligned} \text{I}(A \cap B) &= f(\text{Prob}(A \cap B)) \\ &= f(\text{Prob}(A)\,\text{Prob}(B)) \end{aligned} \tag{41}$$

Luego, igualando ambas expresiones para $\text{I}(A \cap B)$, se observa que la función $f : [0,1] \to \mathbb{R}$ buscada debe cumplir que $f(x) + f(y) = f(xy)$ para $x, y \in [0,1]$. Una función continua natural que cumple esto es el logaritmo, $f(x) = c \cdot \log(x)$, para cualquier constante $c \in \mathbb{R}$. Más aún, la ecuación funcional de Cauchy permite demostrar que el logaritmo es la única función continua que cumple esta propiedad. Por lo tanto, la definición natural para la información de un evento $A \in \mathcal{B}$ es $\text{I}(A) = c \cdot \log \text{Prob}(A)$, para algún valor apropiado de $c \in \mathbb{R}$.

Por otro lado, por el requisito de positividad, y notando que $\log \text{Prob}(A) \leq 0$ para cualquier evento $A \in \mathcal{B}$, necesariamente se debe elegir una constante $c < 0$. Además, considerando esta constante de la forma $c = -\frac{1}{\log} b$ para $b > 1$, la propiedad de cambio de base de los logaritmos permite definir la **función de información de Shannon** (para una base fija $b > 1$) como

$$\text{I}(A) \coloneqq -\log_b \text{Prob}(A), \tag{42}$$

donde $A \in \mathcal{B}$ es un evento cualquiera. Cuando $b = 2$ la información se mide en bits, mientras que cuando $b = e$ (constante de Euler), la información se mide en nats. Sin embargo, la base que se utilice es irrelevante ya que las definiciones solo difieren por una constante multiplicativa. En este libro se considerará siempre $b = e$ para trabajar con el logaritmo natural, lo cual es la elección estándar en machine learning.

Teniendo definido el concepto de información de un evento, es posible definir la entropía de una distribución $p(x)$ como la información esperada para una variable $x \sim p(x)$ que sigue dicha distribución:

$$\begin{aligned} H(p) &\coloneqq \mathbb{E}_{p(x)}[\text{I}(x)] \\ &= \mathbb{E}_{p(x)}[-\log p(x)] \\ &= -\int_{\mathbb{R}^D} \log p(x)\, p(x)\, \mathrm{d}\, x, \end{aligned} \tag{43}$$

donde en la última igualdad se asumió que $x \sim p(x)$ es una variable aleatoria continua. En el caso discreto, en cambio, $H(p) = -\sum_{n=1}^{N} \log p(k_n) \cdot p(k_n)$, si se considera $\text{supp}(x) = \{k_1, ..., k_N\}$.

Recordando la interpretación de la información de Shannon dada anteriormente, la entropía de una variable aleatoria $x \sim p(x)$ se puede interpretar como el grado de aleatoriedad o incertidumbre que esta posee. Por ejemplo, para $x \sim \text{Bernoulli}(r)$, $H(p) = -r \cdot \log(r) - (1-r) \cdot \log(1-r)$. Se puede verificar que esta cantidad es máxima cuando $r = \frac{1}{2}$ (máxima incertidumbre) y es mínima cuando $r = 0$ o $r = 1$ (no hay incertidumbre). De forma similar, la distribución uniforme $x \sim \text{Uniforme}([a,b])$ es la de máxima entropía (incertidumbre) entre todas las distribuciones con soporte acotado, mientras que la distribución gaussiana es la de máxima incertidumbre entre todas las distribuciones con soporte $\mathbb{R}$ (y de varianza finita).

2.4.3.2. Entropía cruzada y divergencia KL

Teniendo definido el concepto de entropía, la entropía cruzada entre dos distribuciones, $p(x)$ y $q(x)$, se define como la información esperada de x pero cuando se considera la distribución p en el cálculo de esperanza y la distribución q en el cálculo de información:

$$\mathrm{CE}(p, q) \coloneqq \mathbb{E}_{p(x)}[-\log q(x)] = -\int_{\mathbb{R}^D} \log q(x)\, p(x)\, \mathrm{d}\ x \tag{44}$$

