

Parte III

Investigación de aprendizaje profundo

Esta parte del libro describe los enfoques más ambiciosos y avanzados para el aprendizaje profundo, actualmente perseguidos por la comunidad investigadora.

En las partes anteriores del libro, hemos mostrado cómo resolver problemas de aprendizaje supervisado: cómo aprender a mapear un vector con otro, dados suficientes ejemplos de mapeo.

No todos los problemas que podríamos querer resolver entran en esta categoría. Es posible que deseemos generar nuevos ejemplos, o determinar qué tan probable es algún punto, o manejar los valores faltantes y aprovechar un gran conjunto de ejemplos sin etiquetar o ejemplos de tareas relacionadas. Una deficiencia del estado actual de la técnica para aplicaciones industriales es que nuestros algoritmos de aprendizaje requieren grandes cantidades de datos supervisados para lograr una buena precisión. En esta parte del libro, discutimos algunos de los enfoques especulativos para reducir la cantidad de datos etiquetados necesarios para que los modelos existentes funcionen bien y sean aplicables a una gama más amplia de tareas. Lograr estos objetivos generalmente requiere alguna forma de aprendizaje no supervisado o semi-supervisado.

Se han diseñado muchos algoritmos de aprendizaje profundo para abordar problemas de aprendizaje no supervisado, pero ninguno realmente ha resuelto el problema de la misma manera que el aprendizaje profundo ha resuelto en gran medida el problema del aprendizaje supervisado para una amplia variedad de tareas. En esta parte del libro, describimos los enfoques existentes para el aprendizaje no supervisado y algunas de las ideas populares sobre cómo podemos progresar en este campo.

Una causa central de las dificultades del aprendizaje no supervisado es la alta dimensionalidad de las variables aleatorias que se modelan. Esto trae dos desafíos distintos: un desafío estadístico y un desafío computacional. El desafío estadístico con respecto a la generalización: la cantidad de configuraciones que podemos querer distinguir puede crecer exponencialmente con la cantidad de dimensiones de interés, y esto rápidamente se vuelve mucho mayor que la cantidad de ejemplos que uno puede tener (o usar con recursos computacionales limitados). El desafío computacional asociado con distribuciones de alta dimensión surge porque muchos algoritmos para aprender o usar un modelo entrenado (especialmente aquellos basados en estimar una función de probabilidad explícita) involucran cálculos intratables que crecen exponencialmente con el número de dimensiones.

Con los modelos probabilísticos, este desafío computacional surge de la necesidad de realizar una inferencia intratable o simplemente de la necesidad de normalizar la distribución.

- *inferencia intratable*: la inferencia se analiza principalmente en el capítulo 19. Se trata de la cuestión de adivinar los valores probables de algunas variables a , dadas otras variables b , con respecto a un modelo que captura la distribución conjunta sobre

a, b y c. Para incluso calcular tales probabilidades condicionales, es necesario sumar los valores de las variables c, así como calcular una constante de normalización que sume los valores de a y c.

- *Constantes de normalización intratables (la función de partición)*: la función de partición se analiza principalmente en el capítulo 18. Las constantes de normalización de las funciones de probabilidad surgen en la inferencia (arriba) así como en el aprendizaje. Muchos modelos probabilísticos implican una constante de normalización de este tipo. Desafortunadamente, aprender tal modelo a menudo requiere calcular el gradiente del logaritmo de la función de partición con respecto a los parámetros del modelo. Ese cálculo es generalmente tan intratable como calcular la función de partición en sí. Métodos de la cadena de Markov de Monte Carlo (MCMC) (capítulo 17) se utilizan a menudo para tratar con la función de partición (calculándola o su gradiente). Desafortunadamente, los métodos MCMC sufren cuando los modos de distribución del modelo son numerosos y están bien separados, especialmente en espacios de alta dimensión (sección 17.5).

Una forma de enfrentar estos cálculos intratables es aproximarlos, y se han propuesto muchos enfoques como se analiza en esta tercera parte del libro. Otra forma interesante, también discutida aquí, sería evitar estos cálculos intratables por completo por diseño, y los métodos que no requieren tales cálculos son, por lo tanto, muy atractivos. En los últimos años se han propuesto varios modelos generativos con esa motivación. Una amplia variedad de enfoques contemporáneos para el modelado generativo se discuten en el capítulo 20.

Parte **tercero** es el más importante para un investigador, alguien que quiere comprender la amplitud de perspectivas que se han llevado al campo del aprendizaje profundo e impulsar el campo hacia la verdadera inteligencia artificial.

Capítulo 13

Modelos de factores lineales

Muchas de las fronteras de la investigación en el aprendizaje profundo implican la construcción de un modelo probabilístico de la entrada, $pag_{\text{modelo}}(X)$. Tal modelo puede, en principio, usar la inferencia probabilística para predecir cualquiera de las variables en su entorno dada cualquiera de las otras variables. Muchos de estos modelos también tienen variables latentes h , con $pag_{\text{modelo}}(X) = \text{mi } h \text{ pag}_{\text{modelo}}(x / h)$. Estas variables latentes proporcionan otro medio de representar los datos. Las representaciones distribuidas basadas en variables latentes pueden obtener todas las ventajas del aprendizaje de representaciones que hemos visto con feedforward profundo y redes recurrentes.

En este capítulo, describimos algunos de los modelos probabilísticos más simples con variables latentes: modelos de factores lineales. Estos modelos se utilizan a veces como bloques de construcción de modelos mixtos (Hinton *et al.*, 1995a; Ghahramani y Hinton, 1996; Rowe *et al.*, 2002) o modelos probabilísticos profundos más grandes (Espiga *et al.*, 2012). También muestran muchos de los enfoques básicos necesarios para construir modelos generativos que los modelos profundos más avanzados ampliarán aún más.

Un modelo de factor lineal se define mediante el uso de una función decodificadora lineal estocástica que genera X añadiendo ruido a una transformación lineal de h .

Estos modelos son interesantes porque nos permiten descubrir factores explicativos que tienen una distribución conjunta simple. La simplicidad de usar un decodificador lineal hizo de estos modelos algunos de los primeros modelos de variables latentes que se estudiaron ampliamente.

Un modelo de factor lineal describe el proceso de generación de datos de la siguiente manera. Primero, muestreamos los factores explicativos h de una distribución

$$h \sim pag(h), \quad (13.1)$$

dónde $pag(h)$ es una distribución factorial, $\text{con } pag(h) = \prod_i pag(h_i)$, para que sea fácil

muestra de A continuación muestreamos las variables observables de valor real dados los factores:

$$X = \mathcal{Q}h + b + \text{ruido} \quad (13.2)$$

donde el ruido es típicamente gaussiano y diagonal (independiente entre dimensiones). Esto se ilustra en la figura 13.1.

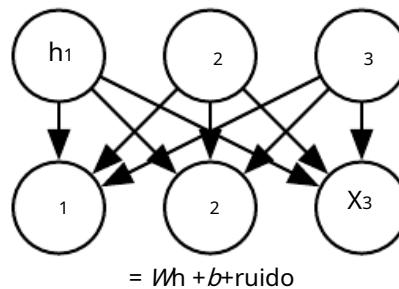


Figura 13.1: El modelo gráfico dirigido que describe la familia de modelos de factores lineales, en la que asumimos que un vector de datos observado X se obtiene mediante una combinación lineal de factores latentes independientes h , además de algo de ruido. Los diferentes modelos, como el PCA probabilístico, el análisis factorial o el ICA, toman decisiones diferentes sobre la forma del ruido y del anterior $p_{\text{ag}}(h)$.

13.1 PCA probabilístico y análisis factorial

El PCA probabilístico (análisis de componentes principales), el análisis factorial y otros modelos de factores lineales son casos especiales de las ecuaciones anteriores (13.1 y 13.2) y solo difieren en las elecciones realizadas para la distribución del ruido y las variables previas del modelo sobre las latentes h antes de observar X .

En **análisis factorial** (Bartolomé, 1987; basilevski, 1994), la variable latente anterior es solo la varianza unitaria gaussiana

$$h \sim \text{norte}(h; 0, I) \quad (13.3)$$

mientras que las variables observadas X se supone que son **condicionalmente independiente**, dado h . Específicamente, se supone que el ruido se extrae de una distribución gaussiana de covarianza diagonal, con matriz de covarianza $\psi = \text{diag}(\sigma^2)$, con $\sigma^2 = [\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2]$ - un vector de varianzas por variable.

El papel de las variables latentes es, por lo tanto, *capturar las dependencias* entre las diferentes variables observadas X_i . De hecho, se puede demostrar fácilmente que X es simplemente una variable aleatoria normal multivariante, con

$$X \sim \text{norte}(X; \text{segundo}, WW + \psi). \quad (13.4)$$

Para proyectar PCA en un marco probabilístico, podemos hacer una ligera modificación al modelo de análisis factorial, haciendo que las varianzas condicionales σ_2 iguales entre sí. En ese caso la covarianza de X es solo $WW + \sigma_2 I$, donde σ_2 ahora es un escalar. Esto produce la distribución condicional

$$X \sim \text{norte}(X; \text{segundo}, WW + \sigma_2 I) \quad (13.5)$$

o equivalente

$$x = Wh + b + \sigma z \quad (13.6)$$

dónde $\sim \text{norte}(z; 0, I)$ es ruido gaussiano. [Propinas y obispo\(1999\)](#) luego muestran un algoritmo EM iterativo para estimar los parámetros W y σ_2 .

Este **PCA probabilístico** El modelo aprovecha la observación de que la mayoría de las variaciones en los datos pueden ser capturadas por las variables latentes h , hasta algún pequeño residuo **error de reconstrucción** σ_2 . Como se muestra [Propinas y obispo\(1999\)](#), el PCA probabilístico se convierte en PCA como $\sigma \rightarrow 0$. En ese caso, el valor esperado condicional de h dado x se convierte en una proyección ortogonal de $x - \text{segundo}$ en el espacio ocupado por las columnas de W , como en PCA.

Como $\sigma \rightarrow 0$, el modelo de densidad definido por PCA probabilístico se vuelve muy nítido alrededor de estos d dimensiones abarcadas por las columnas de W . Esto puede hacer que el modelo asigne una probabilidad muy baja a los datos si los datos no se agrupan cerca de un hiperplano.

13.2 Análisis de componentes independientes (ICA)

El análisis de componentes independientes (ICA) se encuentra entre los algoritmos de aprendizaje de representación más antiguos ([Hérault y Ans, 1984](#); [Jutten y Hérault, 1991](#); [Común, 1994](#); [Hyvärinen, 1999](#); [Hyvärinen et al., 2001a](#); [Hinton et al., 2001](#); [Teh et al., 2003](#)). Es un enfoque para modelar factores lineales que busca separar una señal observada en muchas señales subyacentes que se escalan y se suman para formar los datos observados. Estas señales están destinadas a ser completamente independientes, en lugar de simplemente descorrelacionarse entre sí.¹

Muchas metodologías específicas diferentes se conocen como ICA. La variante que es más similar a los otros modelos generativos que hemos descrito aquí es una variante ([Pham et al., 1992](#)) que entrena un modelo generativo totalmente paramétrico. La distribución previa sobre los factores subyacentes, $p_{\text{ag}}(h)$, debe ser arreglado con anticipación por el usuario. Luego, el modelo genera de manera determinista $X = ?$ Podemos realizar un

¹Mira la sección [3.8](#) para una discusión de la diferencia entre variables no correlacionadas y variables independientes.

cambio no lineal de variables (usando la ecuación 3.47) para determinar $pag(X)$. Luego, el aprendizaje del modelo procede como de costumbre, utilizando la máxima verosimilitud.

La motivación de este enfoque es que al elegir $pag(h)$ para ser independientes, podemos recuperar factores subyacentes que estén lo más cerca posible de ser independientes. Esto se usa comúnmente, no para capturar factores causales abstractos de alto nivel, sino para recuperar señales de bajo nivel que se han mezclado. En este escenario, cada ejemplo de entrenamiento es un momento en el tiempo, cada X es la observación de un sensor de las señales mixtas, y cada h es una estimación de una de las señales originales. Por ejemplo, podríamos tener n personas hablando simultáneamente. Si tenemos n diferentes micrófonos colocados en diferentes lugares, ICA puede detectar los cambios en el volumen entre cada altavoz según lo escucha cada micrófono, y separar las señales para que cada h_i contiene una sola persona que habla claramente. Esto se usa comúnmente en neurociencia para electroencefalografía, una tecnología para registrar señales eléctricas que se originan en el cerebro. Muchos sensores de electrodos colocados en la cabeza del sujeto se utilizan para medir muchas señales eléctricas provenientes del cuerpo. Por lo general, el experimentador solo está interesado en las señales del cerebro, pero las señales del corazón y los ojos del sujeto son lo suficientemente fuertes como para confundir las mediciones tomadas en el cuero cabelludo del sujeto. Las señales llegan a los electrodos mezcladas, por lo que la ICA es necesaria para separar la firma eléctrica del corazón de las señales que se originan en el cerebro y para separar las señales en diferentes regiones del cerebro entre sí.

Como se mencionó anteriormente, son posibles muchas variantes de ICA. Algunos añaden algo de ruido en la generación de X en lugar de utilizar un decodificador determinista. La mayoría no utiliza el criterio de máxima verosimilitud, sino que pretende hacer que los elementos de $h = W^{-1}X$ independientes unos de otros. Muchos criterios que logran este objetivo son posibles. Ecuación 3.47 requiere tomar el determinante de W , que puede ser una operación costosa y numéricamente inestable. Algunas variantes de ICA evitan esta operación problemática al restringir W ser ortogonal.

Todas las variantes de ICA requieren que $pag(h)$ sea no gaussiano. Esto es porque si $pag(h)$ es un previo independiente con componentes gaussianas, entonces W no es identificable. Podemos obtener la misma distribución sobre $pag(X)$ para muchos valores de W . Esto es muy diferente de otros modelos de factores lineales como PCA probabilístico y análisis factorial, que a menudo requieren $pag(h)$ sea gaussiano para que muchas operaciones sobre el modelo tengan soluciones de forma cerrada. En el enfoque de máxima verosimilitud donde el usuario especifica explícitamente la distribución, una elección típica es utilizar $pag(h_i) = \text{d}_{\text{exp}}(h_i)$. Las elecciones típicas de estas distribuciones no gaussianas tienen picos más grandes cerca de 0 que la distribución gaussiana, por lo que también podemos ver la mayoría de las implementaciones de ICA como características escasas de aprendizaje.

Muchas variantes de ICA no son modelos generativos en el sentido en que usamos la frase. En este libro, un modelo generativo representa $p_{\theta}(X)$ o puede extraer muestras de él. Muchas variantes de ICA solo saben cómo transformar entre $X \rightarrow h$, pero no tienen ninguna forma de representar $p_{\theta}(h)$, y por lo tanto no imponen una distribución sobre $p_{\theta}(h)$. Por ejemplo, muchas variantes de ICA tienen como objetivo aumentar la curtosis de la muestra de $h = W^{-1}X$, porque una curtosis alta indica que $p_{\theta}(h)$ no es gaussiano, pero esto se logra sin representar explícitamente $p_{\theta}(h)$. Esto se debe a que ICA se usa más a menudo como una herramienta de análisis para separar señales que para generar datos o estimar su densidad.

Así como PCA se puede generalizar a los codificadores automáticos no lineales descritos en el capítulo 14, ICA se puede generalizar a un modelo generativo no lineal, en el que usamos una función no lineal f para generar los datos observados. Ver [Hyvärinen y Pajunen \(1999\)](#) por el trabajo inicial en ICA no lineal y su uso exitoso con aprendizaje conjunto por [Roberts y Everson \(2001\)](#) y [Lappalainen et al. \(2000\)](#). Otra extensión no lineal de ICA es el enfoque de **estimación de componentes independientes no lineales**, o AGRADABLE ([comedore et al., 2014](#)), que apila una serie de transformaciones invertibles (etapas de codificador) que tienen la propiedad de que el determinante del jacobiano de cada transformación se puede calcular de manera eficiente. Esto hace posible calcular la probabilidad exactamente y, como ICA, intenta transformar los datos en un espacio donde tiene una distribución marginal factorizada, pero es más probable que tenga éxito gracias al codificador no lineal. Debido a que el codificador está asociado con un decodificador que es su inverso perfecto, es sencillo generar muestras del modelo (primero tomando muestras de $p_{\theta}(h)$ y luego aplicar el decodificador).

Otra generalización de ICA es aprender grupos de características, con dependencia estadística permitida dentro de un grupo pero desaconsejada entre grupos ([Hyvärinen y Hoyer, 1999; Hyvärinen et al., 2001b](#)). Cuando los grupos de unidades relacionadas se eligen para que no se superpongan, esto se denomina **análisis subespacial independiente**. También es posible asignar coordenadas espaciales a cada unidad oculta y formar grupos superpuestos de unidades espacialmente vecinas. Esto anima a las unidades cercanas a aprender características similares. Cuando se aplica a imágenes naturales, este **ICA topográfica** El enfoque aprende los filtros Gabor, de modo que las características vecinas tienen una orientación, ubicación o frecuencia similares. Dentro de cada región se producen muchos desplazamientos de fase diferentes de funciones de Gabor similares, de modo que la combinación de regiones pequeñas produce invariancia de traducción.

13.3 Análisis de características lentes

Análisis de características lento(SFA) es un modelo de factor lineal que utiliza información de

señales de tiempo para aprender características invariantes (Wiskott y Sejnowski, 2002).

El análisis de características lentas está motivado por un principio general llamado principio de lentitud. La idea es que las características importantes de las escenas cambien muy lentamente en comparación con las medidas individuales que componen la descripción de una escena. Por ejemplo, en la visión artificial, los valores de píxeles individuales pueden cambiar muy rápidamente. Si una cebra se mueve de izquierda a derecha en la imagen, un píxel individual cambiará rápidamente de negro a blanco y viceversa cuando las rayas de la cebra pasen sobre el píxel. En comparación, la característica que indica si una cebra está en la imagen no cambiará en absoluto y la característica que describe la posición de la cebra cambiará lentamente. Por lo tanto, es posible que deseemos regularizar nuestro modelo para aprender características que cambian lentamente con el tiempo.

El principio de lentitud es anterior al análisis de características lentas y se ha aplicado a una amplia variedad de modelos (Hinton, 1989; Földiak, 1989; Mabahiet et al., 2009; Bergstra y Bengio, 2009). En general, podemos aplicar el principio de lentitud a cualquier modelo diferenciable entrenado con gradiente descendente. El principio de lentitud se puede introducir agregando un término a la función de costo de la forma

$$\lambda \sum_t L(F(X_{(t+1)}), F(X_{(t)})) \quad (13.7)$$

dónde λ es un hiperparámetro que determina la fuerza del término de regularización de la lentitud, t es el índice en una secuencia de tiempo de ejemplos, F es el extractor de características a regularizar, y L es una función de pérdida que mide la distancia entre $F(X_{(t)})$ y $F(X_{(t+1)})$. Una opción común para L es la diferencia cuadrática media.

El análisis de características lentas es una aplicación particularmente eficiente del principio de lentitud. Es eficiente porque se aplica a un extractor de características lineales y, por lo tanto, se puede entrenar en forma cerrada. Al igual que algunas variantes de ICA, SFA no es un modelo generativo per se, en el sentido de que define un mapa lineal entre el espacio de entrada y el espacio de características, pero no define un espacio de características previo y, por lo tanto, no impone una distribución $p_{\text{prior}}(\mathbf{x})$ en el espacio de entrada.