Si bien el concepto de entropía cruzada parece menos interpretable que el de entropía, este surge naturalmente cuando se consideran las motivaciones originales de Shannon al introducir la teoría de la información, donde el objetivo principal es poder codificar distintos símbolos (e.g., palabras de un vocabulario) de manera eficiente (símbolos frecuentes se codifican con códigos cortos, mientras que símbolos poco usuales se codifican con códigos más extensos). Bajo esta mirada, el primer teorema de Shannon indica que la entropía $H(p)$ indica cuántos bits por símbolo (considerando $b = 2$) se necesitan en promedio para codificar un mensaje (sin pérdida de información), asumiendo que los símbolos son generados según la distribución $x \sim p(x)$. De esta forma, la entropía cruzada $\mathrm{CE}(p, q)$ corresponde a la cantidad de bits que se requieren (en promedio) para codificar un símbolo cuando el esquema de codificación se optimiza asumiendo que los símbolos siguen una distribución $y \sim q(y)$, cuando en realidad siguen una distribución $x \sim p(x)$.

Por otro lado, la interpretación de compresibilidad de la entropía cruzada no es estrictamente necesaria en el estudio de redes bayesianas ya que, usualmente, la entropía cruzada solo se menciona por la relación que surge al compararla con la verosimilitud $\log p_\theta(\mathcal{D})$ para un conjunto de entrenamiento i.i.d., $\mathcal{D} = \{x^1, ..., x^K\}$. En efecto, se puede ver que buscar el estimador de máxima verosimilitud, $\max_\theta \log p_\theta(\mathcal{D})$, equivale a minimizar la entropía cruzada entre la distribución empírica $p_\mathcal{D}(x) = \frac{1}{K}\sum_{k=1}^{K} \delta_{x^k}(x)$ y la distribución del modelo que se está ajustando, $p_\theta(x)$. Para ver esto, notar que

$$\begin{aligned} \mathrm{CE}(p_\mathcal{D}, p_\theta) &= -\mathbb{E}_{p_\mathcal{D}(x)}[\log p_\theta(x)] \\ &= -\frac{1}{K}\sum_{k=1}^{K} \log p_\theta(x^k) \\ &= -\frac{1}{K}\log\left(\prod_{k=1}^{K} p_\theta(x^k)\right) \\ &= -\frac{1}{K}\log p_\theta(\mathcal{D}), \end{aligned} \tag{45}$$

por lo que $\arg\max_\theta \log p_\theta(\mathcal{D}) = \arg\min_\theta \mathrm{CE}(p_\mathcal{D}, p_\theta)$. Por otro lado, recordando que $\mathbb{E}_{\mathrm{p_{data}}(x)}[...]$ puede ser aproximada mediante una estimación de Monte Carlo usando las muestras de $\mathcal{D}$, también se puede considerar el estimador de máxima verosimilitud como una aproximación de la solución del problema $\min_\theta \mathrm{CE}(\mathrm{p_{data}}, p_\theta)$.

Por otro lado, retomando la interpretación de la entropía cruzada $\mathrm{CE}(p, q)$ como la cantidad de bits promedio que requiere codificar un símbolo considerando una distribución $y \sim q(y)$ cuando en realidad la distribución real es $x \sim p(x)$, se puede descomponer la cantidad $\mathrm{CE}(p, q)$ en un término asociado a la cantidad de información irreducible (asociada a la propia información de

$p(x)$) y en un término asociado al error de considerar como distribución de símbolos a $q(x)$ en vez de $p(x)$:

$$\begin{aligned} \mathrm{CE}(p, q) &= \mathbb{E}_{p(x)}[-\log q(x)] \\ &= \mathbb{E}_{p(x)}[-\log p(x)] + \mathbb{E}_{p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right] \\ &= H(p) + \mathrm{D}_{\mathrm{KL}}(p, q), \end{aligned} \tag{46}$$

donde $\mathrm{D}_{\mathrm{KL}}(p, q)$ es la **divergencia de Kullback-Leibler** (también llamada entropía relativa) entre $p(x)$ y $q(x)$:

$$\begin{aligned} \mathrm{D}_{\mathrm{KL}}(p, q) &:= \mathbb{E}_{p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right] \\ &= \int_{\mathbb{R}^D} \log\left(\frac{p(x)}{q(x)}\right) p(x)\, \mathrm{d}\ x \end{aligned} \tag{47}$$

Por lo tanto, el operador de Kullback-Leibler puede interpretarse como una medida de discrepancia entre las distribuciones $p(x)$ y $q(x)$, donde la comparación se realiza con el cociente $\frac{p(x)}{q(x)}$. En efecto, si las dos distribuciones son similares, entonces el cociente es cercano a 1, por lo que $\log\left(\frac{p(x)}{q(x)}\right) \approx 0$. En particular, $\mathrm{D}_{\mathrm{KL}}(p, q) = 0$ cuando $p = q$.