El algoritmo SFA (Wiskott y Sejnowski, 2002) consiste en definir $F(X; \theta)$ ser una transformación lineal, y resolviendo el problema de optimización

$$\min_{\theta} \sum_t \|F(X_{(t+1)}) - F(X_{(t)})\|^2 \quad (13.8)$$

sujeto a las restricciones

$$\sum_t F(X_{(t)}) = 0 \quad (13.9)$$

y

$$\sum_t [F(X_{(t)})]^2 = 1. \quad (13.10)$$

La restricción de que la característica aprendida tenga media cero es necesaria para que el problema tenga una solución única; de lo contrario, podríamos agregar una constante a todos los valores de las características y obtener una solución diferente con el mismo valor del objetivo de lentitud. La restricción de que las características tienen varianza unitaria es necesaria para evitar la solución patológica donde todas las características colapsan para 0. Al igual que PCA, las funciones de SFA están ordenadas, siendo la primera función la más lenta. Para aprender múltiples funciones, también debemos agregar la restricción

$$\forall y \in \mathcal{Y}, \sum_j \text{mit}[f(X(t))_j f(X(-t))_j] = 0. \quad (13.11)$$

Esto especifica que las características aprendidas deben estar linealmente decorrelacionadas entre sí. Sin esta restricción, todas las características aprendidas simplemente capturarían la señal más lenta. Uno podría imaginar usar otros mecanismos, como minimizar el error de reconstrucción, para forzar la diversificación de las características, pero este mecanismo de decorrelación admite una solución simple debido a la linealidad de las características SFA. El problema SFA puede resolverse en forma cerrada mediante un paquete de álgebra lineal.

SFA se usa típicamente para aprender características no lineales aplicando una expansión de base no lineal a X antes de ejecutar SFA. Por ejemplo, es común reemplazar X por la expansión de la base cuadrática, un vector que contiene elementos $X_i X_j$ para todos i, j . Luego, los módulos SFA lineales se pueden componer para aprender extractores de características lentas no lineales profundos al aprender repetidamente un extractor de características SFA lineal, aplicar una expansión de base no lineal a su salida y luego aprender otro extractor de características SFA lineal además de esa expansión.

Cuando se entrena en pequeños parches espaciales de videos de escenas naturales, SFA con expansiones de base cuadrática aprende características que comparten muchas características con las de células complejas en la corteza V1 ([Berkes y Wiskott, 2005](#)). Cuando se entrena en videos de movimiento aleatorio dentro de entornos renderizados por computadora en 3D, Deep SFA aprende características que comparten muchas características con las características representadas por las neuronas en los cerebros de ratas que se utilizan para la navegación ([Francisco et al., 2007](#)). Por lo tanto, SFA parece ser un modelo razonablemente plausible desde el punto de vista biológico.

Una gran ventaja de SFA es que es posible predecir teóricamente qué características aprenderá SFA, incluso en un entorno no lineal profundo. Para hacer tales predicciones teóricas, uno debe conocer la dinámica del entorno en términos de espacio de configuración (p. ej., en el caso del movimiento aleatorio en el entorno renderizado en 3D, el análisis teórico procede del conocimiento de la distribución de probabilidad sobre la posición y velocidad de la cámara). Dado el conocimiento de cómo cambian realmente los factores subyacentes, es posible resolver analíticamente las funciones óptimas que expresan estos factores. En la práctica, los experimentos con SFA profundo aplicados a datos simulados parecen recuperar las funciones predichas teóricamente.

Esto es en comparación con otros algoritmos de aprendizaje donde la función de costo depende en gran medida de valores de píxeles específicos, lo que hace que sea mucho más difícil determinar qué funciones aprenderá el modelo.

Deep SFA también se ha utilizado para aprender funciones para el reconocimiento de objetos y la estimación de poses (Francisco et al., 2008). Hasta ahora, el principio de lentitud no se ha convertido en la base de ninguna aplicación de vanguardia. No está claro qué factor ha limitado su rendimiento. Especulamos que tal vez la lentitud previa sea demasiado fuerte y que, en lugar de imponer una previa de que las características deberían ser aproximadamente constantes, sería mejor imponer una previa de que las características deberían ser fáciles de predecir de un paso de tiempo al siguiente. La posición de un objeto es una función útil independientemente de si la velocidad del objeto es alta o baja, pero el principio de lentitud anima al modelo a ignorar la posición de los objetos que tienen una velocidad alta.

13.4 Codificación dispersa

Codificación escasa(Olshausen y campo, 1996) es un modelo de factor lineal que ha sido muy estudiado como un mecanismo de aprendizaje y extracción de características no supervisado. Estrictamente hablando, el término "codificación dispersa" se refiere al proceso de inferir el valor de h en este modelo, mientras que "modelado disperso" se refiere al proceso de diseño y aprendizaje del modelo, el término "codificación dispersa" se usa a menudo para referirse a ambos.

Como la mayoría de los otros modelos de factores lineales, utiliza un decodificador lineal más ruido para obtener reconstrucciones de X , como se especifica en la ecuación 13.2. Más específicamente, los modelos de codificación dispersa generalmente asumen que los factores lineales tienen ruido gaussiano con precisión isotrópica. β :

$$pag(x / h) = norte(X; \{Qué? + b, y\}) \cdot \frac{1}{\beta}. \quad (13.12)$$

La distribución $pag(h)$ se elige para ser uno con picos agudos cerca de 0 (Olshausen y campo, 1996). Las opciones comunes incluyen Laplace factorizado, Cauchy o Student-factorizado. t distribuciones. Por ejemplo, el previo de Laplace parametrizado en términos del coeficiente de penalización por escasez λ dado por

$$pag(h_i) = \text{Laplace}(h_i; 0, \frac{2}{\lambda}) = \frac{\lambda}{4} m_i \frac{e^{-\lambda/|h_i|}}{2}, \quad (13.13)$$

y el estudiante- t antes de

$$pag(h_i) \propto \frac{1}{(1 + \frac{|h_i|}{\sqrt{v}})^{\frac{v+1}{2}}}. \quad (13.14)$$

El entrenamiento de codificación dispersa con la máxima probabilidad es intratable. En cambio, el entrenamiento alterna entre codificar los datos y entrenar al decodificador para reconstruir mejor los datos dada la codificación. Este enfoque se justificará más adelante como una aproximación basada en principios a la máxima verosimilitud, en la sección 19.3.

Para modelos como PCA, hemos visto el uso de una función de codificador paramétrico que predice h y consiste únicamente en la multiplicación por una matriz de pesos. El codificador que usamos con codificación escasa no es un codificador paramétrico. En cambio, el codificador es un algoritmo de optimización que resuelve un problema de optimización en el que buscamos el valor de código único más probable:

$$h = \underset{h}{\text{argmax}} \, p_{\text{ag}}(h / X). \quad (13.15)$$

Cuando se combina con la ecuación 13.13 y ecuación 13.12, esto produce el siguiente problema de optimización:

$$\underset{h}{\text{argmax}} \, p_{\text{ag}}(h / X) \quad (13.16)$$

$$= \underset{h}{\text{registro máximo de }} \underset{h}{\text{argmax}} \, p_{\text{ag}}(h / X) \quad (13.17)$$

$$= \underset{h}{\text{argmin}} \lambda \|h\|_1 + \beta \|X - Wh\|_2^2, \quad (13.18)$$

donde hemos descartado términos que no dependen de h dividido por factores de escala positivos para simplificar la ecuación.

Debido a la imposición de una norma sobre h , este procedimiento producirá una escasa h . Mira la sección 7.1.2).

Para entrenar el modelo en lugar de solo realizar inferencias, alternamos entre minimización con respecto a h y minimización con respecto a W . En esta presentación tratamos β como un hiperparámetro. Por lo general, se establece en 1 porque su función en este problema de optimización se comparte con λ y no hay necesidad de ambos hiperparámetros. En principio, también podríamos tratar β como un parámetro del modelo y aprenderlo. Nuestra presentación aquí ha descartado algunos términos que no dependen de h pero depende de β . Aprender β , estos términos deben ser incluidos, o β se derrumbará a 0.

No todos los enfoques de la codificación dispersa construyen explícitamente un $p_{\text{ag}}(h)$ y un $p_{\text{ag}}(x / h)$. A menudo, solo estamos interesados en aprender un diccionario de características con valores de activación que a menudo serán cero cuando se extraigan mediante este procedimiento de inferencia.

Si tomamos muestras h de un previo de Laplace, de hecho es un evento de probabilidad cero para un elemento de h ser realmente cero. El modelo generativo en sí no es especialmente escaso, solo lo es el extractor de características. [Buen compañero et al. \(2013d\)](#) describirá un aproximado

inferencia en una familia de modelos diferente, el modelo de codificación dispersa Spike and Slab, para el cual las muestras del anterior generalmente contienen ceros verdaderos.

El enfoque de codificación escasa combinado con el uso del codificador no paramétrico puede, en principio, minimizar la combinación de error de reconstrucción y registro previo mejor que cualquier codificador paramétrico específico. Otra ventaja es que no hay error de generalización en el codificador. Un codificador paramétrico debe aprender a mapear \mathbf{x} a \mathbf{h} de una manera que generaliza por inusual \mathbf{x} que no se parecen a los datos de entrenamiento, un codificador paramétrico aprendido puede no encontrar un \mathbf{h} que da como resultado una reconstrucción precisa o un código escaso. Para la gran mayoría de las formulaciones de modelos de codificación dispersa, donde el problema de inferencia es convexo, el procedimiento de optimización siempre encontrará el código óptimo (a menos que ocurran casos degenerados como vectores de peso replicados). Obviamente, los costos de escasez y reconstrucción aún pueden aumentar en puntos desconocidos, pero esto se debe a un error de generalización en los pesos del decodificador, más que a un error de generalización en el codificador. La falta de error de generalización en el proceso de codificación basado en la optimización de la codificación dispersa puede resultar en una mejor generalización cuando la codificación dispersa se usa como un extracto de características para un clasificador que cuando se usa una función paramétrica para predecir el código. [Coates y Ng\(2011\)](#) demostraron que las funciones de codificación dispersas se generalizan mejor para las tareas de reconocimiento de objetos que las funciones de un modelo relacionado basado en un codificador paramétrico, el autocodificador sigmoide lineal. Inspirándose en su trabajo, [Buen compañero et al. \(2013d\)](#) mostró que una variante de codificación dispersa se generaliza mejor que otros extractores de características en el régimen donde hay muy pocas etiquetas disponibles (veinte o menos etiquetas por clase).

La principal desventaja del codificador no paramétrico es que requiere más tiempo para calcular \mathbf{h} dado \mathbf{x} porque el enfoque no paramétrico requiere ejecutar un algoritmo iterativo. El enfoque del codificador automático paramétrico, desarrollado en el capítulo [14](#), utiliza solo un número fijo de capas, a menudo solo una. Otra desventaja es que no es sencillo propagar hacia atrás a través del codificador no paramétrico, lo que dificulta el entrenamiento previo de un modelo de codificación dispersa con un criterio no supervisado y luego ajustarlo con un criterio supervisado. Existen versiones modificadas de codificación dispersa que permiten derivados aproximados, pero no se usan ampliamente ([Bagnell y Bradley, 2009](#)).

La codificación dispersa, al igual que otros modelos de factores lineales, a menudo produce muestras deficientes, como se muestra en la figura [13.2](#). Esto sucede incluso cuando el modelo puede reconstruir bien los datos y proporcionar características útiles para un clasificador. La razón es que cada característica individual puede aprenderse bien, pero el factorial previo en el código oculto da como resultado que el modelo incluya subconjuntos aleatorios de todas las características en cada muestra generada. Esto motiva el desarrollo de modelos más profundos que puedan imponer un

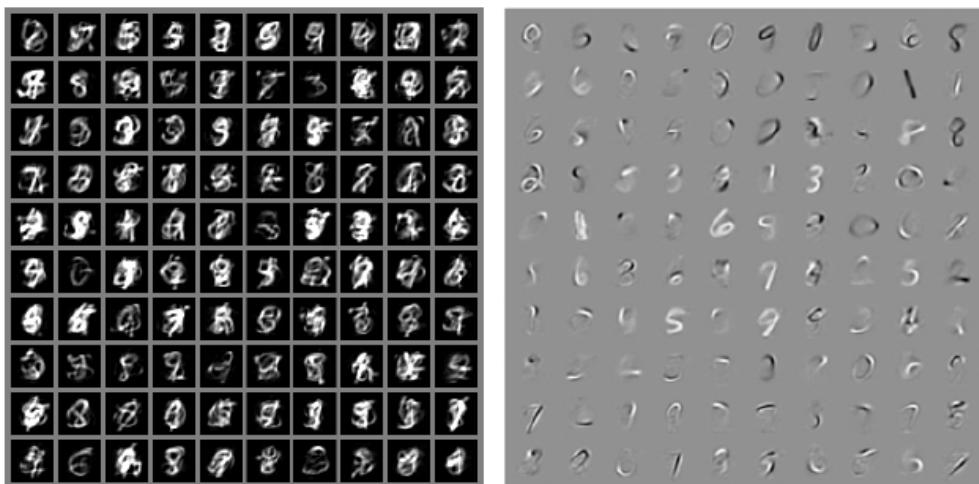


Figura 13.2: Ejemplos de muestras y pesos de un modelo de codificación dispersa de espiga y losa entrenado en el conjunto de datos MNIST.(Izquierda) Las muestras del modelo no se parecen a los ejemplos de entrenamiento. A primera vista, se podría suponer que el modelo no se ajusta bien.(Derecha) Los vectores de peso del modelo han aprendido a representar trazos de pluma y, a veces, dígitos completos. El modelo ha aprendido características útiles. El problema es que el factorial anterior sobre las características da como resultado la combinación aleatoria de subconjuntos de características. Pocos de estos subconjuntos son apropiados para formar un dígito MNIST reconocible. Esto motiva el desarrollo de modelos generativos que tienen distribuciones más poderosas sobre sus códigos latentes. Figura reproducida con permiso de [Bueno et al. \(2013d\)](#).

distribución factorial en la capa de código más profunda, así como el desarrollo de modelos superficiales más sofisticados.

13.5 Interpretación múltiple de PCA

Los modelos de factores lineales que incluyen PCA y análisis factorial pueden interpretarse como el aprendizaje de una variedad ([Hinton et al., 1997](#)). Podemos considerar que el PCA probabilístico define una región delgada con forma de panqueque de alta probabilidad, una distribución gaussiana que es muy estrecha a lo largo de algunos ejes, al igual que un panqueque es muy plano a lo largo de su eje vertical, pero se alarga a lo largo de otros ejes, al igual que un el panqueque es ancho a lo largo de sus ejes horizontales. Esto se ilustra en la figura 13.3. PCA puede interpretarse como la alineación de este panqueque con una variedad lineal en un espacio de dimensiones superiores. Esta interpretación se aplica no solo al PCA tradicional, sino también a cualquier codificador automático lineal que aprenda matrices. *My goal* con el objetivo de hacer la reconstrucción de x acostarse tan cerca de x como sea posible,

Deje que el codificador sea

$$h = f(X) = W(x - \mu). \quad (13.19)$$

El codificador calcula una representación de baja dimensión de h . Con la vista de codificador automático, tenemos un decodificador que calcula la reconstrucción

$$X = \text{gramo}(h) = b + Vh. \quad (13.20)$$

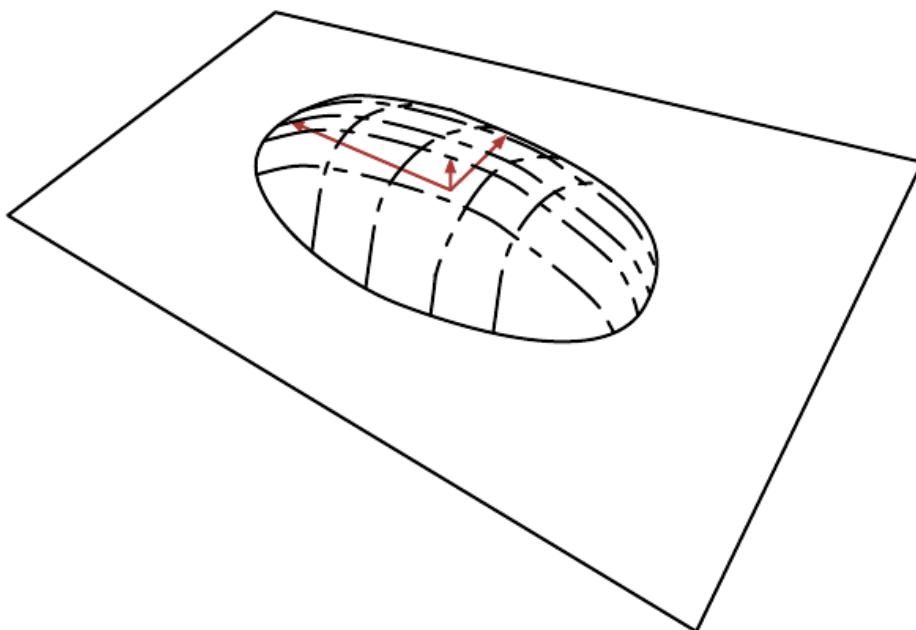


Figura 13.3: Gaussiana plana que captura la concentración de probabilidad cerca de una variedad de baja dimensión. La figura muestra la mitad superior del "panqueque" por encima del "plano múltiple" que pasa por su centro. La variación en la dirección ortogonal a la variedad es muy pequeña (flecha que apunta fuera del plano) y puede considerarse como "ruido", mientras que las otras variaciones son grandes (flechas en el plano) y corresponden a "señal" y una coordenada. sistema para los datos de dimensiones reducidas.

Las opciones de codificador y decodificador lineal que minimizan el error de reconstrucción

$$\text{MI}[\|x - \hat{x}\|/2] \quad (13.21)$$

corresponden a las $V=W$, $\mu=b=\text{MI}[X]$ y las columnas de W forman una base ortonormal que abarca el mismo subespacio que los vectores propios principales de la matriz de covarianza

$$C = \text{MI}[(x - \mu)(x - \mu)^T]. \quad (13.22)$$

En el caso de PCA, las columnas de W son estos vectores propios, ordenados por la magnitud de los valores propios correspondientes (que son todos reales y no negativos).

También se puede demostrar que el valor propio λ_i de C corresponde a la varianza de X en la dirección del vector propio v_i . Si $X \in \mathbb{R}^d$ y $h \in \mathbb{R}^d$ con $r \leq d < r_e$, entonces el

error de reconstrucción óptimo (elegir μ, b, V y W como arriba) es

$$\min_{\mu, b} \text{MI}[\|x - \hat{x}\|^2] = \sum_{i=d+1}^D \lambda_i. \quad (13.23)$$

Por lo tanto, si la covarianza tiene rango d , los valores propios λ_{d+1} a λ_D son 0 y el error de reconstrucción es 0.

Además, también se puede demostrar que la solución anterior se puede obtener maximizando las varianzas de los elementos de h , bajo ortogonal W , en lugar de minimizar el error de reconstrucción.

Los modelos de factores lineales son algunos de los modelos generativos más simples y algunos de los modelos más simples que aprenden una representación de datos. Así como los clasificadores lineales y los modelos de regresión lineal pueden extenderse a redes de retroalimentación profunda, estos modelos de factores lineales pueden extenderse a redes de codificador automático y modelos probabilísticos profundos que realizan las mismas tareas pero con una familia de modelos mucho más poderosa y flexible.

capítulo 14

Codificadores automáticos

Uncodificador automático es una red neuronal que está entrenada para intentar copiar su entrada a su salida. Internamente, tiene una capa oculta. h que describe un **código** utilizado para representar la entrada. Se puede considerar que la red consta de dos partes: una función de codificador $h = f(X)$ y un decodificador que produce una reconstrucción $r = g_{\text{ramo}}(h)$. Esta arquitectura se presenta en la figura 14.1. Si un codificador automático tiene éxito simplemente aprendiendo a configurar $g_{\text{ramo}}(f(X)) = X$ en todas partes, entonces no es especialmente útil. En cambio, los codificadores automáticos están diseñados para no poder aprender a copiar perfectamente. Por lo general, están restringidos de manera que les permiten copiar solo aproximadamente y copiar solo la entrada que se parece a los datos de entrenamiento. Debido a que el modelo se ve obligado a priorizar qué aspectos de la entrada deben copiarse, a menudo aprende propiedades útiles de los datos.

Los autocodificadores modernos han generalizado la idea de un codificador y un decodificador más allá de las funciones deterministas a las asignaciones estocásticas. $p_{\text{codificador}}(h / X)$ y $p_{\text{descifrador}}(x / h)$.

La idea de los autocodificadores ha formado parte del panorama histórico de las redes neuronales durante décadas (lecun, 1987; Bourlard y Camp, 1988; Hinton y Zemel, 1994). Tradicionalmente, los codificadores automáticos se usaban para la reducción de la dimensionalidad o el aprendizaje de funciones. Recientemente, las conexiones teóricas entre los codificadores automáticos y los modelos de variables latentes han llevado a los codificadores automáticos al frente del modelado generativo, como veremos en el capítulo 20. Se puede considerar que los codificadores automáticos son un caso especial de las redes de realimentación y se pueden entrenar con todas las mismas técnicas, por lo general, el descenso de gradientes de minibatches siguiendo los gradientes calculados por propagación inversa. A diferencia de las redes feedforward generales, los codificadores automáticos también pueden entrenarse usando **recirculación** (Hinton y McClelland, 1988), un algoritmo de aprendizaje basado en la comparación de las activaciones de la red sobre la entrada original

a las activaciones en la entrada reconstruida. La recirculación se considera biológicamente más plausible que la retropropagación, pero rara vez se usa para aplicaciones de aprendizaje automático.

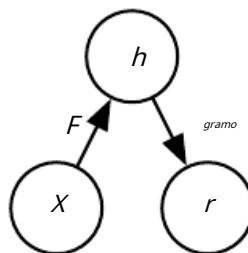


Figura 14.1: La estructura general de un autoencoder, mapeando una entrada X a una salida (llamada *reconstrucción*) a través de una representación o código interno h . El autoencoder tiene dos componentes: el codificador F (cartografía X a h) y el decodificador $gramo$ (cartografía h a r).

14.1 Codificadores automáticos incompletos

Copiar la entrada a la salida puede sonar inútil, pero normalmente no estamos interesados en la salida del decodificador. En cambio, esperamos que entrenar al codificador automático para realizar la tarea de copia de entrada resulte en h adquiriendo propiedades útiles.

Una forma de obtener características útiles del codificador automático es restringir/tener una dimensión menor que X . Un codificador automático cuya dimensión de código es menor que la dimensión de entrada se llama **incompleto**. El aprendizaje de una representación incompleta obliga al codificador automático a capturar las características más destacadas de los datos de entrenamiento.

El proceso de aprendizaje se describe simplemente como la minimización de una función de pérdida

$$L(X, gramof(F(X))) \quad (14.1)$$

dónde L es una función de pérdida que penaliza $gramof(F(X))$ por ser diferente a X , como el error cuadrático medio.

Cuando el decodificador es lineal y L es el error cuadrático medio, un codificador automático incompleto aprende a abarcar el mismo subespacio que PCA. En este caso, un codificador automático entrenado para realizar la tarea de copia ha aprendido el subespacio principal de los datos de entrenamiento como efecto secundario.

Codificadores automáticos con funciones de codificador no lineal F y funciones de decodificador no lineal $gramo$ puede así aprender una generalización no lineal más poderosa de PCA. Desafortunadamente

Por supuesto, si el codificador y el decodificador tienen demasiada capacidad, el codificador automático puede aprender a realizar la tarea de copia sin extraer información útil sobre la distribución de los datos. En teoría, uno podría imaginar que un codificador automático con un código unidimensional pero un codificador no lineal muy potente podría aprender a representar cada ejemplo de entrenamiento X_i con el código i . El decodificador podría aprender a mapear estos índices enteros a los valores de ejemplos de entrenamiento específicos. Este escenario específico no ocurre en la práctica, pero ilustra claramente que un codificador automático capacitado para realizar la tarea de copia puede no aprender nada útil sobre el conjunto de datos si se permite que la capacidad del codificador automático sea demasiado grande.

14.2 Autocodificadores Regularizados

Los codificadores automáticos incompletos, con una dimensión de código menor que la dimensión de entrada, pueden aprender las características más destacadas de la distribución de datos. Hemos visto que estos codificadores automáticos no aprenden nada útil si el codificador y el decodificador tienen demasiada capacidad.

Un problema similar ocurre si se permite que el código oculto tenga una dimensión igual a la entrada, y en el **sobrecompleto** caso en el que el código oculto tiene una dimensión mayor que la entrada. En estos casos, incluso un codificador lineal y un decodificador lineal pueden aprender a copiar la entrada a la salida sin aprender nada útil sobre la distribución de datos.

Idealmente, se podría entrenar con éxito cualquier arquitectura de autocodificador, eligiendo la dimensión del código y la capacidad del codificador y del decodificador en función de la complejidad de la distribución a modelar. Los codificadores automáticos regularizados brindan la capacidad de hacerlo. En lugar de limitar la capacidad del modelo manteniendo el codificador y el decodificador poco profundos y el tamaño del código pequeño, los codificadores automáticos regularizados utilizan una función de pérdida que anima al modelo a tener otras propiedades además de la capacidad de copiar su entrada a su salida. Estas otras propiedades incluyen la escasez de la representación, la pequeñez de la derivada de la representación y la robustez al ruido o a las entradas faltantes.

Además de los métodos descritos aquí, que se interpretan de manera más natural como codificadores automáticos regularizados, casi cualquier modelo generativo con variables latentes y equipado con un procedimiento de inferencia (para calcular las representaciones latentes dadas las entradas) puede verse como una forma particular de codificador automático. Dos enfoques de modelado generativo que enfatizan esta conexión con los codificadores automáticos son los descendientes de la máquina de Helmholtz ([Hinton et al., 1995b](#)), como la variacional

codificador automático (sección 20.10.3) y las redes estocásticas generativas (sección 20.12). Estos modelos aprenden de forma natural codificaciones demasiado completas y de alta capacidad de la entrada y no requieren regularización para que estas codificaciones sean útiles. Sus codificaciones son naturalmente útiles porque los modelos fueron entrenados para maximizar aproximadamente la probabilidad de los datos de entrenamiento en lugar de copiar la entrada a la salida.

14.2.1 Codificadores automáticos dispersos

Un codificador automático disperso es simplemente un codificador automático cuyo criterio de entrenamiento implica una penalización por escasez $\Omega(h)$ en la capa de código h , además del error de reconstrucción:

$$L(X, \text{gramo}(F(X))) + \Omega(h) \quad (14.2)$$

dónde $\text{gramo}(h)$ es la salida del decodificador y típicamente tenemos $h = F(X)$, la salida del codificador.

Los codificadores automáticos dispersos se utilizan normalmente para aprender funciones para otra tarea, como la clasificación. Un codificador automático que se ha regularizado para que sea escaso debe responder a las características estadísticas únicas del conjunto de datos en el que se ha entrenado, en lugar de simplemente actuar como una función de identidad. De esta manera, el entrenamiento para realizar la tarea de copiar con una penalización por escasez puede generar un modelo que ha aprendido características útiles como subproducto.

Podemos pensar en la pena $\Omega(h)$ simplemente como un término regularizador agregado a una red de retroalimentación cuya tarea principal es copiar la entrada a la salida (objetivo de aprendizaje no supervisado) y posiblemente también realizar alguna tarea supervisada (con un objetivo de aprendizaje supervisado) que depende de estas características dispersas. A diferencia de otros regularizadores como el decaimiento de peso, no existe una interpretación bayesiana directa para este regularizador. Como se describe en la sección 5.6.1, el entrenamiento con caída de peso y otras penalizaciones de regularización pueden interpretarse como una aproximación MAP a la inferencia bayesiana, con la penalización de regularización añadida correspondiente a una distribución de probabilidad previa sobre los parámetros del modelo. Desde este punto de vista, la máxima verosimilitud regularizada corresponde a la maximización $\text{pag}(\theta / X)$, lo que equivale a maximizar $\text{registro pag}(x / \theta) + \text{registro pag}(\theta)$. El término $\text{registro pag}(x / \theta)$ es el término habitual de probabilidad de registro de datos y el término $\text{registro pag}(\theta)$ el término, el logaritmo anterior sobre los parámetros, incorpora la preferencia sobre valores particulares de θ . Esta vista se describió en la sección 5.6. Los autocodificadores regularizados desafían tal interpretación porque el regularizador depende de los datos y, por lo tanto, por definición, no es un previo en el sentido formal de la palabra. Todavía podemos pensar que estos términos de regularización expresan implícitamente una preferencia sobre funciones.

En lugar de pensar en la penalización por escasez como un regularizador de la tarea de copia, podemos pensar en todo el marco del codificador automático disperso como una aproximación

entrenamiento de máxima verosimilitud de un modelo generativo que tiene variables latentes. Supongamos que tenemos un modelo con variables visibles X y variables latentes h , con una distribución conjunta explícita $pag_{\text{modelo}}(x, h) = pag_{\text{modelo}}(h) pag_{\text{modelo}}(x | h)$. Nos referimos a $pag_{\text{modelo}}(h)$ como la distribución previa del modelo sobre las variables latentes, que representan las creencias del modelo antes de ver X . Esto es diferente de la forma en que hemos usado previamente la palabra "previo", para referirnos a la distribución $pag(\theta)$ codificar nuestras creencias sobre los parámetros del modelo antes de que hayamos visto los datos de entrenamiento. La log-verosimilitud se puede descomponer como

$$\underset{h}{\text{registro}} pag_{\text{modelo}}(X) = \underset{h}{\text{registro}} pag_{\text{modelo}}(h) + \underset{h}{\text{registro}} pag_{\text{modelo}}(x | h). \quad (14.3)$$

Podemos pensar en el codificador automático como una aproximación de esta suma con una estimación puntual de solo un valor altamente probable para h . Esto es similar al modelo generativo de codificación dispersa (sección 13.4), pero con h siendo la salida del codificador paramétrico en lugar del resultado de una optimización que infiere el más probable h . Desde este punto de vista, con este elegido h , estamos maximizando

$$\text{registro} pag_{\text{modelo}}(h, x) = \text{registro} pag_{\text{modelo}}(h) + \text{registro} pag_{\text{modelo}}(x | h). \quad (14.4)$$

El registro $pag_{\text{modelo}}(h)$ El término puede inducir a la escasez. Por ejemplo, el prior de Laplace,

$$pag_{\text{modelo}}(h_i) = \frac{\lambda}{2} e^{-\lambda|h_i|}, \quad (14.5)$$

corresponde a una penalización por escasez de valor absoluto. Expresando el log-prior como una penalización de valor absoluto, obtenemos

$$\Omega(h) = \lambda \sum_i |h_i| \quad (14.6)$$

$$-\text{registro} pag_{\text{modelo}}(h) = -\sum_i \lambda |h_i| - \text{registro} \frac{\lambda}{2} = \Omega(h) + \text{constante} \quad (14.7)$$

donde el término constante depende sólo de λ no h . Normalmente tratamos λ como un hiperparámetro y descartar el término constante ya que no afecta el aprendizaje del parámetro. Otros antecedentes como el Student-t anterior también puede inducir la escasez. Desde este punto de vista de la escasez como resultado del efecto de $pag_{\text{modelo}}(h)$ en el aprendizaje de máxima verosimilitud aproximada, la penalización por escasez no es un término de regularización en absoluto. Es sólo una consecuencia de la distribución del modelo sobre sus variables latentes. Esta vista proporciona una motivación diferente para entrenar un codificador automático: es una forma de entrenar aproximadamente un modelo generativo. También proporciona una razón diferente para

por qué las características aprendidas por el codificador automático son útiles: describen las variables latentes que explican la entrada.

Primeros trabajos sobre codificadores automáticos dispersos ([Ranzato et al., 2007a, 2008](#)) exploró varias formas de escasez y propuso una conexión entre la pena de escasez y el registro, término que surge al aplicar la máxima verosimilitud a un modelo probabilístico $\text{pag}(X) = \text{exp}(\text{pag}(X))$. La idea es que minimizando el registro previene un modelo probabilístico tenga una alta probabilidad en todas partes, y la imposición de escasez en un codificador automático evita que el codificador automático tenga un error de reconstrucción bajo en todas partes. En este caso, la conexión está en el nivel de una comprensión intuitiva de un mecanismo general en lugar de una correspondencia matemática. La interpretación de la penalización por escasez como correspondiente a registrar $\text{pag}_{\text{modelo}}(h)$ en un modelo dirigido $\text{pag}_{\text{modelo}}(h)$ es más matemáticamente sencillo.

Una forma de lograr ceros reales en h para codificadores automáticos dispersos (y sin ruido) se introdujo en [glorria et al. \(2011b\)](#). La idea es utilizar unidades lineales rectificadas para producir la capa de código. Con un prior que realmente lleva las representaciones a cero (como la penalización del valor absoluto), uno puede controlar indirectamente el número promedio de ceros en la representación.

14.2.2 Eliminación de ruido de codificadores automáticos

En lugar de agregar una penalización a la función de costo, podemos obtener un codificador automático que aprende algo útil al cambiar el término de error de reconstrucción de la función de costo.

Tradicionalmente, los codificadores automáticos minimizan alguna función

$$L(X, \text{gramo}(F(X))) \quad (14.8)$$

dónde L es una función de pérdida que penaliza $\text{gramo}(F(X))$ por ser diferente a X , tales como el L_2 norma de su diferencia. Esto alienta $\text{gramo} \circ F$ aprender a ser simplemente una función de identidad si tienen la capacidad de hacerlo.

Acodificador automático de eliminación de ruido DAE en su lugar minimiza

$$L(X, \text{gramo}(F(\tilde{X}))), \quad (14.9)$$

dónde \tilde{X} es una copia de X que ha sido corrompido por alguna forma de ruido. Por lo tanto, los codificadores automáticos de eliminación de ruido deben deshacer esta corrupción en lugar de simplemente copiar su entrada.

Fuerzas de entrenamiento de eliminación de ruido $F \circ \text{gramo}$ aprender implícitamente la estructura de $\text{pag}_{\text{datos}}(X)$, como se muestra [Alain y Bengio \(2013\)](#) y [bengio et al. \(2013c\)](#). eliminación de ruido

los codificadores automáticos proporcionan otro ejemplo más de cómo las propiedades útiles pueden surgir como un subproducto de minimizar el error de reconstrucción. También son un ejemplo de cómo los modelos demasiado completos y de alta capacidad pueden usarse como codificadores automáticos siempre que se tenga cuidado de evitar que aprendan la función de identidad. Los codificadores automáticos de eliminación de ruido se presentan con más detalle en la sección 14.5.

14.2.3 Regularización por penalización de derivados

Otra estrategia para regularizar un autoencoder es usar una penalización Ω como en autocodificadores dispersos,

$$L(X, \text{gramo}(R(X))) + \Omega(h, x), \quad (14.10)$$

pero con otra forma de Ω :

$$\Omega(h, x) = \lambda \sum_i \| \nabla_x h_i \|_2. \quad (14.11)$$

Esto obliga al modelo a aprender una función que no cambia mucho cuando X cambia ligeramente. Debido a que esta penalización se aplica solo en ejemplos de entrenamiento, obliga al codificador automático a aprender funciones que capturan información sobre la distribución de entrenamiento.

Un autocodificador regularizado de esta manera se llama **codificador automático contractivo** o CAE. Este enfoque tiene conexiones teóricas con la eliminación de ruido de los codificadores automáticos, el aprendizaje múltiple y el modelado probabilístico. El CAE se describe con más detalle en la sección 14.7.

14.3 Poder de representación, tamaño y profundidad de la capa

Los codificadores automáticos a menudo se entranan con solo un codificador de una sola capa y un decodificador de una sola capa. Sin embargo, esto no es un requisito. De hecho, el uso de codificadores y decodificadores profundos ofrece muchas ventajas.

Recuperar de la sección 6.4.1 que hay muchas ventajas a la profundidad en una red feedforward. Debido a que los codificadores automáticos son redes de avance, estas ventajas también se aplican a los codificadores automáticos. Además, el codificador es en sí mismo una red de alimentación directa al igual que el decodificador, por lo que cada uno de estos componentes del codificador automático puede beneficiarse individualmente de la profundidad.

Una de las principales ventajas de la profundidad no trivial es que el teorema del aproximador universal garantiza que una red neuronal realimentada con al menos una capa oculta puede representar una aproximación de cualquier función (dentro de una clase amplia) a una

grado arbitrario de precisión, siempre que tenga suficientes unidades ocultas. Esto significa que un codificador automático con una sola capa oculta puede representar arbitrariamente bien la función de identidad a lo largo del dominio de los datos. Sin embargo, el mapeo de la entrada al código es poco profundo. Esto significa que no podemos imponer restricciones arbitrarias, como que el código debe ser escaso. Un codificador automático profundo, con al menos una capa oculta adicional dentro del propio codificador, puede aproximar cualquier asignación desde la entrada hasta el código arbitrariamente bien, dadas suficientes unidades ocultas.

La profundidad puede reducir exponencialmente el costo computacional de representar algunas funciones. La profundidad también puede disminuir exponencialmente la cantidad de datos de entrenamiento necesarios para aprender algunas funciones. Mira la sección 6.4.1 para una revisión de las ventajas de la profundidad en las redes feedforward.

Experimentalmente, los codificadores automáticos profundos producen una compresión mucho mejor que los codificadores automáticos superficiales o lineales correspondientes ([Hinton y Salakhutdinov, 2006](#)).

Una estrategia común para entrenar un codificador automático profundo es entrenar previamente con avidez la arquitectura profunda entrenando una pila de codificadores automáticos superficiales, por lo que a menudo nos encontramos con codificadores automáticos superficiales, incluso cuando el objetivo final es entrenar un codificador automático profundo.

14.4 Codificadores y decodificadores estocásticos

Los codificadores automáticos son solo redes de avance. Las mismas funciones de pérdida y tipos de unidades de salida que se pueden usar para redes feedforward tradicionales también se usan para codificadores automáticos.

Como se describe en la sección 6.2.2.4, una estrategia general para diseñar las unidades de salida y la función de pérdida de una red feedforward es definir una distribución de salida $p_{\text{ag}}(y | X)$ y minimizar la log-verosimilitud negativa-registro $p_{\text{ag}}(y | X)$. En ese escenario, y era un vector de objetivos, como etiquetas de clase.

En el caso de un codificador automático, X hora es el objetivo, así como la entrada. Sin embargo, todavía podemos aplicar la misma maquinaria que antes. Dado un código oculto h , podemos pensar que el decodificador proporciona una distribución condicional $p_{\text{ag}}(\text{descifrador}(x) | h)$. Entonces podemos entrenar el codificador automático minimizando $-\text{registro} p_{\text{ag}}(\text{descifrador}(x) | h)$. La forma exacta de esta función de pérdida cambiará dependiendo de la forma de $p_{\text{ag}}(\text{descifrador})$. Al igual que con las redes feedforward tradicionales, generalmente usamos unidades de salida lineales para parametrizar la media de una distribución gaussiana si X es de valor real. En ese caso, la verosimilitud logarítmica negativa produce un criterio de error cuadrático medio. Del mismo modo, binario X valores corresponden a una distribución de Bernoulli cuyos parámetros vienen dados por una unidad de salida sigmoidea, discreta X los valores corresponden a una distribución softmax, y así sucesivamente.

Por lo general, las variables de salida se tratan como condicionalmente independientes dado h por lo que esta distribución de probabilidad es económica de evaluar, pero algunas técnicas, como las salidas de densidad de mezcla, permiten el modelado manejable de salidas con correlaciones.

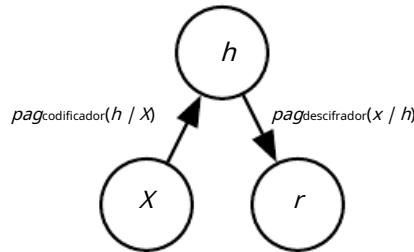


Figura 14.2: La estructura de un codificador automático estocástico, en el que tanto el codificador como el decodificador no son funciones simples, sino que implican una inyección de ruido, lo que significa que su salida puede verse como una muestra de una distribución. $pag_{\text{codificador}}(h / X)$ para el codificador y $pag_{\text{descifrador}}(x / h)$ para el decodificador.

Para hacer una desviación más radical de las redes feedforward que hemos visto anteriormente, también podemos generalizar la noción de **una función de codificación** $F(X)$ a una **distribución de codificación** $pag_{\text{codificador}}(h / X)$, como se ilustra en la figura 14.2.