Por otro lado, se puede probar que la divergencia de Kullback-Leibler es siempre no negativa (esto se conoce como desigualdad de Gibbs). Más aún, dado que $\mathrm{CE}(p_{\mathcal{D}}, p_\theta) = H(p_{\mathcal{D}}) + \mathrm{D}_{\mathrm{KL}}(p_{\mathcal{D}}, p_\theta)$, con $H(p_{\mathcal{D}}) \geq 0$ una constante independiente de θ, se puede obtener la siguiente equivalencia al enfoque de máxima verosimilitud:

$$\arg\max_\theta \log p_\theta(\mathcal{D}) = \arg\min_\theta \mathrm{D}_{\mathrm{KL}}(p_{\mathcal{D}}, p_\theta) \tag{48}$$

En resumen, buscar el estimador de máxima verosimilitud resulta ser equivalente a minimizar la entropía cruzada $\mathrm{CE}(p_{\mathcal{D}}, p_\theta)$, lo que a su vez resulta ser equivalente a minimizar la divergencia $\mathrm{D}_{\mathrm{KL}}(p_{\mathcal{D}}, p_\theta)$. Por otra parte, si bien la divergencia de Kullback-Leibler puede verse como una medida de discrepancia entre dos distribuciones de probabilidad, no es una función de distancia (o métrica) en el sentido estricto. En particular, no es simétrica ($\mathrm{D}_{\mathrm{KL}}(p, q) \neq \mathrm{D}_{\mathrm{KL}}(q, p)$) ni cumple la desigualdad triangular ($\mathrm{D}_{\mathrm{KL}}(p, q) \neq \mathrm{D}_{\mathrm{KL}}(p, r) + \mathrm{D}_{\mathrm{KL}}(r, q)$). La siguiente figura muestra cómo cambia el argumento minimizante dependiendo del orden que se utilice para las distribuciones dentro de la divergencia:

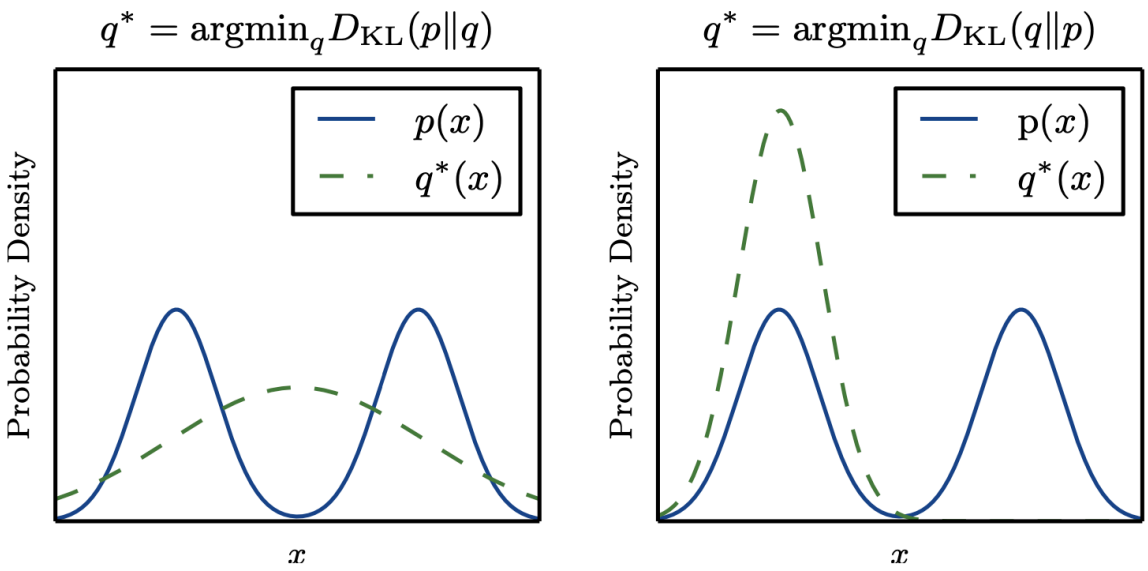

Figura 16: Efecto del orden de los argumentos en la divergencia de Kullback-Leibler sobre el argumento minimizante [17].