Cualquier modelo de variable latente $pag_{\text{modelo}}(h, x)$ define un codificador estocástico

$$pag_{\text{codificador}}(h / X) = pag_{\text{modelo}}(h / X) \quad (14.12)$$

y un decodificador estocástico

$$pag_{\text{descifrador}}(x / h) = pag_{\text{modelo}}(x / h). \quad (14.13)$$

En general, las distribuciones de codificador y decodificador no son necesariamente distribuciones condicionales compatibles con una distribución conjunta única $pag_{\text{modelo}}(x, h)$. [alain et al. \(2015\)](#) mostró que entrenar el codificador y el decodificador como un codificador automático de eliminación de ruido tenderá a hacerlos compatibles asintóticamente (con suficiente capacidad y ejemplos).

14.5 Eliminación de ruido de codificadores automáticos

El **codificador automático de eliminación de ruido**(DAE) es un codificador automático que recibe un punto de datos corrupto como entrada y está capacitado para predecir el punto de datos original no corrupto como salida.

El procedimiento de entrenamiento DAE se ilustra en la figura 14.3. Introducimos un proceso de corrupción $C(X/X)$ que representa una distribución condicional sobre

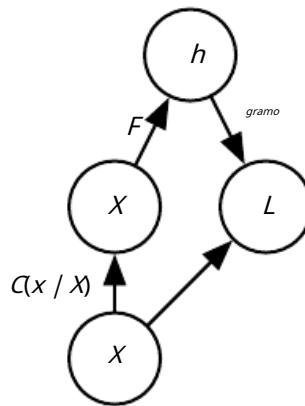


Figura 14.3: El gráfico computacional de la función de costo para un codificador automático de eliminación de ruido, que está entrenado para reconstruir el punto de datos limpio X de su versión corrupta $X̃$. Esto se logra minimizando la pérdida $L = -\text{registro} p_{\text{descifrador}}(x / h = F(X))$, donde X es una versión corrupta del ejemplo de datos X , obtenido a través de un determinado proceso de corrupción $C(x / X)$. Normalmente la distribución $p_{\text{descifrador}}$ es una distribución factorial cuyos parámetros medios son emitidos por una red feedforward gramo .

muestras corruptas \tilde{x} , dada una muestra de datos x . El codificador automático luego aprende un **distribución de reconstrucción** $p_{\text{reconstruir}}(x / \tilde{x})$ estimado a partir de pares de entrenamiento (x, \tilde{x}) , como sigue:

1. Muestra un ejemplo de entrenamiento x de los datos de entrenamiento.
2. Pruebe una versión corrupta \tilde{x} de $C(x / x = \tilde{x})$.
3. Usar (x, \tilde{x}) como ejemplo de entrenamiento para estimar la distribución de reconstrucción del autocodificador $p_{\text{reconstruir}}(x / X) = p_{\text{descifrador}}(x / h)$ con h la salida del codificador $F(X)$ y $p_{\text{descifrador}}$ típicamente definido por un decodificador $\text{gramo}(h)$.

Por lo general, podemos simplemente realizar una minimización aproximada basada en gradientes (como el descenso de gradiente de mini lotes) en la probabilidad logarítmica negativa $-\text{registro} p_{\text{descifrador}}(x / h)$. Siempre que el codificador sea determinista, el autocodificador de eliminación de ruido es una red de alimentación directa y puede entrenarse exactamente con las mismas técnicas que cualquier otra red de alimentación directa.

Por lo tanto, podemos ver el DAE realizando un descenso de gradiente estocástico con la siguiente expectativa:

$$-\text{mix}_{\sim p_{\text{datos}}(X)} \text{mix}_{\sim C(x / x)} \text{registro} p_{\text{descifrador}}(x / h = F(X)) \quad (14.14)$$

dónde $p_{\text{datos}}(X)$ es la distribución de entrenamiento.

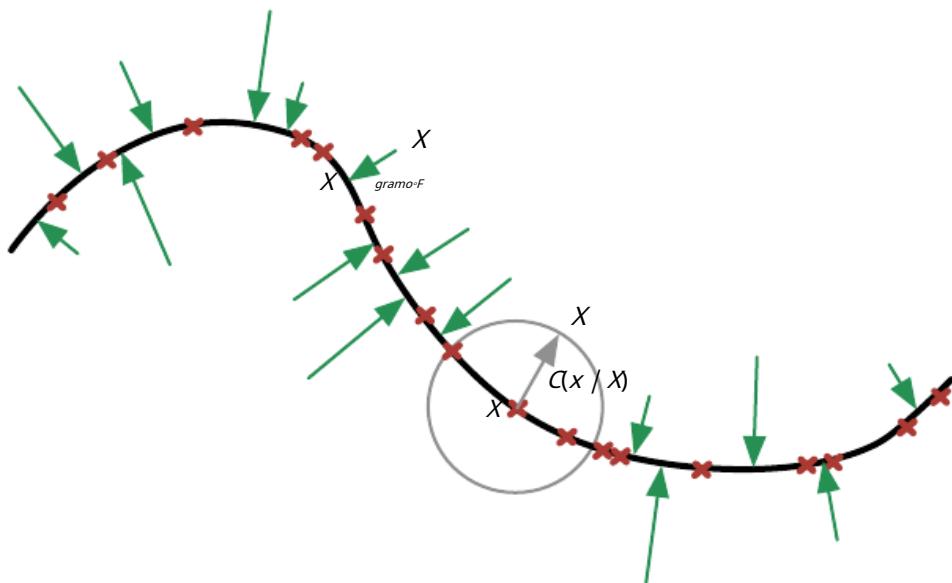


Figura 14.4: Se entrena un codificador automático de eliminación de ruido para mapear un punto de datos corrupto X volver al punto de datos original X . Ilustramos ejemplos de entrenamiento X como cruces rojas que se encuentran cerca de una variedad de baja dimensión ilustrada con la línea negra en negrita. Ilustramos el proceso de corrupción $C(x / X)$ con un círculo gris de corrupciones equiprobables. Una flecha gris demuestra cómo un ejemplo de capacitación se transforma en una muestra de este proceso de corrupción. Cuando el codificador automático de eliminación de ruido se entrena para minimizar el promedio de errores cuadráticos $\|gromo(F(X)) - X\|^2 / 2$, la reconstrucción $gromo(F(X))$ estimados $\text{mix}_{X \sim \text{datos}(X)} C(X/X)$. El vector $gromo(F(X)) - X$ apunta aproximadamente

hacia el punto más cercano en la variedad, ya que $gromo(F(X))$ estima el centro de masa de los puntos limpios X que podría haber dado lugar a X . El autocodificador aprende así un campo vectorial $gromo(F(X)) - X$ indicado por las flechas verdes. Este campo vectorial estima la puntuación $\nabla_{\tilde{X}} \text{registro} \text{pag}_{\text{datos}(X)}$ hasta un factor multiplicativo que es el error de reconstrucción cuadrático medio de la raíz media.

14.5.1 Estimación de la puntuación

Coincidencia de puntuación ([Hyvärinen, 2005](#)) es una alternativa a la máxima verosimilitud. Proporciona un estimador consistente de distribuciones de probabilidad basado en alentar al modelo a tener el mismo **puntaje** como la distribución de datos en cada punto de entrenamiento X . En este contexto, la partitura es un campo de gradiente particular:

$$\nabla_{\text{registro}} \text{pag}(X). \quad (14.15)$$

La coincidencia de puntuación se analiza más adelante en la sección [18.4](#). Para la discusión actual sobre autocodificadores, es suficiente entender que aprender el campo de gradiente $\nabla_{\text{registro}} \text{pag}$ es una manera de aprender la estructura de pag sí mismo.

Una propiedad muy importante de los DAE es que su criterio de entrenamiento (con condiciones condicionalmente gaussianas) $\text{pag}(x / h)$ hace que el codificador automático aprenda un campo vectorial ($\text{gramo}(F(X) - X)$) que estima la puntuación de la distribución de datos. Esto se ilustra en la figura [14.4](#).

Entrenamiento de eliminación de ruido de un tipo específico de autocodificador (unidades ocultas sigmoidales, unidades de reconstrucción lineal) utilizando ruido gaussiano y error cuadrático medio, ya que el costo de reconstrucción es equivalente ([Vicente, 2011](#)) para entrenar un tipo específico de modelo probabilístico no dirigido llamado RBM con unidades visibles gaussianas. Este tipo de modelo se describirá en detalle en la sección [20.5.1](#); para la presente discusión basta saber que es un modelo que proporciona una $\text{pag}_{\text{modelo}}(X; \theta)$. Cuando el RBM se entrena usando **coincidencia de puntuación de eliminación de ruido** ([Kingma y Le Cun, 2010](#)), su algoritmo de aprendizaje es equivalente al entrenamiento de eliminación de ruido en el autocodificador correspondiente. Con un nivel de ruido fijo, la coincidencia de puntuación regularizada no es un estimador consistente; en cambio, recupera una versión borrosa de la distribución. Sin embargo, si se elige que el nivel de ruido se acerque a 0 cuando el número de ejemplos se acerque al infinito, entonces se recupera la consistencia. La coincidencia de puntuación de eliminación de ruido se analiza con más detalle en la sección [18.5](#).

Existen otras conexiones entre autocodificadores y RBM. La igualación de puntajes aplicada a los RBM arroja una función de costo idéntica al error de reconstrucción combinado con un término de regularización similar a la penalización contractiva del CAE ([swersky et al., 2011](#)). [Bengio y Delalleau \(2009\)](#) mostró que un gradiente de autocodificador proporciona una aproximación al entrenamiento de divergencia contrastiva de RBM.

Para valores continuos X , el criterio de eliminación de ruido con corrupción gaussiana y distribución de reconstrucción produce un estimador de la puntuación que es aplicable a las parametrizaciones generales de codificador y decodificador ([Alain y Bengio, 2013](#)). Esto significa que se puede hacer una arquitectura genérica de codificador-decodificador para estimar la puntuación

entrenando con el criterio de error al cuadrado

$$\| \text{gramo}(F(X)) - x \|_2^2 \quad (14.16)$$

y corrupción

$$C(\tilde{x} = \tilde{x}/X) = \text{norte}(X; \mu = X\Sigma = \sigma_2 I) \quad (14.17)$$

con variación de ruido σ_2 . Ver figura 14.5 para una ilustración de cómo funciona esto.

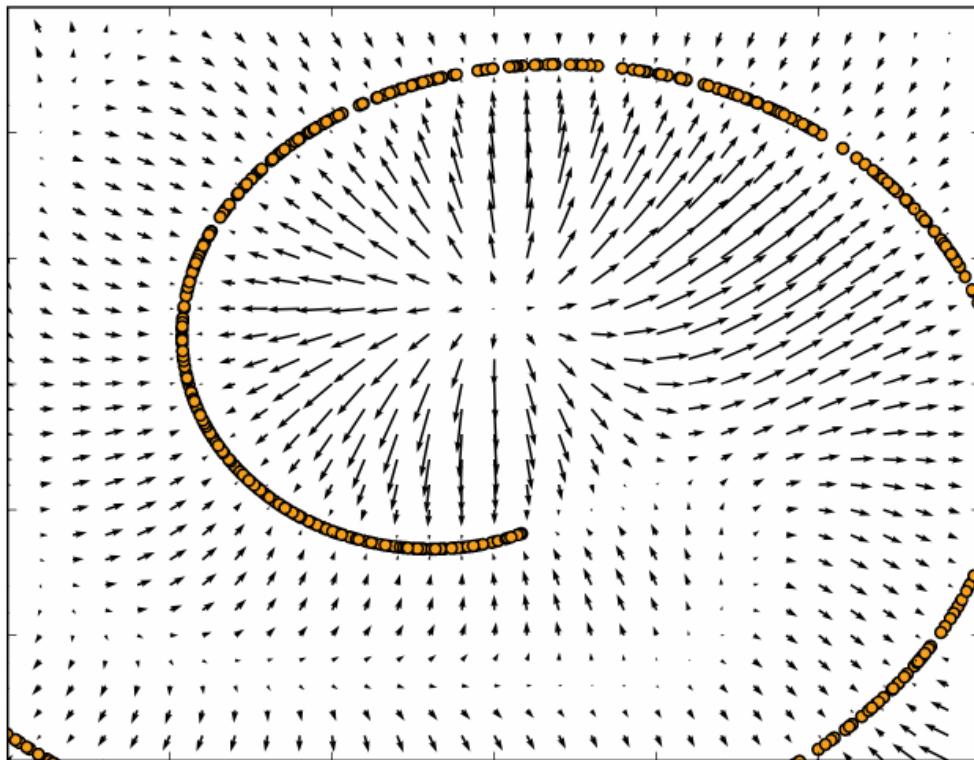


Figura 14.5: Campo vectorial aprendido por un codificador automático que elimina el ruido alrededor de una variedad curva 1-D cerca de la cual los datos se concentran en un espacio 2-D. Cada flecha es proporcional a la reconstrucción menos el vector de entrada del codificador automático y apunta hacia una mayor probabilidad de acuerdo con la distribución de probabilidad implícitamente estimada. El campo vectorial tiene ceros tanto en el máximo de la función de densidad estimada (en las variedades de datos) como en el mínimo de esa función de densidad. Por ejemplo, el brazo espiral forma una variedad unidimensional de máximos locales que están conectados entre sí. Los mínimos locales aparecen cerca de la mitad del espacio entre dos brazos. Cuando la norma del error de reconstrucción (mostrado por la longitud de las flechas) es grande, significa que la probabilidad puede aumentar significativamente moviéndose en la dirección de la flecha, y ese es principalmente el caso en lugares de baja probabilidad. El codificador automático asigna estos puntos de baja probabilidad a reconstrucciones de mayor probabilidad. Cuando la probabilidad es máxima, las flechas se reducen porque la reconstrucción se vuelve más precisa. Figura reproducida con permiso de Alain y Bengio(2013).

En general, no hay garantía de que la reconstrucción $\text{gramo}(F(X))$ menos la entrada X corresponde al gradiente de cualquier función, y mucho menos a la partitura. Eso es

por qué los primeros resultados (Vicente, 2011) están especializados en parametrizaciones particulares donde $gramo(R(X)) - X$ puede obtenerse tomando la derivada de otra función. Kamyshanska y Memisevic (2015) generalizaron los resultados de Vicente (2011) mediante la identificación de una familia de autocodificadores superficiales tales que $gramo(R(X)) - X$ corresponde a una puntuación para todos los miembros de la familia.

Hasta ahora solo hemos descrito cómo el codificador automático que elimina el ruido aprende a representar una distribución de probabilidad. En términos más generales, es posible que desee utilizar el codificador automático como modelo generativo y extraer muestras de esta distribución. Esto se describirá más adelante, en la sección 20.11.

14.5.1.1 Perspectiva histórica

La idea de usar MLP para eliminar el ruido se remonta al trabajo de lecun(1987) y Gallinari et al.(1987). Behnke (2001) también utilizó redes recurrentes para eliminar el ruido de las imágenes. Los codificadores automáticos de eliminación de ruido son, en cierto sentido, solo MLP entrenados para eliminar ruido. Sin embargo, el nombre "codificador automático de eliminación de ruido" se refiere a un modelo que no solo pretende aprender a eliminar el ruido de su entrada, sino aprender una buena representación interna como efecto secundario de aprender a eliminar el ruido. Esta idea vino mucho más tarde (Vicente et al., 2008, 2010). La representación aprendida se puede usar para entrenar previamente una red no supervisada más profunda o una red supervisada. Al igual que los codificadores automáticos escasos, la codificación escasa, los codificadores automáticos contractivos y otros codificadores automáticos regularizados, la motivación de los DAE era permitir el aprendizaje de un codificador de muy alta capacidad y evitar que el codificador y el decodificador aprendieran una función de identidad inútil.

Antes de la introducción del DAE moderno, Inayoshi y Kurita (2005) exploraron algunos de los mismos objetivos con algunos de los mismos métodos. Su enfoque minimiza el error de reconstrucción además de un objetivo supervisado mientras inyecta ruido en la capa oculta de un MLP supervisado, con el objetivo de mejorar la generalización al introducir el error de reconstrucción y el ruido inyectado. Sin embargo, su método se basaba en un codificador lineal y no podía aprender familias de funciones tan potentes como el DAE moderno.

14.6 Colectores de aprendizaje con codificadores automáticos

Al igual que muchos otros algoritmos de aprendizaje automático, los codificadores automáticos explotan la idea de que los datos se concentran en una variedad de baja dimensión o en un pequeño conjunto de tales variedades, como se describe en la sección 5.11.3. Algunos algoritmos de aprendizaje automático explotan esta idea solo en la medida en que aprenden una función que se comporta correctamente en la variedad pero que puede tener un comportamiento inusual si se les da una entrada que está fuera de la variedad.

Los codificadores automáticos llevan esta idea más allá y tienen como objetivo aprender la estructura de la variedad.

Para comprender cómo hacen esto los codificadores automáticos, debemos presentar algunas características importantes de las variedades.

Una caracterización importante de una variedad es el conjunto de sus **planos tangentes**. En un punto x en una variedad dimensional, el plano tangente viene dado por vectores base que abarcan las direcciones locales de variación permitidas en la variedad. Como se ilustra en la figura 14.6, estas direcciones locales especifican cómo se puede cambiar x infinitesimalmente mientras permanece en la variedad.

Todos los procedimientos de entrenamiento de autocodificador implican un compromiso entre dos fuerzas:

1. Aprender una representación h de un ejemplo de entrenamiento x tal que se puede recuperar aproximadamente x a través de un decodificador. El hecho de que se extrae de los datos de entrenamiento es crucial, porque significa que el codificador automático no necesita reconstruir con éxito las entradas que no son probables bajo la distribución de generación de datos.
2. Satisfacer la restricción o sanción de regularización. Esto puede ser una restricción arquitectónica que limite la capacidad del autocodificador, o puede ser un término de regularización agregado al costo de reconstrucción. Estas técnicas generalmente prefieren soluciones que son menos sensibles a la entrada.

Claramente, ni la fuerza por sí sola sería útil: copiar la entrada a la salida no es útil por sí solo, ni ignorar la entrada. En cambio, las dos fuerzas juntas son útiles porque obligan a la representación oculta a capturar información sobre la estructura de la distribución de generación de datos. El principio importante es que el codificador automático puede permitirse representar *solo las variaciones que se necesitan para reconstruir ejemplos de entrenamiento*. Si la distribución de generación de datos se concentra cerca de una variedad de baja dimensión, esto produce representaciones que capturan implícitamente un sistema de coordenadas local para esta variedad: solo las variaciones tangentes a la variedad alrededor de x deben corresponder a los cambios en $h = f(x)$. Por lo tanto, el codificador aprende un mapeo del espacio de entrada x a un espacio de representación, un mapeo que solo es sensible a los cambios a lo largo de las múltiples direcciones, pero que es insensible a los cambios ortogonales a la variedad.

Un ejemplo unidimensional se ilustra en la figura 14.7, mostrando que, al hacer que la función de reconstrucción sea insensible a las perturbaciones de la entrada alrededor de los puntos de datos, hacemos que el codificador automático recupere la estructura múltiple.