En particular, para el caso de la izquierda se observa la creencia usual de que optimizar un modelo generativo maximizando la log-verosimilitud fuerza al modelo a capturar todas las modas de la distribución de los datos de entrenamiento.

El hecho de que la divergencia de Kullback-Leibler no sea una distancia no permite aplicar distintos resultados conocidos sobre espacios métricos, lo cual muchas veces es útil para obtener garantías teóricas (e.g., convergencia) sobre los temas estudiados. Además, si bien es posible adaptar la divergencia de Kullback-Leibler para construir la distancia de Jensen-Shannon (la cual aparece en el estudio de las GANs), $d_{\text{JS}}(p,q)^2 = \frac{1}{2}\text{D}_{\text{KL}}\big(p, \frac{p+q}{2}\big) + \frac{1}{2}\text{D}_{\text{KL}}\big(q, \frac{p+q}{2}\big)$, esta distancia parece un poco forzada y tampoco tiene muy buenas propiedades como la distancia de Wasserstein, la cual se definirá al revisar algunos tópicos de transporte óptimo. Por otro lado, si bien la divergencia de Kullback-Leibler no es una distancia, sí es una función de divergencia, lo que permitirá definir las f-GANs, las cuales pueden verse como una generalización de las GANs para distintos funcionales de optimización.

2.4.4. Ejemplo usando una red neuronal

En esta última subsección se implementará un modelo probabilístico de juguete para poder aproximar un estimador de máxima verosimilitud usando una red neuronal. Las librerías que se usarán son las siguientes:

```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
```

Como red bayesiana se usará una mixtura (modelo de variable latente $p(x,z) = p(z)\,p(x \mid z)$, donde z es una variable categórica) de dos componentes gaussianos unidimensionales, es decir:

$$p(x) = p(z=0)\,p_0(x) + p(z=1)\,p_1(x), \tag{49}$$

donde $p_0(x) \sim \mathcal{N}(\mu_0, \sigma_0^2)$ y $p_1(x) \sim \mathcal{N}(\mu_1, \sigma_1^2)$, mientras que $p(z) \sim \text{Bernoulli}(r)$, con $r \in [0, 1]$ un parámetro que indica el peso asignado a cada distribución. En consecuencia:

$$p(x) = (1-r) \cdot \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{(x-\mu_0)^2}{2\sigma_0^2}\right) + r \cdot \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right) \tag{50}$$

La siguiente clase `Mixture` implementa esta red bayesiana, asumiendo todos los parámetros como conocidos:

```
class Mixture:

    def __init__(self, r: float, gaussians: list[tuple[float, float]]) -> None:
        self.r = r
        self.gaussians = gaussians  # (mu_0,sigma_0), (mu_1,sigma_1).

    def generate_data(self, n_samples: int = 1000) -> torch.Tensor:
        r = torch.full([n_samples], self.r)
        z = torch.bernoulli(r)
        x = torch.empty(n_samples)
        x[z == 0] = torch.normal(*self.gaussians[0], [(z == 0).sum()])
        x[z == 1] = torch.normal(*self.gaussians[1], [(z == 1).sum()])
        return x

    @staticmethod
    def p_x(x: torch.Tensor, r: float | torch.Tensor, gaussians: list[tuple[float,
float]]) -> torch.Tensor:
        mu = gaussians[0][0], gaussians[1][0]
        var = gaussians[0][1]**2, gaussians[1][1]**2

        p0 = (2*torch.pi*var[0])**(-1/2) * torch.exp(-1/(2*var[0]) * (x-mu[0])**2)
        p1 = (2*torch.pi*var[1])**(-1/2) * torch.exp(-1/(2*var[1]) * (x-mu[1])**2)
        p = (1 - r) * p0 + r * p1
        return p
```