Para comprender por qué los codificadores automáticos son útiles para el aprendizaje múltiple, es instructivo compararlos con otros enfoques. Lo que se aprende más comúnmente para caracterizar una variedad es una **representación** de los puntos de datos en (o cerca)

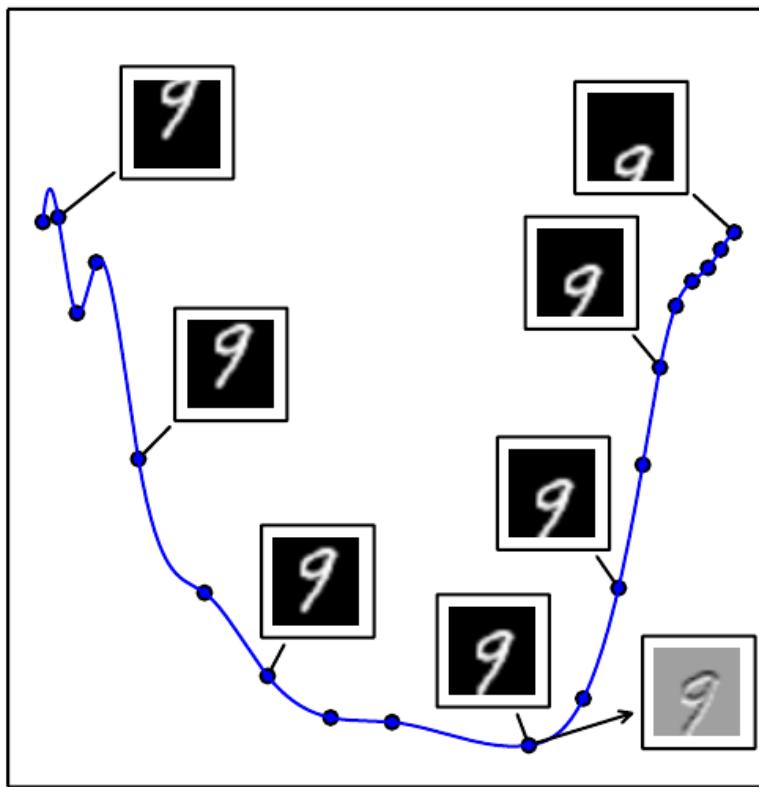


Figura 14.6: Una ilustración del concepto de hiperplano tangente. Aquí creamos una variedad unidimensional en un espacio de 784 dimensiones. Tomamos una imagen MNIST con 784 píxeles y la transformamos traduciéndola verticalmente. La cantidad de traslación vertical define una coordenada a lo largo de una variedad unidimensional que traza una trayectoria curva a través del espacio de la imagen. Esta gráfica muestra algunos puntos a lo largo de esta variedad. Para la visualización, hemos proyectado la variedad en un espacio bidimensional usando PCA. Un *norte*-variedad dimensional tiene un *norte*plano tangente -dimensional en cada punto. Este plano tangente toca la variedad exactamente en ese punto y está orientado paralelo a la superficie en ese punto. Define el espacio de direcciones en el que es posible moverse permaneciendo en la variedad. Esta variedad unidimensional tiene una sola línea tangente. Indicamos un ejemplo de línea tangente en un punto, con una imagen que muestra cómo aparece esta dirección tangente en el espacio de la imagen. Los píxeles grises indican píxeles que no cambian a medida que nos movemos a lo largo de la línea tangente, los píxeles blancos indican píxeles que se iluminan y los píxeles negros indican píxeles que se oscurecen.

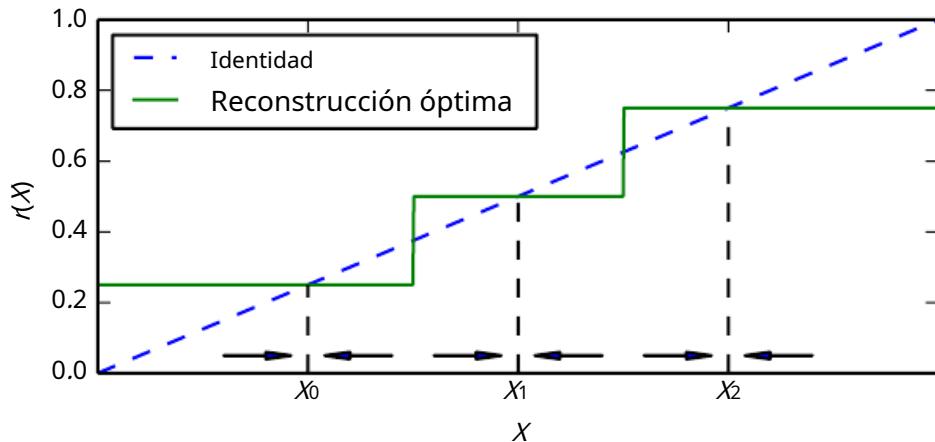


Figura 14.7: Si el codificador automático aprende una función de reconstrucción que es invariable a pequeñas perturbaciones cerca de los puntos de datos, captura la estructura múltiple de los datos. Aquí la estructura múltiple es una colección de variedades dimensionales. La línea diagonal discontinua indica el objetivo de la función de identidad para la reconstrucción. La función de reconstrucción óptima cruza la función de identidad dondequiera que haya un punto de datos. Las flechas horizontales en la parte inferior del gráfico indican el $r(X)$ -vector de dirección de reconstrucción en la base de la flecha, en el espacio de entrada, siempre apuntando hacia la "variedad" más cercana (un solo punto de datos, en el caso de 1-D). El codificador automático de eliminación de ruido trata explícitamente de hacer que la derivada de la función de reconstrucción $r(X)$ sea pequeño alrededor de los puntos de datos. El autocodificador contractivo hace lo mismo con el codificador. Aunque la derivada de $r(X)$ se pide que sea pequeño alrededor de los puntos de datos, puede ser grande entre los puntos de datos. El espacio entre los puntos de datos corresponde a la región entre las variedades, donde la función de reconstrucción debe tener una gran derivada para mapear los puntos dañados de nuevo en la variedad.

el múltiple Tal representación para un ejemplo particular también se llama su incrustación. Por lo general, viene dado por un vector de baja dimensión, con menos dimensiones que el espacio "ambiente" del cual la variedad es un subconjunto de baja dimensión. Algunos algoritmos (algoritmos de aprendizaje múltiple no paramétricos, que se analizan a continuación) aprenden directamente una incrustación para cada ejemplo de entrenamiento, mientras que otros aprenden un mapeo más general, a veces llamado codificador o función de representación, que mapea cualquier punto en el espacio ambiental (la entrada espacio) a su incrustación.

El aprendizaje múltiple se ha centrado principalmente en procedimientos de aprendizaje no supervisados que intentan capturar estos múltiples. La mayor parte de la investigación inicial de aprendizaje automático sobre el aprendizaje de variedades no lineales se ha centrado en **no paramétricos** métodos basados en **gráfico de vecino más cercano**. Este gráfico tiene un nodo por ejemplo de entrenamiento y bordes que conectan vecinos cercanos entre sí. Estos métodos (Scholkopf *et al.*, 1998; Roweis y Saúl, 2000; Tenenbaum *et al.*, 2000; Marca, 2003; Belkin

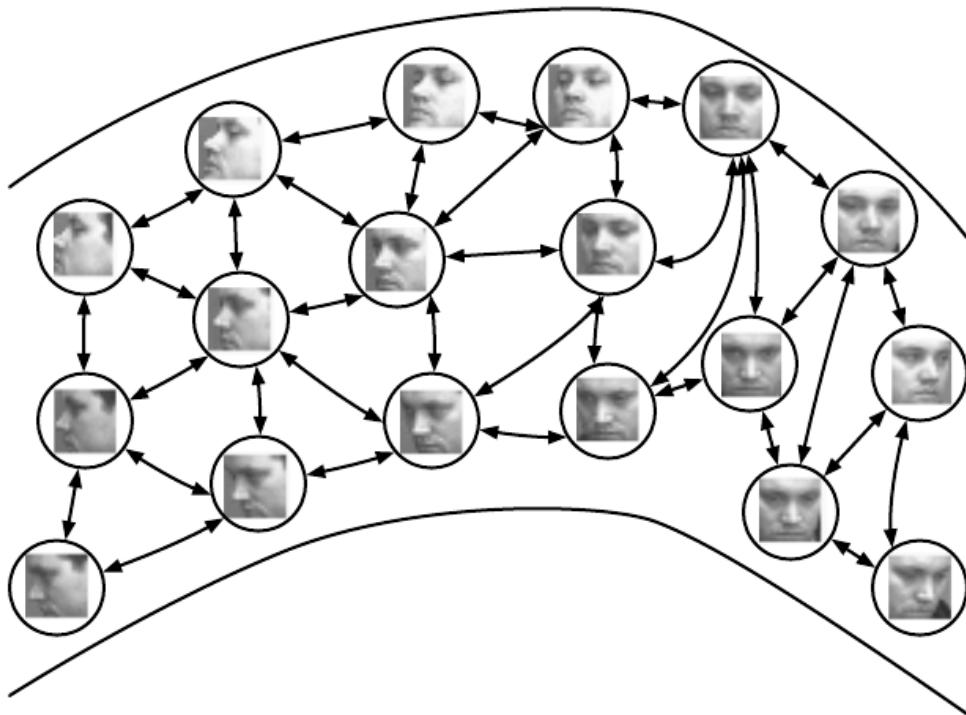


Figura 14.8: Los procedimientos de aprendizaje múltiple no paramétrico construyen un gráfico de vecino más cercano en el que los nodos representan ejemplos de entrenamiento y los bordes dirigidos indican las relaciones de vecino más cercano. Varios procedimientos pueden así obtener el plano tangente asociado con una vecindad del gráfico, así como un sistema de coordenadas que asocia cada ejemplo de entrenamiento con una posición de vector de valor real, o **incrustación**. Es posible generalizar tal representación a nuevos ejemplos mediante una forma de interpolación. Siempre que el número de ejemplos sea lo suficientemente grande como para cubrir la curvatura y los giros de la variedad, estos enfoques funcionan bien. Imágenes del QMUL Multiview Face Dataset ([Gong et al., 2000](#)).

y Niyogi, 2003; Donoho y Grimes, 2003; Weinberger y Saúl, 2004; Hinton y Roweis, 2003; van der Maaten y Hinton, 2008) asocie cada uno de los nodos con un plano tangente que atravesie las direcciones de variaciones asociadas con los vectores de diferencia entre el ejemplo y sus vecinos, como se ilustra en la figura 14.8.

Entonces se puede obtener un sistema de coordenadas global mediante una optimización o resolviendo un sistema lineal. Cifra 14.9 ilustra cómo se puede teselar una variedad mediante una gran cantidad de parches de tipo gaussiano localmente lineales (o "panqueques", porque los gaussianos son planos en las direcciones tangentes).

Sin embargo, existe una dificultad fundamental con tales enfoques no paramétricos locales para el aprendizaje múltiple, planteada en Bengio y Monperrus (2005): si las variedades no son muy suaves (tienen muchos picos, valles y giros), es posible que se necesite una gran cantidad de ejemplos de entrenamiento para cubrir cada uno de ellos.

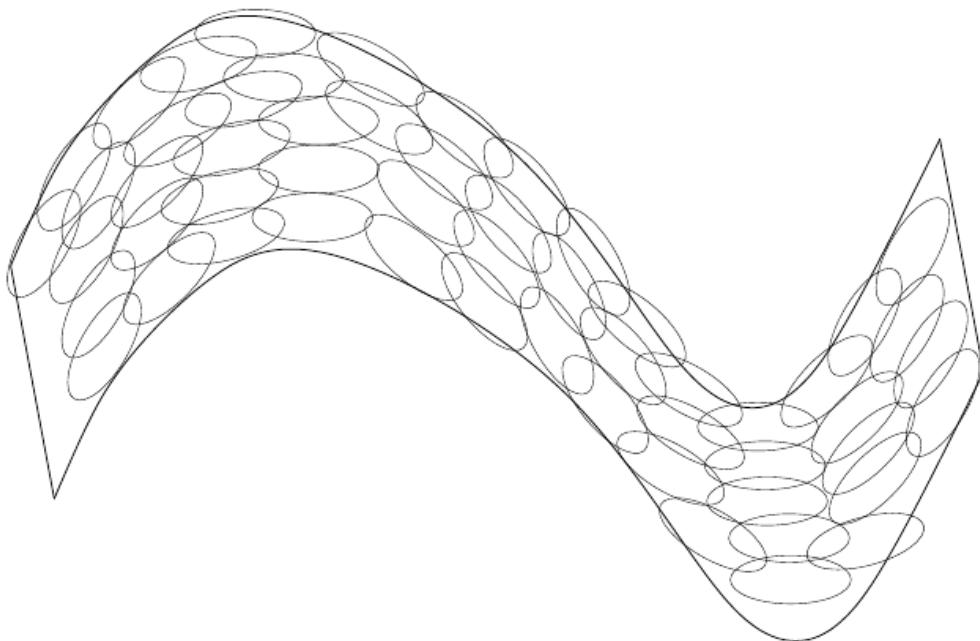


Figura 14.9: Si los planos tangentes (ver figura14.6) en cada ubicación se conocen, luego se pueden teselar para formar un sistema de coordenadas global o una función de densidad. Cada parche local se puede considerar como un sistema de coordenadas euclidianas local o como un gaussiano localmente plano, o "panqueque", con una variación muy pequeña en las direcciones ortogonales al panqueque y una variación muy grande en las direcciones que definen el sistema de coordenadas en el panqueque Una mezcla de estas gaussianas proporciona una función de densidad estimada, como en el algoritmo de ventana múltiple de Parzen ([Vicente y Bengio,2003](#)) o su variante basada en red neuronal no local ([bengio et al.,2006c](#)).

estas variaciones, sin posibilidad de generalizar a variaciones invisibles. De hecho, estos métodos solo pueden generalizar la forma de la variedad interpolando entre ejemplos vecinos. Desafortunadamente, los múltiples involucrados en los problemas de IA pueden tener una estructura muy complicada que puede ser difícil de capturar solo a partir de la interpolación local. Considere, por ejemplo, la variedad resultante de la traducción que se muestra en la figura14.6. Si observamos solo una coordenada dentro del vector de entrada, X_i , a medida que se traduce la imagen, observaremos que una coordenada encuentra un pico o un valle en su valor una vez por cada pico o valle en el brillo de la imagen. En otras palabras, la complejidad de los patrones de brillo en una plantilla de imagen subyacente impulsa la complejidad de las variedades que se generan al realizar transformaciones de imagen simples. Esto motiva el uso de representaciones distribuidas y aprendizaje profundo para capturar estructuras múltiples.

14.7 Autocodificadores contractivos

El autocodificador contractivo ([rifai et al., 2011a,b](#)) introduce un regularizador explícito en el código $h = f(X)$, fomentando los derivados de ser lo más pequeño posible:

$$\Omega(h) = \lambda \cdot \frac{\|\nabla f(X)\|_F^2}{F}. \quad (14.18)$$

La penalidad $\Omega(h)$ es la norma de Frobenius al cuadrado (suma de elementos al cuadrado) de la matriz jacobiana de derivadas parciales asociada a la función codificador.

Existe una conexión entre el autoencoder de eliminación de ruido y el autoencoder contractivo: [Alain y Bengio\(2013\)](#) mostró que en el límite de un pequeño ruido de entrada gaussiano, el error de reconstrucción de eliminación de ruido es equivalente a una penalización contractiva en la función de reconstrucción que mapea $x \mapsto g(\nabla f(X)x)$. En otras palabras, los codificadores automáticos de eliminación de ruido hacen que la función de reconstrucción resista perturbaciones pequeñas pero de tamaño finito de la entrada, mientras que los codificadores automáticos contractivos hacen que la función de extracción de características resista perturbaciones infinitesimales de la entrada. Al usar la penalización contractiva basada en jacobiano para preentrenar características $f(X)$ para usar con un clasificador, la mejor precisión de clasificación generalmente resulta de aplicar la penalización contractiva a $f(X)$ en lugar de $g(\nabla f(X))$. Una sanción contractiva sobre $f(X)$ también tiene conexiones cercanas con la coincidencia de puntajes, como se analiza en la sección [14.5.1](#).

El nombre **contractivo** surge de la forma en que el CAE deforma el espacio. Específicamente, debido a que el CAE está entrenado para resistir las perturbaciones de su entrada, se recomienda mapear una vecindad de puntos de entrada a una vecindad más pequeña de puntos de salida. Podemos pensar en esto como contraer la vecindad de entrada a una vecindad de salida más pequeña.

Para aclarar, el CAE es contractivo solo localmente: todas las perturbaciones de un punto de entrenamiento X están mapeados cerca de $f(X)$. Globalmente, dos puntos diferentes X y X' se puede asignar a $f(X)$ y $f(X')$ puntos que están más separados que los puntos originales. Es plausible que f expandirse entre o lejos de las variedades de datos (ver, por ejemplo, lo que sucede en el ejemplo de juguete 1-D de la figura [14.7](#)). Cuando el $\Omega(h)$ Si se aplica una penalización a las unidades sigmoideas, una manera fácil de reducir el jacobiano es hacer que las unidades sigmoideas se saturen a 0 o 1. Esto anima al CAE a codificar puntos de entrada con valores extremos del sigmoide que pueden interpretarse como un código binario. También garantiza que el CAE distribuirá sus valores de código en la mayor parte del hipercubo que pueden abarcar sus unidades ocultas sigmoidales.

Podemos pensar en la matriz jacobiana en un punto X como aproximación al codificador no lineal $f(X)$ como un operador lineal. Esto nos permite usar la palabra “contractiva” más formalmente. En la teoría de los operadores lineales, un operador lineal

Se dice que es contractivo si la norma de λ permanece menor o igual que 1 para todas las unidades-norma X . En otras palabras, es contractivo si encoge la esfera unitaria. Podemos pensar en el CAE como penalizando la norma de Frobenius de la aproximación lineal local de $\mathcal{F}(X)$ en cada punto de entrenamiento X para alentar a cada uno de estos operadores lineales locales a convertirse en una contracción.

Como se describe en la sección 14.6, los codificadores automáticos regularizados aprenden múltiples al equilibrar dos fuerzas opuestas. En el caso del CAE, estas dos fuerzas son el error de reconstrucción y la penalización contractiva $\Omega(h)$. El error de reconstrucción por sí solo alentaría al CAE a aprender una función de identidad. La penalización contractiva por sí sola alentaría al CAE a aprender características que son constantes con respecto a X . El compromiso entre estas dos fuerzas produce un autocodificador cuyas derivadas $\frac{\partial \mathcal{F}(X)}{\partial X}$ son en su mayoría diminutas. Solo un pequeño número de unidades ocultas, correspondientes a un pequeño número de direcciones en la entrada, pueden tener derivadas significativas.

El objetivo del CAE es aprender la estructura múltiple de los datos. Direcciones X con gran λ cambian rápidamente h , por lo que es probable que estas sean direcciones que se aproximen a los planos tangentes de la variedad. Experimentos por rifaiet al. (2011a) y rifaiet al. (2011b) muestran que entrenar el CAE da como resultado la mayoría de los valores singulares de $\mathcal{F}(X)$ cayendo debajo de 1 en magnitud y por lo tanto volverse contractiva. Sin embargo, algunos valores singulares permanecen por encima de 1, porque la penalización por error de reconstrucción anima al CAE a codificar las direcciones con la mayor variación local. Las direcciones correspondientes a los valores singulares más grandes se interpretan como las direcciones tangentes que ha aprendido el autocodificador contractivo. Idealmente, estas direcciones tangentes deberían corresponder a variaciones reales en los datos. Por ejemplo, un CAE aplicado a imágenes debe aprender vectores tangentes que muestren cómo cambia la imagen a medida que los objetos en la imagen cambian gradualmente de posición, como se muestra en la figura 14.6. Las visualizaciones de los vectores singulares obtenidos experimentalmente parecen corresponder a transformaciones significativas de la imagen de entrada, como se muestra en la figura 14.10.

Un problema práctico con el criterio de regularización CAE es que, aunque es barato de calcular en el caso de un codificador automático de una sola capa oculta, se vuelve mucho más costoso en el caso de codificadores automáticos más profundos. La estrategia seguida por rifaiet al. (2011a) es entrenar por separado una serie de codificadores automáticos de una sola capa, cada uno entrenado para reconstruir la capa oculta del codificador automático anterior. La composición de estos codificadores automáticos forma un codificador automático profundo. Debido a que cada capa se entrenó por separado para ser contractiva localmente, el codificador automático profundo también es contractivo. El resultado no es el mismo que se obtendría entrenando conjuntamente toda la arquitectura con una penalización sobre el jacobiano del modelo profundo, pero captura muchas de las características cualitativas deseables.

Otra cuestión práctica es que la penalización por contracción puede obtener resultados inútiles

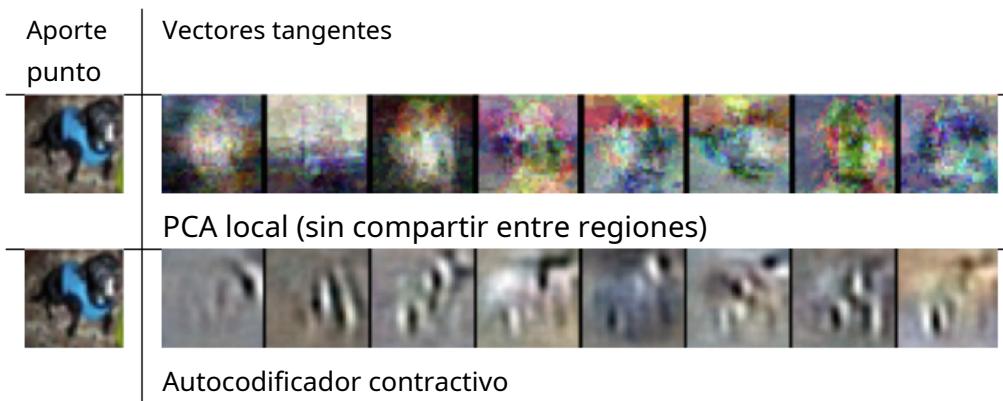


Figura 14.10: Ilustración de vectores tangentes de la variedad estimados por PCA local y por un autocodificador contractivo. La ubicación en el colector está definida por la imagen de entrada de un perro extraído del conjunto de datos CIFAR-10. Los vectores tangentes se estiman por los principales vectores singulares de la matriz jacobiana $\partial h / \partial \theta$ del mapeo de entrada a código. Aunque tanto el PCA local como el CAE pueden capturar tangentes locales, el CAE puede generar estimaciones más precisas a partir de datos de entrenamiento limitados porque aprovecha el uso compartido de parámetros en diferentes ubicaciones que comparten un subconjunto de unidades ocultas activas. Las direcciones tangentes CAE normalmente corresponden a partes móviles o cambiantes del objeto (como la cabeza o las piernas). Imágenes reproducidas con permiso de [rifai et al. \(2011c\)](#).

si no imponemos algún tipo de escala en el decodificador. Por ejemplo, el codificador podría consistir en multiplicar la entrada por una pequeña constante y el decodificador podría consistir en dividir el código por \sqrt{N} . Como enfoques 0, el codificador impulsa la penalización contractiva $\Omega(h)$ acercándose a 0 sin haber aprendido nada sobre la distribución. Mientras tanto, el decodificador mantiene una reconstrucción perfecta. En [rifai et al. \(2011a\)](#), esto se evita atando los pesos de F y g . Ambos F y g son capas de red neuronal estándar que consisten en una transformación afín seguida de una no linealidad por elementos, por lo que es sencillo establecer la matriz de peso de g como ser la transpuesta de la matriz de pesos de F .

14.8 Descomposición dispersa predictiva

Descomposición dispersa predictiva (PSD) es un modelo que es un híbrido de codificación dispersa y codificadores automáticos paramétricos ([Kavukcuoglu et al., 2008](#)). Se entrena un codificador paramétrico para predecir el resultado de la inferencia iterativa. PSD se ha aplicado al aprendizaje de características no supervisado para el reconocimiento de objetos en imágenes y video ([Kavukcuoglu et al., 2009, 2010; Jarrett et al., 2009; Farabet et al., 2011](#)), así como para audio ([Henaff et al., 2011](#)). El modelo consta de un codificador $F(X)$ y un decodificador $g(m(h))$; ambos son paramétricos. Durante el entrenamiento, h es controlado por el

algoritmo de optimización. El entrenamiento procede minimizando

$$\|x - \text{gramo}(h)\|^2 + \lambda \|h\|_1 + \gamma \|h - f(X)\|^2. \quad (14.19)$$

Al igual que en la codificación dispersa, el algoritmo de entrenamiento alterna entre minimización con respecto a h y minimización con respecto a los parámetros del modelo. Minimización con respecto a h es rápido porque $f(X)$ proporciona un buen valor inicial de h y la función de costo restringe h permanecer cerca $f(X)$ de todos modos. El descenso de gradiente simple puede obtener valores razonables de h en tan solo diez pasos.

El procedimiento de entrenamiento utilizado por PSD es diferente de entrenar primero un modelo de codificación dispersa y luego entrenar $f(X)$ para predecir los valores de las características de codificación dispersas. El procedimiento de entrenamiento PSD regulariza el decodificador para usar parámetros para los cuales $f(X)$ puede inferir buenos valores de código.

La codificación dispersa predictiva es un ejemplo de **inferencia aproximada aprendida**. En la sección 19.5, este tema se desarrolla más. Las herramientas presentadas en el capítulo 19 deje en claro que PSD puede interpretarse como el entrenamiento de un modelo probabilístico de codificación dispersa dirigida al maximizar un límite inferior en la verosimilitud logarítmica del modelo.

En aplicaciones prácticas de PSD, la optimización iterativa solo se usa durante el entrenamiento. El codificador paramétrico f se utiliza para calcular las funciones aprendidas cuando se implementa el modelo, evaluando f computacionalmente económico en comparación con inferir h mediante descenso de gradiente. Porque f es una función paramétrica diferenciable, los modelos PSD se pueden apilar y usar para inicializar una red profunda para entrenarla con otro criterio.

14.9 Aplicaciones de Autoencoders

Los autocodificadores se han aplicado con éxito a tareas de reducción de dimensionalidad y recuperación de información. La reducción de la dimensionalidad fue una de las primeras aplicaciones del aprendizaje de representación y el aprendizaje profundo. Fue una de las primeras motivaciones para estudiar los codificadores automáticos. Por ejemplo, [Hinton y Salakhutdinov \(2006\)](#) entrenó una pila de RBM y luego usó sus pesos para inicializar un codificador automático profundo con capas ocultas gradualmente más pequeñas, que culminó en un cuello de botella de 30 unidades. El código resultante produjo menos errores de reconstrucción que PCA en 30 dimensiones y la representación aprendida fue cualitativamente más fácil de interpretar y relacionar con las categorías subyacentes, manifestándose estas categorías como grupos bien separados.

Las representaciones de menor dimensión pueden mejorar el rendimiento en muchas tareas, como la clasificación. Los modelos de espacios más pequeños consumen menos memoria y tiempo de ejecución.

Muchas formas de reducción de dimensionalidad colocan ejemplos semánticamente relacionados uno cerca del otro, como lo observan Salakhutdinov y Hinton (2007b) y Torralba et al. (2008). Las pistas proporcionadas por el mapeo al espacio de dimensiones inferiores ayudan a la generalización.

Una tarea que se beneficia incluso más de lo habitual de la reducción de la dimensionalidad es **recuperación de información**, la tarea de encontrar entradas en una base de datos que se parezcan a una entrada de consulta. Esta tarea obtiene los beneficios habituales de la reducción de dimensionalidad que obtienen otras tareas, pero también obtiene el beneficio adicional de que la búsqueda puede volverse extremadamente eficiente en ciertos tipos de espacios de baja dimensión. Específicamente, si entrenamos el algoritmo de reducción de dimensionalidad para producir un código de baja dimensión *y binario*, entonces podemos almacenar todas las entradas de la base de datos en una tabla hash que asigna vectores de código binario a las entradas. Esta tabla hash nos permite realizar la recuperación de información devolviendo todas las entradas de la base de datos que tienen el mismo código binario que la consulta. También podemos buscar entradas un poco menos similares de manera muy eficiente, simplemente volteando bits individuales de la codificación de la consulta. Este enfoque para la recuperación de información a través de la reducción de la dimensionalidad y la binarización se denomina **hashing semántico** (Salakhutdinov y Hinton, 2007b, 2009b), y se ha aplicado tanto a la entrada de texto (Salakhutdinov y Hinton, 2007b, 2009b) e imágenes (Torralba et al., 2008; Weiss et al., 2008; Krizhevsky y Hinton, 2011).

Para producir códigos binarios para hash semántico, normalmente se usa una función de codificación con sigmoides en la capa final. Las unidades sigmoideas deben ser entrenadas para estar saturadas a casi 0 o casi 1 para todos los valores de entrada. Un truco que puede lograr esto es simplemente injectar ruido aditivo justo antes de la no linealidad sigmoidea durante el entrenamiento. La magnitud del ruido debería aumentar con el tiempo. Para combatir ese ruido y preservar la mayor cantidad de información posible, la red debe aumentar la magnitud de las entradas a la función sigmoidea, hasta que se produzca la saturación.

La idea de aprender una función hash se ha explorado más a fondo en varias direcciones, incluida la idea de entrenar las representaciones para optimizar una pérdida más directamente relacionada con la tarea de encontrar ejemplos cercanos en la tabla hash (Norouzi y Flota, 2011).

Capítulo 15

Representación Aprendizaje

En este capítulo, primero discutimos qué significa aprender representaciones y cómo la noción de representación puede ser útil para diseñar arquitecturas profundas. Discutimos cómo los algoritmos de aprendizaje comparten la fuerza estadística en diferentes tareas, incluido el uso de información de tareas no supervisadas para realizar tareas supervisadas. Las representaciones compartidas son útiles para manejar múltiples modalidades o dominios, o para transferir el conocimiento aprendido a tareas para las que se dan pocos o ningún ejemplo, pero existe una representación de tareas. Finalmente, damos un paso atrás y discutimos las razones del éxito del aprendizaje de representaciones, comenzando con las ventajas teóricas de las representaciones distribuidas ([Hinton et al., 1986](#)) y representaciones profundas y terminando con la idea más general de supuestos subyacentes sobre el proceso de generación de datos, en particular sobre las causas subyacentes de los datos observados.

Muchas tareas de procesamiento de información pueden ser muy fáciles o muy difíciles dependiendo de cómo se represente la información. Este es un principio general aplicable a la vida diaria, la informática en general y el aprendizaje automático. Por ejemplo, es sencillo para una persona dividir 210 entre 6 usando una división larga. La tarea se vuelve considerablemente menos sencilla si, en cambio, se plantea utilizando la representación en números romanos de los números. La mayoría de las personas modernas a las que se les pide que dividan CCX por VI comenzarían convirtiendo los números a la representación de números arábigos, lo que permite largos procedimientos de división que hacen uso del sistema de valor posicional. Más concretamente, podemos cuantificar el tiempo de ejecución asintótico de varias operaciones usando representaciones apropiadas o inapropiadas. Por ejemplo, $\mathcal{O}(\text{norte})$ operación si la lista se representa como una lista enlazada, pero sólo $\mathcal{O}(\text{registro norte})$ si la lista se representa como un árbol rojo-negro.

En el contexto del aprendizaje automático, ¿qué hace que una representación sea mejor que

¿otro? En términos generales, una buena representación es aquella que facilita una tarea de aprendizaje posterior. La elección de la representación dependerá normalmente de la elección de la tarea de aprendizaje posterior.

Podemos pensar en las redes feedforward entrenadas por aprendizaje supervisado como si realizaran un tipo de aprendizaje de representación. Específicamente, la última capa de la red suele ser un clasificador lineal, como un clasificador de regresión softmax. El resto de la red aprende a proporcionar una representación a este clasificador. El entrenamiento con un criterio supervisado conduce naturalmente a que la representación en cada capa oculta (pero más cerca de la capa oculta superior) adquiera propiedades que facilitan la tarea de clasificación. Por ejemplo, las clases que no eran linealmente separables en las entidades de entrada pueden volverse linealmente separables en la última capa oculta. En principio, la última capa podría ser otro tipo de modelo, como un clasificador vecino más cercano ([Salakhutdinov y Hinton, 2007a](#)). Las entidades en la penúltima capa deben aprender diferentes propiedades según el tipo de la última capa.

El entrenamiento supervisado de redes feedforward no implica imponer explícitamente ninguna condición sobre las características intermedias aprendidas. Otros tipos de algoritmos de aprendizaje de representación a menudo se diseñan explícitamente para dar forma a la representación de alguna manera particular. Por ejemplo, supongamos que queremos aprender una representación que facilite la estimación de la densidad. Las distribuciones con más independencias son más fáciles de modelar, por lo que podríamos diseñar una función objetivo que fomente los elementos del vector de representación h para ser independiente. Al igual que las redes supervisadas, los algoritmos de aprendizaje profundo no supervisados tienen un objetivo de entrenamiento principal, pero también aprenden una representación como efecto secundario. Independientemente de cómo se obtuvo una representación, se puede utilizar para otra tarea. Alternativamente, se pueden aprender múltiples tareas (algunas supervisadas, otras no supervisadas) junto con alguna representación interna compartida.

La mayoría de los problemas de aprendizaje de representación se enfrentan a un compromiso entre conservar la mayor cantidad posible de información sobre la entrada y obtener buenas propiedades (como la independencia).

El aprendizaje de representación es particularmente interesante porque proporciona una forma de realizar aprendizaje no supervisado y semisupervisado. A menudo tenemos grandes cantidades de datos de entrenamiento sin etiquetar y relativamente pocos datos de entrenamiento etiquetados. El entrenamiento con técnicas de aprendizaje supervisado en el subconjunto etiquetado a menudo da como resultado un sobreajuste severo. El aprendizaje semisupervisado ofrece la oportunidad de resolver este problema de sobreajuste al aprender también de los datos no etiquetados. Específicamente, podemos aprender buenas representaciones para los datos no etiquetados y luego usar estas representaciones para resolver la tarea de aprendizaje supervisado.

Los humanos y los animales pueden aprender de muy pocos ejemplos etiquetados. Hacemos

Todavía no sé cómo es esto posible. Muchos factores podrían explicar el desempeño humano mejorado; por ejemplo, el cerebro puede usar conjuntos muy grandes de clasificadores o técnicas de inferencia bayesianas. Una hipótesis popular es que el cerebro puede aprovechar el aprendizaje no supervisado o semisupervisado. Hay muchas maneras de aprovechar los datos sin etiquetar. En este capítulo, nos enfocamos en la hipótesis de que los datos no etiquetados pueden usarse para aprender una buena representación.

15.1 Preentrenamiento sin supervisión Greedy Layer-Wise

El aprendizaje no supervisado desempeñó un papel histórico clave en el renacimiento de las redes neuronales profundas, lo que permitió a los investigadores por primera vez entrenar una red supervisada profunda sin necesidad de especializaciones arquitectónicas como la convolución o la recurrencia. A este procedimiento lo llamamos **entrenamiento previo sin supervisión**, o más precisamente, **entrenamiento previo no supervisado por capas codicioso**. Este procedimiento es un ejemplo canónico de cómo una representación aprendida para una tarea (aprendizaje no supervisado, tratando de capturar la forma de la distribución de entrada) a veces puede ser útil para otra tarea (aprendizaje supervisado con el mismo dominio de entrada).

El preentrenamiento codicioso por capas sin supervisión se basa en un algoritmo de aprendizaje de representación de una sola capa, como un RBM, un codificador automático de una sola capa, un modelo de codificación dispersa u otro modelo que aprende representaciones latentes. Cada capa se entrena previamente mediante aprendizaje no supervisado, tomando la salida de la capa anterior y produciendo como salida una nueva representación de los datos, cuya distribución (o su relación con otras variables como las categorías a predecir) es, con suerte, más sencilla. Ver algoritmo 15.1 para una descripción formal.

Durante mucho tiempo se han utilizado procedimientos de entrenamiento codiciosos por capas basados en criterios no supervisados para eludir la dificultad de entrenar conjuntamente las capas de una red neuronal profunda para una tarea supervisada. Este enfoque se remonta al menos hasta el Neocognitron ([fukushima,1975](#)). El renacimiento del aprendizaje profundo de 2006 comenzó con el descubrimiento de que este procedimiento de aprendizaje voraz podría usarse para encontrar una buena inicialización para un procedimiento de aprendizaje conjunto en todas las capas, y que este enfoque podría usarse para entrenar con éxito incluso arquitecturas completamente conectadas ([Hinton et al.,2006;Hinton y Salakhutdinov,2006;Hinton,2006;bengio et al.,2007;Ranzato et al.,2007a](#)). Antes de este descubrimiento, solo las redes profundas convolucionales o las redes cuya profundidad resultaba de la recurrencia se consideraban factibles de entrenar. Hoy en día, sabemos que no se requiere un entrenamiento previo codicioso por capas para entrenar arquitecturas profundas completamente conectadas, pero el enfoque de entrenamiento previo no supervisado fue el primer método para tener éxito.

El preentrenamiento codicioso por capas se llama **avarío** porque es un **algoritmo codicioso**

ritmo, lo que significa que optimiza cada pieza de la solución de forma independiente, una pieza a la vez, en lugar de optimizar todas las piezas de forma conjunta. Se llama **por capas** porque estas piezas independientes son las capas de la red. Específicamente, el preentrenamiento codicioso por capas procede una capa a la vez, entrenando el k -ésima capa manteniendo las anteriores fijas. En particular, las capas inferiores (que se entranan primero) no se adaptan después de introducir las capas superiores. Se llamasin **supervisión** porque cada capa se entrena con un algoritmo de aprendizaje de representación no supervisado. Sin embargo también se le llama **Pre-entrenamiento**, porque se supone que es solo un primer paso antes de que se aplique un algoritmo de entrenamiento conjunto **afisintonía** todas las capas juntas. En el contexto de una tarea de aprendizaje supervisado, puede verse como un regularizador (en algunos experimentos, el entrenamiento previo disminuye el error de prueba sin disminuir el error de entrenamiento) y una forma de inicialización de parámetros.

Es común utilizar la palabra “preentrenamiento” para referirse no solo a la etapa de preentrenamiento en sí, sino a todo el protocolo de dos fases que combina la fase de preentrenamiento y una fase de aprendizaje supervisado. La fase de aprendizaje supervisado puede implicar el entrenamiento de un clasificador simple además de las funciones aprendidas en la fase de preentrenamiento, o puede implicar el ajuste fino supervisado de toda la red aprendida en la fase de preentrenamiento. No importa qué tipo de algoritmo de aprendizaje no supervisado o qué tipo de modelo se emplee, en la gran mayoría de los casos, el esquema general de entrenamiento es casi el mismo. Si bien la elección del algoritmo de aprendizaje no supervisado obviamente afectará los detalles, la mayoría de las aplicaciones de preentrenamiento no supervisado siguen este protocolo básico.

El preentrenamiento no supervisado por capas codicioso también se puede utilizar como inicialización para otros algoritmos de aprendizaje no supervisados, como los codificadores automáticos profundos ([Hinton y Salakhutdinov, 2006](#)) y modelos probabilísticos con muchas capas de variables latentes. Tales modelos incluyen redes de creencias profundas ([Hinton et al., 2006](#)) y máquinas profundas de Boltzmann ([Salakhutdinov y Hinton, 2009a](#)). Estos modelos generativos profundos se describirán en el capítulo [20](#).

Como se discutió en la sección [8.7.4](#), también es posible tener codiciosos por capas *supervisado* Pre-entrenamiento. Esto se basa en la premisa de que entrenar una red superficial es más fácil que entrenar una profunda, lo que parece haber sido validado en varios contextos ([Erhan et al., 2010](#)).

15.1.1 ¿Cuándo y por qué funciona el preentrenamiento no supervisado?

En muchas tareas, el preentrenamiento no supervisado por capas codicioso puede producir mejoras sustanciales en el error de prueba para las tareas de clasificación. Esta observación fue responsable del renovado interés en las redes neuronales profundas a partir de 2006 ([Hinton et al.,](#),

Algoritmo 15.1 Protocolo de preentrenamiento no supervisado por capas Greedy.