El método `generate_data` genera muestras por ancestral sampling: primero decide desde qué clase generará cada muestra (sampleando desde $p(z) \sim \text{Bernoulli}(r)$), y luego genera las muestras observadas desde las respectivas distribuciones gaussianas. En este caso, se utilizarán como parámetros reales a $(\mu_0, \sigma_0) = (-1, 1)$ y a $(\mu_1, \sigma_1) = (3, 2)$, mientras que como prior de clase se fijará $r = 0.8$. Con estos parámetros fijos, se generarán $|\mathcal{D}| = 1000$ muestras desde la red bayesiana:

```
r = 0.8
gaussians = [(-1, 1), (3, 2)]

# Datos:
dataset = Mixture(r, gaussians)
x = dataset.generate_data()

# Gráfico:
x_plot = torch.linspace(min(x), max(x), 1000)
density = Mixture.p_x(x_plot, r, gaussians)

plt.figure(figsize=(8, 5))
```

```
plt.plot(x_plot, density)
plt.hist(x, bins=100, density=True, alpha=0.3)
plt.xlabel('x')
plt.ylabel('Densidad')
plt.grid(alpha=0.3)
plt.show()
```

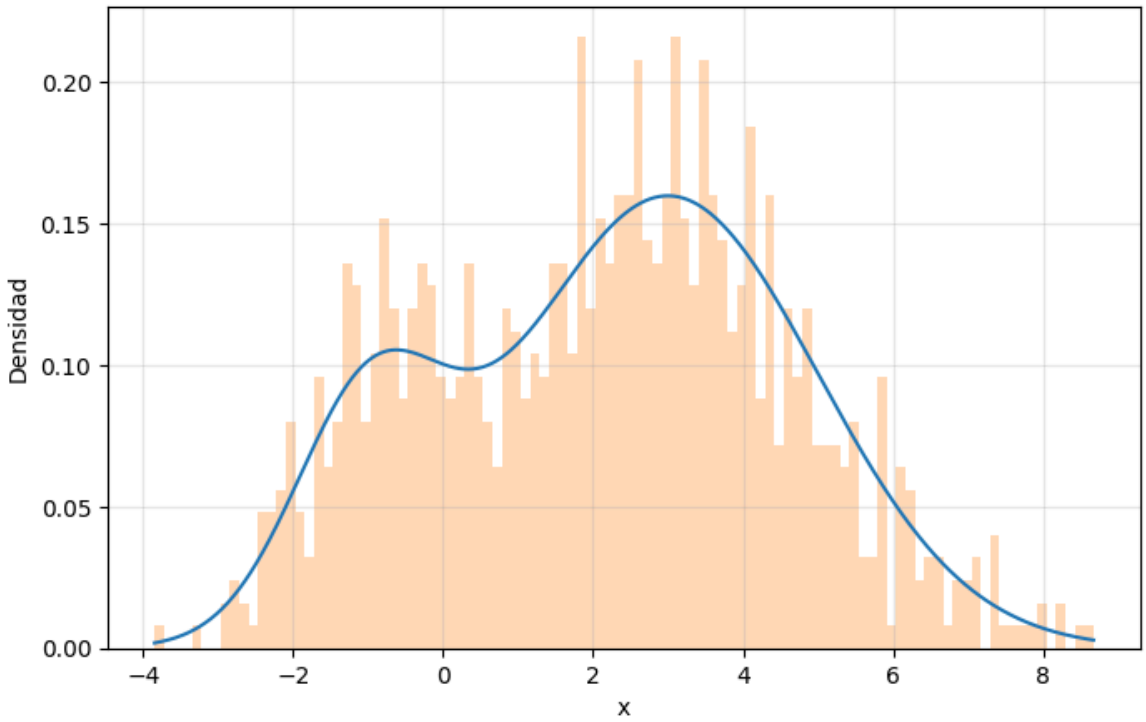

Figura 17: Densidad teórica de la mixtura gaussiana (curva) junto al histograma de las $|\mathcal{D}| = 1000$ muestras generadas por ancestral sampling.