Dado lo siguiente: Algoritmo de aprendizaje de funciones no supervisado L , que toma un conjunto de ejemplos de entrenamiento y devuelve un codificador o función de característica F . Los datos de entrada sin procesar son X , con una fila por ejemplo y $F_1(X)$ es la salida del codificador de primera etapa en X . En el caso de que se realice un ajuste fino, utilizamos un módulo de aprendizaje T que toma una función inicial F , ejemplos de entrada X y en el caso de ajuste fino supervisado, objetivos asociados Y , y devuelve una función ajustada. El número de etapas es m .

```
 $F \leftarrow$  función de  
identidad  $X = X$   
para  $k = 1, \dots, m$  hacer  
     $F_k = L(X)$   
     $F \leftarrow F_k \circ F$   
     $X \leftarrow F_k(X)$   
fin para  
si  $f$  es intonación entonces  
     $F \leftarrow T(F, X, Y)$   
terminara si  
Devolver  $F$ 
```

2006; Bengio *et al.*, 2007; Ranzato *et al.*, 2007a). Sin embargo, en muchas otras tareas, el entrenamiento previo no supervisado no confiere ningún beneficio o incluso causa un daño notable. Mamá *et al.* (2015) estudiaron el efecto del entrenamiento previo en modelos de aprendizaje automático para la predicción de la actividad química y descubrieron que, en promedio, el entrenamiento previo era levemente dañino, pero para muchas tareas fue significativamente útil. Debido a que la capacitación previa sin supervisión a veces es útil pero a menudo dañina, es importante comprender cuándo y por qué funciona para determinar si es aplicable a una tarea en particular.

Desde el principio, es importante aclarar que la mayor parte de esta discusión se limita en particular al preentrenamiento codicioso no supervisado. Existen otros paradigmas completamente diferentes para realizar el aprendizaje semisupervisado con redes neuronales, como el entrenamiento de confrontación virtual descrito en la sección 7.13. También es posible entrenar un autocodificador o un modelo generativo al mismo tiempo que el modelo supervisado. Los ejemplos de este enfoque de una sola etapa incluyen la RBM discriminativa (Larochelle y Bengio, 2008) y la red de escalera (Rasmussen *et al.*, 2015), en el que el objetivo total es una suma explícita de los dos términos (uno usando las etiquetas y otro usando solo la entrada).

El preentrenamiento no supervisado combina dos ideas diferentes. En primer lugar, hace uso de

la idea de que la elección de parámetros iniciales para una red neuronal profunda puede tener un efecto regularizador significativo en el modelo (y, en menor medida, que puede mejorar la optimización). En segundo lugar, hace uso de la idea más general de que aprender sobre la distribución de entradas puede ayudar a aprender sobre el mapeo de entradas a salidas.

Ambas ideas involucran muchas interacciones complicadas entre varias partes del algoritmo de aprendizaje automático que no se entienden por completo.

La primera idea, que la elección de los parámetros iniciales para una red neuronal profunda puede tener un fuerte efecto de regularización en su desempeño, es la menos comprendida. En el momento en que el preentrenamiento se hizo popular, se entendía como inicializar el modelo en una ubicación que haría que se acercara a un mínimo local en lugar de a otro. Hoy en día, los mínimos locales ya no se consideran un problema grave para la optimización de redes neuronales. Ahora sabemos que nuestros procedimientos estándar de entrenamiento de redes neuronales generalmente no llegan a un punto crítico de ningún tipo. Sigue siendo posible que el entrenamiento previo inicialice el modelo en una ubicación que de otro modo sería inaccesible, por ejemplo, una región que está rodeada de áreas donde la función de costo varía tanto de un ejemplo a otro que los minilotes solo brindan una estimación muy ruidosa del gradiente. , o una región rodeada de áreas donde la matriz Hessiana está tan mal acondicionada que los métodos de descenso de gradiente deben usar pasos muy pequeños. Sin embargo, nuestra capacidad para caracterizar exactamente qué aspectos de los parámetros preentrenados se retienen durante la etapa de entrenamiento supervisado es limitada. Esta es una de las razones por las que los enfoques modernos suelen utilizar aprendizaje supervisado y no supervisado simultáneos en lugar de dos etapas secuenciales. También se puede evitar luchar con estas ideas complicadas sobre cómo la optimización en la etapa de aprendizaje supervisado conserva la información de la etapa de aprendizaje no supervisado simplemente congelando los parámetros para los extractores de características y usando el aprendizaje supervisado solo para agregar un clasificador sobre las características aprendidas.

La otra idea, que un algoritmo de aprendizaje puede usar la información aprendida en la fase no supervisada para funcionar mejor en la etapa de aprendizaje supervisado, se comprende mejor. La idea básica es que algunas características que son útiles para la tarea no supervisada también pueden ser útiles para la tarea de aprendizaje supervisado. Por ejemplo, si entrenamos un modelo generativo de imágenes de automóviles y motocicletas, necesitará saber sobre ruedas y sobre cuántas ruedas debe haber en una imagen. Si tenemos suerte, la representación de las ruedas adoptará una forma de fácil acceso para el alumno supervisado. Esto aún no se entiende a nivel teórico matemático, por lo que no siempre es posible predecir qué tareas se beneficiarán del aprendizaje no supervisado de esta manera. Muchos aspectos de este enfoque dependen en gran medida de los modelos específicos utilizados. Por ejemplo,

Además de las funciones preentrenadas, las funciones deben hacer que las clases subyacentes sean linealmente separables. Estas propiedades a menudo ocurren naturalmente, pero no siempre lo hacen. Esta es otra razón por la que puede ser preferible el aprendizaje simultáneo supervisado y no supervisado: las restricciones impuestas por la capa de salida se incluyen naturalmente desde el principio.

Desde el punto de vista del preentrenamiento no supervisado como aprendizaje de una representación, podemos esperar que el preentrenamiento no supervisado sea más efectivo cuando la representación inicial es pobre. Un ejemplo clave de esto es el uso de incrustaciones de palabras. Las palabras representadas por vectores one-hot no son muy informativas porque cada dos vectores one-hot distintos están a la misma distancia entre sí (al cuadrado L_2 distancia de 2). Las incrustaciones de palabras aprendidas codifican naturalmente la similitud entre las palabras por su distancia entre sí. Debido a esto, el entrenamiento previo sin supervisión es especialmente útil cuando se procesan palabras. Es menos útil cuando se procesan imágenes, quizás porque las imágenes ya se encuentran en un rico espacio vectorial donde las distancias proporcionan una métrica de similitud de baja calidad.

Desde el punto de vista del entrenamiento previo no supervisado como regularizador, podemos esperar que el entrenamiento previo no supervisado sea más útil cuando el número de ejemplos etiquetados es muy pequeño. Debido a que la fuente de información agregada por el entrenamiento previo no supervisado son los datos no etiquetados, también podemos esperar que el entrenamiento previo no supervisado funcione mejor cuando la cantidad de ejemplos no etiquetados es muy grande. La ventaja del aprendizaje semisupervisado a través de un preentrenamiento no supervisado con muchos ejemplos no etiquetados y pocos ejemplos etiquetados se hizo particularmente clara en 2011 cuando el preentrenamiento no supervisado ganó dos competencias internacionales de transferencia de aprendizaje ([Mesnil et al., 2011](#); [Buen compañero et al., 2011](#)), en entornos donde la cantidad de ejemplos etiquetados en la tarea de destino era pequeña (desde un puñado hasta docenas de ejemplos por clase). Estos efectos también fueron documentados en experimentos cuidadosamente controlados por [doloret al.\(2014\)](#).

Es probable que intervengan otros factores. Por ejemplo, es probable que el entrenamiento previo sin supervisión sea más útil cuando la función que se va a aprender es extremadamente complicada. El aprendizaje no supervisado difiere de los regularizadores como la disminución del peso porque no sesga al alumno hacia el descubrimiento de una función simple, sino hacia el descubrimiento de funciones características que son útiles para la tarea de aprendizaje no supervisado. Si las verdaderas funciones subyacentes son complicadas y están moldeadas por las regularidades de la distribución de entrada, el aprendizaje no supervisado puede ser un regularizador más apropiado.

Dejando a un lado estas advertencias, ahora analizamos algunos casos de éxito en los que se sabe que el entrenamiento previo sin supervisión causa una mejora, y explicamos lo que se sabe sobre por qué ocurre esta mejora. El preentrenamiento no supervisado generalmente se ha utilizado para mejorar los clasificadores y, por lo general, es más interesante desde el punto de vista de la clasificación.

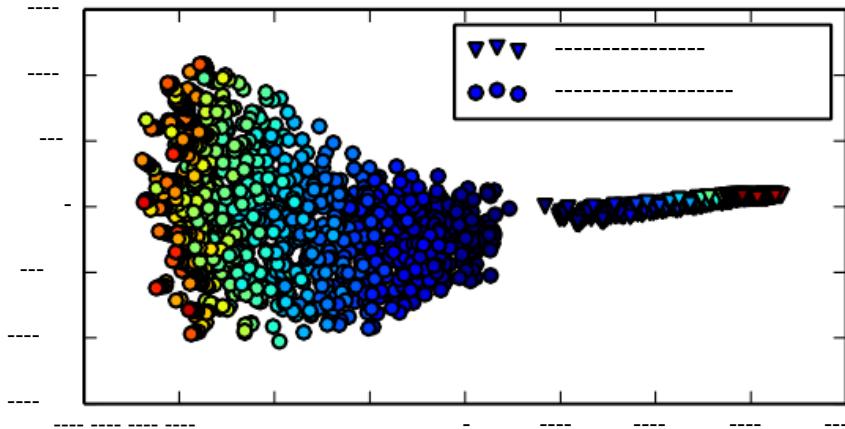


Figura 15.1: Visualización mediante proyección no lineal de las trayectorias de aprendizaje de diferentes redes neuronales en espacio funcional (no espacio de parámetros, para evitar el problema de las asignaciones de muchos a uno de los vectores de parámetros a las funciones), con diferentes inicializaciones aleatorias y con o sin entrenamiento previo no supervisado. Cada punto corresponde a una red neuronal diferente, en un momento particular de su proceso de entrenamiento. Esta figura está adaptada con permiso de [Erhan et al. \(2010\)](#). Una coordenada en el espacio de funciones es un vector de dimensión infinita que asocia cada entrada x con una salida y . [Erhan et al. \(2010\)](#) hizo una proyección lineal al espacio de alta dimensión al concatenar el y para muchos específicos x puntos. Luego hicieron una proyección no lineal adicional a 2-D por Isomap ([Tenenbaum et al., 2000](#)). El color indica el tiempo. Todas las redes se inicializan cerca del centro de la gráfica (correspondiente a la región de funciones que produce distribuciones aproximadamente uniformes sobre la clase y para la mayoría de las entradas). Con el tiempo, el aprendizaje mueve la función hacia afuera, hacia puntos que hacen predicciones sólidas. El entrenamiento termina constantemente en una región cuando se usa el entrenamiento previo y en otra región que no se superpone cuando no se usa el entrenamiento previo. Isomap intenta preservar las distancias relativas globales (y, por lo tanto, los volúmenes), de modo que la pequeña región correspondiente a los modelos preentrenados pueda indicar que el estimador basado en el preentrenamiento tiene una varianza reducida.

reduciendo el error del conjunto de prueba. Sin embargo, el entrenamiento previo no supervisado puede ayudar en otras tareas además de la clasificación y puede actuar para mejorar la optimización en lugar de ser simplemente un regularizador. Por ejemplo, puede mejorar el error de reconstrucción de prueba y tren para codificadores automáticos profundos ([Hinton y Salakhutdinov, 2006](#)).

[Erhan et al. \(2010\)](#) realizaron muchos experimentos para explicar varios éxitos del preentrenamiento no supervisado. Tanto las mejoras en el error de entrenamiento como las mejoras en el error de prueba pueden explicarse en términos de entrenamiento previo no supervisado que lleva los parámetros a una región que de otro modo sería inaccesible. El entrenamiento de redes neuronales no es determinista y converge a una función diferente cada vez que se ejecuta. El entrenamiento puede detenerse en un punto donde la pendiente se vuelve pequeña, un punto en el que detenerse antes de tiempo termina el entrenamiento para evitar el sobreajuste, o en un punto donde la pendiente es grande pero es difícil encontrar un escalón cuesta abajo debido a problemas como la estocasticidad o el mal acondicionamiento. Las redes neuronales que reciben entrenamiento previo no supervisado se detienen constantemente en la misma región del espacio funcional, mientras que las redes neuronales sin entrenamiento previo se detienen constantemente en otra región. Ver figura 15.1 para una visualización de este fenómeno. La región donde llegan las redes preentrenadas es más pequeña, lo que sugiere que el preentrenamiento reduce la varianza del proceso de estimación, lo que a su vez puede reducir el riesgo de un sobreajuste severo. En otras palabras, el entrenamiento previo no supervisado inicializa los parámetros de la red neuronal en una región de la que no escapan, y los resultados que siguen a esta inicialización son más consistentes y es menos probable que sean muy malos que sin esta inicialización.

[Erhan et al. \(2010\)](#) también proporcionan algunas respuestas en cuanto a cuándo el entrenamiento previo funciona mejor: la media y la varianza del error de prueba se redujeron más con el entrenamiento previo para redes más profundas. Tenga en cuenta que estos experimentos se realizaron antes de la invención y popularización de las técnicas modernas para entrenar redes muy profundas (unidades lineales rectificadas, abandono y normalización por lotes), por lo que se sabe menos sobre el efecto del entrenamiento previo no supervisado junto con los enfoques contemporáneos.

Una pregunta importante es cómo el preentrenamiento no supervisado puede actuar como regularizador. Una hipótesis es que el entrenamiento previo alienta al algoritmo de aprendizaje a descubrir características que se relacionan con las causas subyacentes que generan los datos observados. Esta es una idea importante que motiva muchos otros algoritmos además del preentrenamiento no supervisado, y se describe más adelante en la sección 15.3.

Comparado con otras formas de aprendizaje no supervisado, el preentrenamiento no supervisado tiene la desventaja de que opera con dos fases de entrenamiento separadas. Muchas estrategias de regularización tienen la ventaja de permitir al usuario controlar la intensidad de la regularización ajustando el valor de un solo hiperparámetro. El preentrenamiento no supervisado no ofrece una forma clara de ajustar la fuerza de la regularización derivada de la etapa no supervisada. En cambio, hay

muchos hiperparámetros, cuyo efecto puede medirse después del hecho, pero a menudo es difícil de predecir con anticipación. Cuando realizamos aprendizaje no supervisado y supervisado simultáneamente, en lugar de utilizar la estrategia de preentrenamiento, hay un solo hiperparámetro, generalmente un coeficiente adjunto al costo no supervisado, que determina con qué fuerza el objetivo no supervisado regularizará el modelo supervisado. Uno siempre puede predeciblemente obtener menos regularización al disminuir este coeficiente. En el caso del preentrenamiento no supervisado, no hay forma de adaptar de manera flexible la fuerza de la regularización: el modelo supervisado se inicializa con los parámetros preentrenados o no.

Otra desventaja de tener dos fases de entrenamiento separadas es que cada fase tiene sus propios hiperparámetros. El rendimiento de la segunda fase generalmente no se puede predecir durante la primera fase, por lo que hay una gran demora entre proponer hiperparámetros para la primera fase y poder actualizarlos utilizando los comentarios de la segunda fase. El enfoque más basado en principios es utilizar el error de conjunto de validación en la fase supervisada para seleccionar los hiperparámetros de la fase de preentrenamiento, como se explica en [Larochelle et al. \(2009\)](#). En la práctica, algunos hiperparámetros, como el número de iteraciones de preentrenamiento, se establecen de manera más conveniente durante la fase de preentrenamiento, utilizando la detención anticipada en el objetivo no supervisado, lo cual no es ideal, pero computacionalmente es mucho más económico que usar el objetivo supervisado.

Hoy en día, el preentrenamiento no supervisado se ha abandonado en gran medida, excepto en el campo del procesamiento del lenguaje natural, donde la representación natural de las palabras como vectores one-hot no transmite información de similitud y donde hay conjuntos muy grandes sin etiquetar disponibles. En ese caso, la ventaja del preentrenamiento es que uno puede preentrenar una vez en un conjunto enorme sin etiquetar (por ejemplo, con un corpus que contiene miles de millones de palabras), aprender una buena representación (típicamente de palabras, pero también de oraciones) y luego usar esta representación o ajustarlo para una tarea supervisada para la cual el conjunto de entrenamiento contiene muchos menos ejemplos. Este enfoque fue iniciado por [Collobert y Weston \(2008b\)](#), [turiano et al. \(2010\)](#), y [coloberto et al. \(2011a\)](#) y sigue siendo de uso común en la actualidad.

Las técnicas de aprendizaje profundo basadas en el aprendizaje supervisado, regularizado con abandono o normalización por lotes, pueden lograr un rendimiento a nivel humano en muchas tareas, pero solo con conjuntos de datos etiquetados extremadamente grandes. Estas mismas técnicas superan el entrenamiento previo no supervisado en conjuntos de datos de tamaño mediano como CIFAR-10 y MNIST, que tienen aproximadamente 5000 ejemplos etiquetados por clase. En conjuntos de datos extremadamente pequeños, como el conjunto de datos de empalme alternativo, los métodos bayesianos superan a los métodos basados en entrenamiento previo no supervisado ([Srivastava, 2013](#)). Por estas razones, la popularidad del preentrenamiento no supervisado ha disminuido. Sin embargo, el preentrenamiento no supervisado sigue siendo un hito importante en la historia de la investigación del aprendizaje profundo.

y sigue influyendo en los enfoques contemporáneos. La idea de preentrenamiento se ha generalizado a **entrenamiento previo supervisado** discutido en la sección 8.7.4, como un enfoque muy común para el aprendizaje por transferencia. El preentrenamiento supervisado para el aprendizaje de transferencia es popular (Oquab et al., 2014; Yosinski et al., 2014) para usar con redes convolucionales previamente entrenadas en el conjunto de datos de ImageNet. Los profesionales publican los parámetros de estas redes entrenadas para este propósito, al igual que los vectores de palabras preentrenadas se publican para tareas de lenguaje natural (coloberto et al., 2011a; mikolov et al., 2013a).

15.2 Transferencia de aprendizaje y adaptación del dominio

El aprendizaje de transferencia y la adaptación del dominio se refieren a la situación en la que lo que se ha aprendido en un entorno (es decir, distribución PAG_1) se explota para mejorar la generalización en otro entorno (por ejemplo, distribución PAG_2). Esto generaliza la idea presentada en la sección anterior, donde transferimos representaciones entre una tarea de aprendizaje no supervisado y una tarea de aprendizaje supervisado.

En **transferir el aprendizaje**, el alumno debe realizar dos o más tareas diferentes, pero suponemos que muchos de los factores que explican las variaciones en PAG_1 son relevantes para las variaciones que necesitan ser capturadas para el aprendizaje PAG_2 . Esto generalmente se entiende en un contexto de aprendizaje supervisado, donde la entrada es la misma pero el objetivo puede ser de naturaleza diferente. Por ejemplo, podemos aprender sobre un conjunto de categorías visuales, como gatos y perros, en el primer escenario, y luego aprender sobre un conjunto diferente de categorías visuales, como hormigas y avispas, en el segundo escenario. Si hay significativamente más datos en la primera configuración (muestreados de PAG_1), entonces eso puede ayudar a aprender representaciones que son útiles para generalizar rápidamente a partir de muy pocos ejemplos extraídos de PAG_2 . Muchas categorías visuales *comparten* movimientos de bajo nivel de bordes y formas visuales, los efectos de los cambios geométricos, cambios en la iluminación, etc. En general, el aprendizaje por transferencia, el aprendizaje multitarea (sección 7.7), y la adaptación del dominio se puede lograr a través del aprendizaje de representación cuando existen características que son útiles para los diferentes escenarios o tareas, correspondientes a factores subyacentes que aparecen en más de un escenario. Esto se ilustra en la figura 7.2, con capas inferiores compartidas y capas superiores dependientes de tareas.

Sin embargo, a veces, lo que se comparte entre las diferentes tareas no es la semántica de la entrada sino la semántica de la salida. Por ejemplo, un sistema de reconocimiento de voz necesita producir oraciones válidas en la capa de salida, pero las capas anteriores cerca de la entrada pueden necesitar reconocer versiones muy diferentes de los mismos fonemas o vocalizaciones subfonémicas según la persona que esté hablando. En casos como estos, tiene más sentido compartir las capas superiores (cerca de la salida) de la red neuronal y tener un preprocesamiento específico de la tarea, como

ilustrado en la figura 15.2.