En el gráfico de las muestras se aplicó KDE, lo que permite, en este caso, identificar claramente los valores de los parámetros μ_0 y μ_1. Para ejemplificar el problema de inferencia, se supondrá que el prior de clase r es desconocido, mientras que el resto de parámetros se considerarán conocidos por simplicidad. De esta forma, se usarán las muestras en $\mathcal{D} \subset \mathbb{R}$ para inferir el valor del parámetro $r \in [0, 1]$ usando una red neuronal r_θ con salida sigmoidal. Notar que la distribución $p_\theta(z)$ en $p_\theta(x, z) = p_\theta(z)\, p(x \mid z)$ es incondicional, por lo que la red neuronal no recibe ninguna entrada:

```
class PriorEstimator(nn.Module):

    def __init__(self, init_val: float = 0.5) -> None:
        super().__init__()
        self.logit_r = nn.Parameter(torch.tensor(init_val))

    def forward(self) -> torch.Tensor:
        return torch.sigmoid(self.logit_r)
```

La red neuronal $r_\theta \in [0, 1]$ será optimizada para maximizar la verosimilitud (usando el método `p_x` definido en la clase `Mixture`). Para esto, un detalle importante y que es usado en la práctica es que en vez de maximizar directamente la log-verosimilitud $\log p_\theta(\mathcal{D})$, se suele optimizar la cantidad $\frac{1}{|\mathcal{D}|} \log p_\theta(\mathcal{D})$ ya que al normalizar la log-verosimilitud por el tamaño del dataset de

entrenamiento, esta cantidad se vuelve independiente de $|\mathcal{D}|$, lo que evita tener que ajustar la tasa de aprendizaje de acuerdo al tamaño de cada dataset (o batch) de entrenamiento:

```
def train_loop(model: PriorEstimator, optimizer: optim.Optimizer, x: torch.Tensor,
n_epochs: int = 500) -> tuple[list[float], list[float]]:

    normalized_nlls, r_preds = [], []

    for epoch in range(n_epochs):
        r_pred = model()
        p_x = Mixture.p_x(x, r_pred, gaussians)
        normalized_nll = - p_x.log().mean()

        optimizer.zero_grad()
        normalized_nll.backward()
        optimizer.step()

        normalized_nlls.append(normalized_nll.item())
        r_preds.append(r_pred.item())

    return normalized_nlls, r_preds
```

En la función `train_loop` se está almacenando y retornando la dinámica de entrenamiento completa para luego poder visualizarla. Para realizar el entrenamiento solo falta instanciar la red neuronal y un optimizador asociado a su único parámetro:

```
model = PriorEstimator()
optimizer = optim.SGD(model.parameters(), lr=1)

train_loop(model, optimizer, x)

print(f'Estimación: {model().item():.4f}')
print(f'Valor real: {dataset.r}')
```

Estimación: 0.7847 Valor real: 0.8

Se observa que el valor predicho para el parámetro del prior $p_\theta(z) \sim \text{Bernoulli}(r_\theta)$ es cercano al valor real. De hecho, la convergencia puede ocurrir en pocas iteraciones, dependiendo de la tasa de aprendizaje que se utilice:

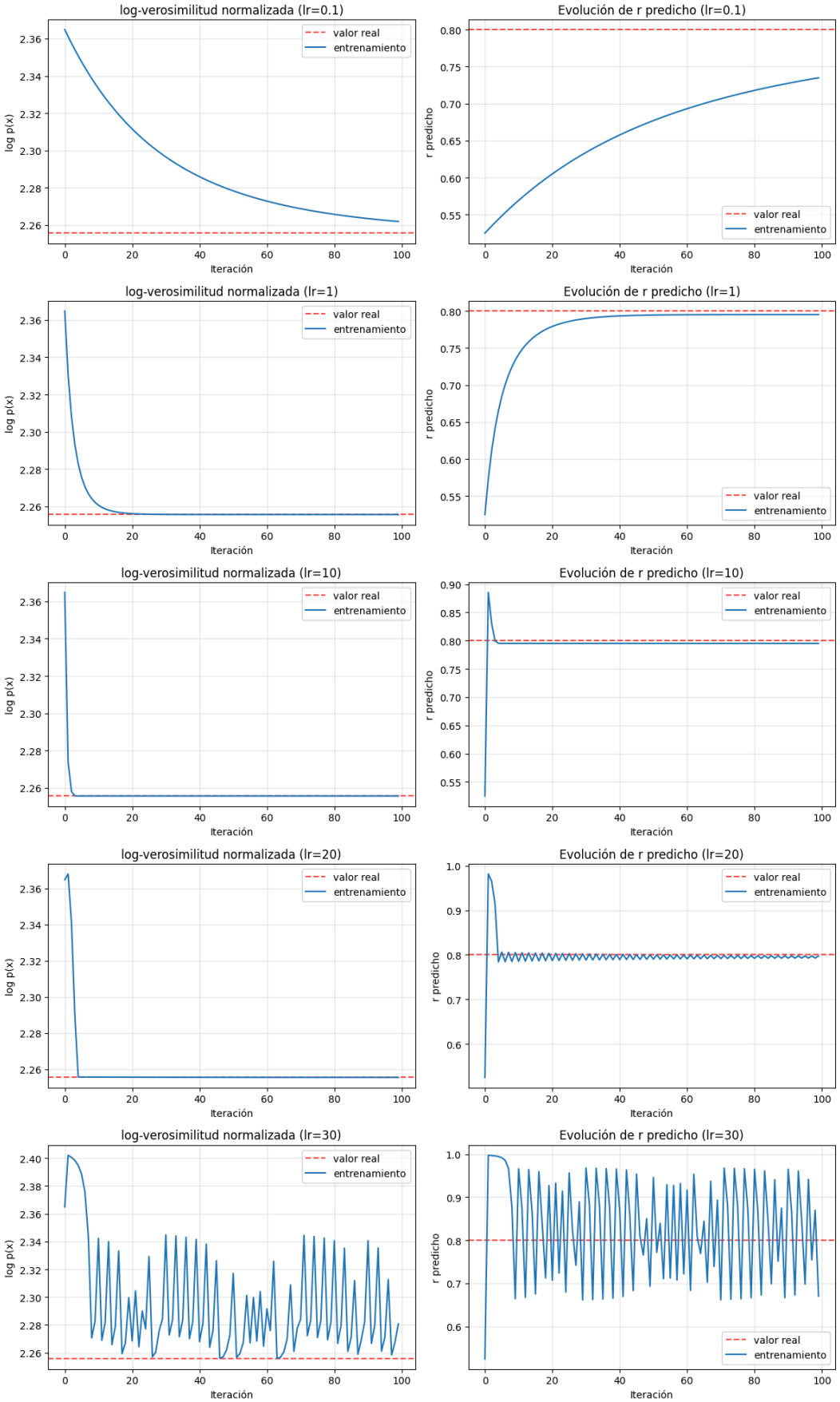

Figura 18: Dinámica de entrenamiento de r_θ para distintos valores de la tasa de aprendizaje: evolución de la log-verosimilitud negativa normalizada (columna izquierda) y del parámetro predicho (columna derecha).

- Código para la figura.

```
learning_rates = [0.1, 1, 10, 20, 30]
n_rows = len(learning_rates)

fig, axes = plt.subplots(n_rows, 2, figsize=(12, n_rows * 4))

for i, lr in enumerate(learning_rates):

    # Entrenamiento:
    model = PriorEstimator(init_val=0.1)
    optimizer = optim.SGD(model.parameters(), lr=lr)
    normalized_nlls, r_preds = train_loop(model, optimizer, x, n_epochs=100)

    # Función de pérdida (NLL normalizado):
    p_x = Mixture.p_x(x, r, gaussians)
    normalized_nll = - p_x.log().mean()

    # Gráfico loss:
    axes[i, 0].axhline(y=normalized_nll, linestyle='--', color='r', alpha=0.7,
label='valor real')
    axes[i, 0].plot(normalized_nlls, label='entrenamiento')
    axes[i, 0].set_xlabel('Iteración')
    axes[i, 0].set_ylabel('- log p(x)')
    axes[i, 0].grid(alpha=0.3)
    axes[i, 0].set_title(f'log-verosimilitud normalizada (lr={lr})')
    axes[i, 0].legend()

    # Gráfico r:
    axes[i, 1].axhline(y=r, linestyle='--', color='r', alpha=0.7,label='valor
real')
    axes[i, 1].plot(r_preds, label='entrenamiento')
    axes[i, 1].set_xlabel('Iteración')
    axes[i, 1].set_ylabel('r predicho')
    axes[i, 1].grid(alpha=0.3)
    axes[i, 1].set_title(f'Evolución de r predicho (lr={lr})')
    axes[i, 1].legend()

plt.tight_layout()
plt.show()
```

Se observa que la dinámica de entrenamiento varía considerablemente dependiendo del valor de la tasa de aprendizaje. Una tasa de aprendizaje pequeña puede provocar un entrenamiento lento, mientras que una tasa de aprendizaje muy alta puede provocar una dinámica de entrenamiento muy errática al estar variando demasiado los parámetros en cada iteración. Esto se observa en los últimos gráficos de la columna derecha, donde el parámetro $r_\theta \in [0, 1]$ oscila con alta amplitud alrededor del valor real.

Este ejemplo muestra que incluso los modelos generativos de juguete (como este, que es de un solo parámetro) pueden tener una dinámica de entrenamiento inestable si no se ajustan bien los hiperparámetros. En modelos más grandes, este ajuste puede ser aún más delicado, por lo que es usual usar heurísticas para ajustar su valor.

[1] A. Vaswani *et al.*, «Attention Is All You Need», en *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. [En línea]. Disponible en: https://arxiv.org/abs/1706.03762

[2] I. Sutskever, O. Vinyals, y Q. V. Le, «Sequence to Sequence Learning with Neural Networks», en *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. [En línea]. Disponible en: https://arxiv.org/abs/1409.3215

[3] A. Ramesh *et al.*, «Zero-Shot Text-to-Image Generation», en *International Conference on Machine Learning (ICML)*, 2021. [En línea]. Disponible en: https://arxiv.org/abs/2102.12092

[4] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, y B. Ommer, «High-Resolution Image Synthesis with Latent Diffusion Models», en *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. [En línea]. Disponible en: https://arxiv.org/abs/2112.10752

[5] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, y M. Chen, «Hierarchical Text-Conditional Image Generation with CLIP Latents», *arXiv preprint arXiv:2204.06125*, 2022, [En línea]. Disponible en: https://arxiv.org/abs/2204.06125

[6] J. Betker y others, «Improving Image Generation with Better Captions (DALL-E 3)», technical report, 2023. [En línea]. Disponible en: https://cdn.openai.com/papers/dall-e--3.pdf

[7] P. Isola, J.-Y. Zhu, T. Zhou, y A. A. Efros, «Image-to-Image Translation with Conditional Adversarial Networks», en *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. [En línea]. Disponible en: https://arxiv.org/abs/1611.07004

[8] J.-Y. Zhu, T. Park, P. Isola, y A. A. Efros, «Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks», en *IEEE International Conference on Computer Vision (ICCV)*, 2017. [En línea]. Disponible en: https://arxiv.org/abs/1703.10593

[9] Black Forest Labs, «FLUX.1 Kontext: Flow Matching for In-Context Image Generation», *arXiv preprint arXiv:2506.15742*, 2025, [En línea]. Disponible en: https://arxiv.org/abs/2506.15742

[10] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, y I. Sutskever, «Robust Speech Recognition via Large-Scale Weak Supervision», en *International Conference on Machine Learning (ICML)*, 2023. [En línea]. Disponible en: https://arxiv.org/abs/2212.04356

[11] W. Peebles y S. Xie, «Scalable Diffusion Models with Transformers», en *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. [En línea]. Disponible en: https://arxiv.org/abs/2212.09748

[12] E. Perez, F. Strub, H. de Vries, V. Dumoulin, y A. Courville, «FiLM: Visual Reasoning with a General Conditioning Layer», en *AAAI Conference on Artificial Intelligence*, 2018. [En línea]. Disponible en: https://arxiv.org/abs/1709.07871

[13] Z. Liu, P. Luo, X. Wang, y X. Tang, «Large-scale CelebFaces Attributes (CelebA) Dataset». [En línea]. Disponible en: https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[14] scikit-learn developers, «sklearn.datasets.make_swiss_roll». [En línea]. Disponible en: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_swiss_roll.html

[15] A. Radford, L. Metz, y S. Chintala, «Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks», *arXiv preprint arXiv:1511.06434*, 2015, [En línea]. Disponible en: https://arxiv.org/abs/1511.06434

[16] T. White, «Sampling Generative Networks», *arXiv preprint arXiv:1609.04468*, 2016, [En línea]. Disponible en: https://arxiv.org/abs/1609.04468

[17] I. Goodfellow, Y. Bengio, y A. Courville, *Deep Learning*. MIT Press, 2016. [En línea]. Disponible en: https://www.deeplearningbook.org/