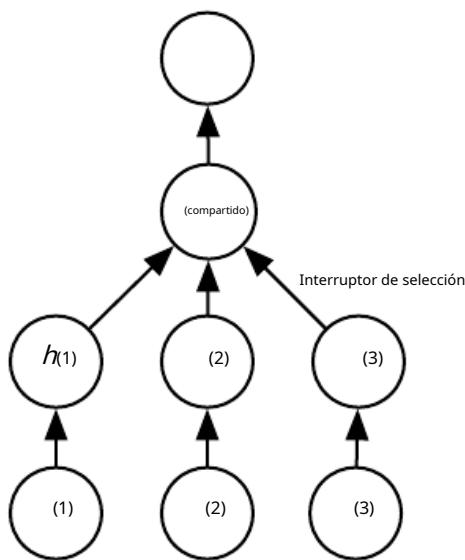


Figura 15.2: Ejemplo de arquitectura para tareas múltiples o transferencia de aprendizaje cuando la variable de salida y tiene la misma semántica para todas las tareas mientras que la variable de entrada X tiene un significado diferente (y posiblemente incluso una dimensión diferente) para cada tarea (o, por ejemplo, cada usuario), llamado $X_{(1)}, X_{(2)} \text{ y } X_{(3)}$ para tres tareas. Los niveles inferiores (hasta el interruptor de selección) son específicos de la tarea, mientras que los niveles superiores son compartidos. Los niveles inferiores aprenden a traducir sus entradas específicas de tareas en un conjunto genérico de funciones.

En el caso relacionado de **adaptación de dominio**, la tarea (y la asignación óptima de entrada a salida) sigue siendo la misma entre cada configuración, pero la distribución de entrada es ligeramente diferente. Por ejemplo, considere la tarea de análisis de sentimiento, que consiste en determinar si un comentario expresa un sentimiento positivo o negativo. Los comentarios publicados en la web provienen de muchas categorías. Puede surgir un escenario de adaptación de dominio cuando un predictor de sentimientos entrenado en las reseñas de los clientes sobre contenido multimedia, como libros, videos y música, se usa más tarde para analizar comentarios sobre productos electrónicos de consumo, como televisores o teléfonos inteligentes. Uno puede imaginar que existe una función subyacente que indica si una declaración es positiva, neutra o negativa, pero, por supuesto, el vocabulario y el estilo pueden variar de un dominio a otro, lo que dificulta la generalización entre dominios.[gloria et al., 2011b](#).

Un problema relacionado es el de **deriva del concepto**, que podemos ver como una forma de transferencia de aprendizaje debido a cambios graduales en la distribución de datos a lo largo del tiempo. Tanto la deriva de conceptos como el aprendizaje por transferencia pueden verse como formas particulares de

aprendizaje multitarea. Si bien la frase "aprendizaje multitarea" generalmente se refiere a tareas de aprendizaje supervisado, la noción más general de aprendizaje por transferencia también se aplica al aprendizaje no supervisado y al aprendizaje por refuerzo.

En todos estos casos, el objetivo es aprovechar los datos del primer escenario para extraer información que pueda ser útil a la hora de aprender o incluso directamente hacer predicciones en el segundo escenario. La idea central del aprendizaje de representaciones es que la misma representación puede ser útil en ambos entornos. El uso de la misma representación en ambas configuraciones permite que la representación se beneficie de los datos de entrenamiento que están disponibles para ambas tareas.

Como se mencionó anteriormente, el aprendizaje profundo no supervisado para el aprendizaje por transferencia ha tenido éxito en algunas competencias de aprendizaje automático ([Mesnil et al., 2011](#); [Buen compañero et al., 2011](#)). En la primera de estas competiciones, el montaje experimental es el siguiente. Cada participante recibe primero un conjunto de datos de la primera configuración (de distribución PAG_1), que ilustran ejemplos de algún conjunto de categorías. Los participantes deben usar esto para aprender un buen espacio de características (asignar la entrada sin procesar a alguna representación), de modo que cuando aplicamos esta transformación aprendida a las entradas del entorno de transferencia (distribución PAG_2), un clasificador lineal se puede entrenar y generalizar bien a partir de muy pocos ejemplos etiquetados. Uno de los resultados más llamativos encontrados en esta competencia es que, como arquitectura, hace uso de representaciones cada vez más profundas (aprendidas de forma puramente no supervisada a partir de los datos recopilados en la primera configuración, PAG_1), la curva de aprendizaje en las nuevas categorías de la segunda configuración (transferencia) PAG_2 se vuelve mucho mejor. Para representaciones profundas, se necesitan menos ejemplos etiquetados de las tareas de transferencia para lograr el rendimiento de generalización aparentemente asintótico.

Dos formas extremas de transferencia de aprendizaje son **aprendizaje de una sola vez** y **aprendizaje de tiro cero**, a veces también llamado **aprendizaje de datos cero**. Solo se da un ejemplo etiquetado de la tarea de transferencia para el aprendizaje de un solo intento, mientras que no se dan ejemplos etiquetados en absoluto para la tarea de aprendizaje de cero intentos.

Aprendizaje de una sola vez ([fei-fei et al., 2006](#)) es posible porque la representación aprende a separar limpiamente las clases subyacentes durante la primera etapa. Durante la etapa de transferencia de aprendizaje, solo se necesita un ejemplo etiquetado para inferir la etiqueta de muchos ejemplos de prueba posibles que se agrupan alrededor del mismo punto en el espacio de representación. Esto funciona en la medida en que los factores de variación correspondientes a estas invariancias se han separado claramente de los otros factores, en el espacio de representación aprendido, y de alguna manera hemos aprendido qué factores importan y qué no importan al discriminar objetos de ciertas categorías.

Como ejemplo de un entorno de aprendizaje de tiro cero, considere el problema de hacer que un alumno lea una gran colección de texto y luego resuelva problemas de reconocimiento de objetos.

Puede ser posible reconocer una clase de objeto específica incluso sin haber visto una imagen de ese objeto, si el texto describe el objeto lo suficientemente bien. Por ejemplo, después de haber leído que un gato tiene cuatro patas y orejas puntiagudas, el alumno podría adivinar que una imagen es un gato, sin haber visto un gato antes.

Aprendizaje de datos cero ([Larochelle et al., 2008](#)) y aprendizaje de tiro cero ([Palatucci et al., 2009; Socher et al., 2013b](#)) solo son posibles porque se ha explotado información adicional durante el entrenamiento. Podemos pensar en el escenario de aprendizaje de datos cero como si incluyera tres variables aleatorias: las entradas tradicionales X , los productos u objetivos tradicionales y , y una variable aleatoria adicional que describe la tarea, T . El modelo está entrenado para estimar la distribución condicional $p_{\text{ad}}(y | X, T)$ donde T es una descripción de la tarea que deseamos que realice el modelo. En nuestro ejemplo de reconocimiento de gatos después de haber leído sobre gatos, la salida es una variable binaria y con $y=1$ indicando "sí" y $y=0$ indicando "no". Variable de la tarea T luego representa preguntas para ser respondidas como "¿Hay un gato en esta imagen?" Si tenemos un conjunto de entrenamiento que contiene ejemplos no supervisados de objetos que viven en el mismo espacio que T , podemos ser capaces de inferir el significado de instancias invisibles de T . En nuestro ejemplo de reconocer gatos sin haber visto una imagen del gato, es importante que hayamos tenido datos de texto sin etiquetar que contengan oraciones como "los gatos tienen cuatro patas" o "los gatos tienen orejas puntiagudas".

El aprendizaje de tiro cero requiere T ser representado de una manera que permita algún tipo de generalización. Por ejemplo, T no puede ser simplemente un código único que indica una categoría de objeto. [Socher et al. \(2013b\)](#) proporcionan en cambio una representación distribuida de categorías de objetos mediante el uso de una palabra aprendida incrustada para la palabra asociada con cada categoría.

Un fenómeno similar ocurre en la traducción automática ([Kleméntievet al., 2012; mikolov et al., 2013b; gouws et al., 2014](#)): tenemos palabras en un idioma, y las relaciones entre palabras se pueden aprender de corpus monolingües; por otro lado, tenemos oraciones traducidas que relacionan palabras en un idioma con palabras en el otro. Aunque es posible que no hayamos etiquetado ejemplos que traduzcan palabras A en idioma X a la palabra B en idioma Y , podemos generalizar y adivinar una traducción para word A porque hemos aprendido una representación distribuida de las palabras en el lenguaje X , una representación distribuida de palabras en el lenguaje Y , y creó un enlace (posiblemente bidireccional) que relaciona los dos espacios, a través de ejemplos de entrenamiento que consisten en pares de oraciones emparejadas en ambos idiomas. Esta transferencia será más exitosa si los tres ingredientes (las dos representaciones y las relaciones entre ellas) se aprenden conjuntamente.

El aprendizaje de disparo cero es una forma particular de aprendizaje por transferencia. El mismo principio explica cómo se puede realizar **aprendizaje multimodal**, capturando una representación

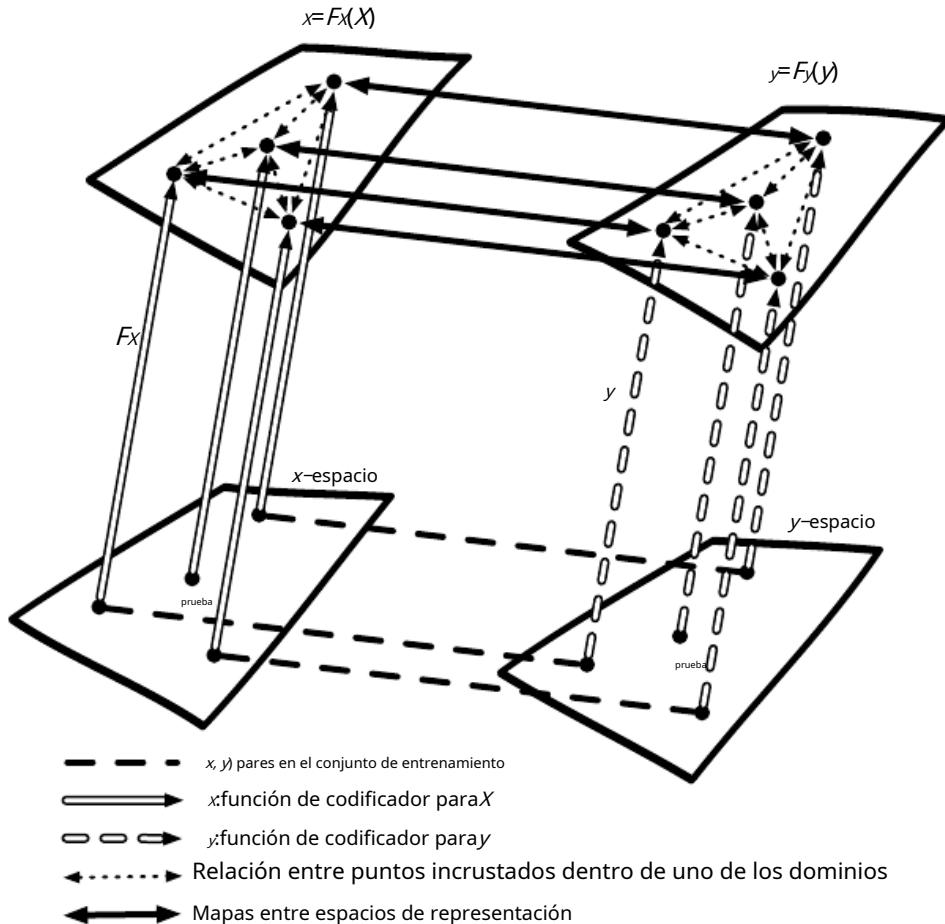


Figura 15.3: Transferencia de aprendizaje entre dos dominios X y Y permite el aprendizaje de tiro cero. Ejemplos etiquetados o no etiquetados de X permiten que uno aprenda una función de representación F_X y de manera similar con ejemplos de Y aprender F_Y . Cada aplicación de la F_X y F_Y funciones aparece como una flecha hacia arriba, con el estilo de las flechas indicando qué función se aplica. Distancia en X espacio proporciona una métrica de similitud entre cualquier par de puntos en X espacio que puede ser más significativo que la distancia en X espacio. Asimismo, la distancia en Y espacio proporciona una métrica de similitud entre cualquier par de puntos en Y espacio. Ambas funciones de similitud se indican con flechas bidireccionales punteadas. Los ejemplos etiquetados (líneas horizontales discontinuas) son pares (x, y) que permiten aprender un mapa unidireccional o bidireccional (flecha sólida bidireccional) entre las representaciones $F_X(X)$ y las representaciones $F_Y(Y)$ y anclar estas representaciones entre sí. El aprendizaje de datos cero se habilita de la siguiente manera. Se puede asociar una imagen X_{prueba} a una palabra Y_{prueba} , incluso si nunca se presentó ninguna imagen de esa palabra, simplemente porque las representaciones de palabras $F_Y(Y_{\text{prueba}})$ y representaciones de imágenes $F_X(X_{\text{prueba}})$ pueden relacionarse entre sí a través de los mapas entre espacios de representación. Funciona porque, aunque esa imagen y esa palabra nunca se emparejaron, sus respectivos vectores de características $F_X(X_{\text{prueba}})$ y $F_Y(Y_{\text{prueba}})$ se han relacionado entre sí. Figura inspirada en la sugerencia de Hrant Khachatrian.

en una modalidad, una representación en la otra, y la relación (en general una distribución conjunta) entre pares (x, y) que consiste en una observación X en una modalidad y otra observación y en la otra modalidad (Srivastava y Salakhutdinov, 2012). Al aprender los tres conjuntos de parámetros (de X su representación, desde ya su representación y la relación entre las dos representaciones), los conceptos en una representación están anclados en la otra, y viceversa, lo que permite generalizar significativamente a nuevos pares. El procedimiento se ilustra en la figura 15.3.

15.3 Desenredo semisupervisado de factores causales

Una pregunta importante sobre el aprendizaje de representaciones es "¿qué hace que una representación sea mejor que otra?" Una hipótesis es que una representación ideal es aquella en la que las características dentro de la representación corresponden a las causas subyacentes de los datos observados, con características o direcciones separadas en el espacio de características correspondientes a diferentes causas, de modo que la representación separa las causas entre sí. Esta hipótesis motiva enfoques en los que primero buscamos una buena representación para $p_{\text{ag}}(X)$. Tal representación también puede ser una buena representación para computar $p_{\text{ag}}(y/X)$ si y es una de las causas más destacadas de X . Esta idea ha guiado una gran cantidad de investigaciones de aprendizaje profundo desde al menos la década de 1990 (Becker y Hinton, 1992; Hinton y Sejnowski, 1999), con más detalle. Para otros argumentos sobre cuándo el aprendizaje semisupervisado puede superar al aprendizaje supervisado puro, remitimos al lector a la sección 1.2 de Capilla et al. (2006).

En otros enfoques del aprendizaje de representaciones, a menudo nos hemos preocupado por una representación que es fácil de modelar, por ejemplo, una cuya entradas son escasas o independientes entre sí. Una representación que separa claramente los factores causales subyacentes puede no ser necesariamente fácil de modelar. Sin embargo, otra parte de la hipótesis que motiva el aprendizaje semisupervisado a través del aprendizaje de representación no supervisado es que, para muchas tareas de IA, estas dos propiedades coinciden: una vez que podemos obtener las explicaciones subyacentes de lo que observamos, generalmente se vuelve fácil aislar atributos de los demás. En concreto, si una representación h representa muchas de las causas subyacentes de lo observado X , y las salidas y se encuentran entre las causas más destacadas, entonces es fácil predecir y de h .

Primero, veamos cómo el aprendizaje semisupervisado puede fallar porque el aprendizaje no supervisado de $p_{\text{ag}}(X)$ no sirve para aprender $p_{\text{ag}}(y/X)$. Consideremos, por ejemplo, el caso en que $p_{\text{ag}}(X)$ se distribuye uniformemente y queremos aprender $R(X) = \text{MI}[y/X]$. Claramente, observar un conjunto de entrenamiento de X valores por sí solos no nos da información sobre $p_{\text{ag}}(y/X)$.

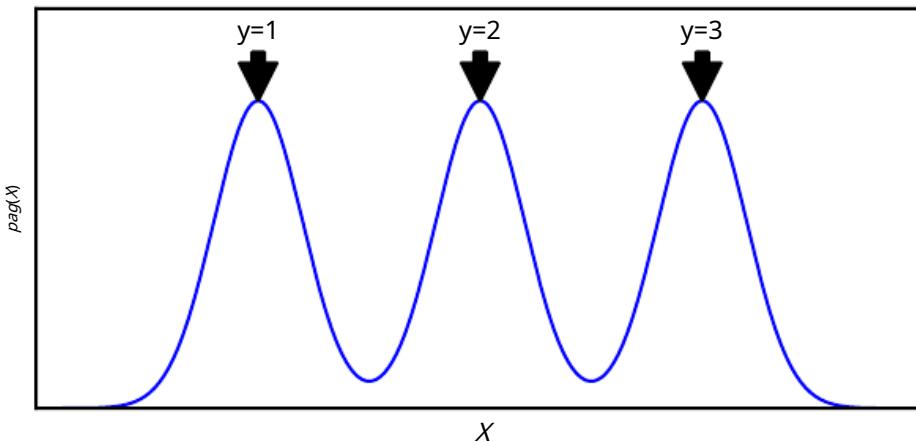


Figura 15.4: Ejemplo de una densidad sobre X que es una mezcla de tres componentes. La identidad del componente es un factor explicativo subyacente, y . Debido a que los componentes de la mezcla (p. ej., clases de objetos naturales en datos de imagen) son estadísticamente sobresalientes, solo modelar $\text{pag}(X)$ de una manera no supervisada sin ejemplo etiquetado ya revela el factor y .

A continuación, veamos un ejemplo sencillo de cómo puede tener éxito el aprendizaje semisupervisado. Considere la situación en la que surge de una mezcla, con un componente de mezcla por valor de y , como se ilustra en la figura 15.4. Si los componentes de la mezcla están bien separados, entonces el modelo $\text{pag}(X)$ revela con precisión dónde está cada componente, y un solo ejemplo etiquetado de cada clase será suficiente para aprender perfectamente $\text{pag}(y/X)$. Pero de manera más general, ¿qué podría hacer $\text{pag}(y/X)$ y $\text{pag}(X)$ estar atados juntos?

Si está íntimamente relacionado con uno de los factores causales de X , entonces $\text{pag}(X)$ y $\text{pag}(y/X)$ estará fuertemente vinculado, y es probable que el aprendizaje de representación no supervisado que trata de desentrañar los factores subyacentes de variación sea útil como estrategia de aprendizaje semisupervisado.

Considere la suposición de que y es uno de los factores causales de X , y deje h representar todos esos factores. El verdadero proceso generativo puede concebirse como estructurado de acuerdo con este modelo gráfico dirigido, con h como padre de X :

$$\text{pag}(h, X) = \text{pag}(X/h)\text{pag}(h). \quad (15.1)$$

Como consecuencia, los datos tienen probabilidad marginal

$$\text{pag}(X) = \min_h \text{pag}(x/h). \quad (15.2)$$

De esta simple observación, concluimos que el mejor modelo posible de X (desde un punto de vista de generalización) es el que descubre el "verdadero" anterior

estructura, con h como una variable latente que explica las variaciones observadas en X . El aprendizaje de representación "ideal" discutido anteriormente debería recuperar estos factores latentes. Si es uno de estos (o está estrechamente relacionado con uno de ellos), entonces será muy fácil aprender a predecir y de X basándose en la representación. También vemos que la distribución condicional de y dado X está vinculado por la regla de Bayes a los componentes de la ecuación anterior:

$$p_{\text{ag}}(y/x) = \frac{p_{\text{ag}}(X/y)p_{\text{ag}}(y)}{p_{\text{ag}}(X)}. \quad (15.3)$$

Así los marginales $p_{\text{ag}}(X)$ están íntimamente ligados al condicional $p_{\text{ag}}(y/X)$ y el conocimiento de la estructura del primero debería ser útil para aprender el segundo. Por lo tanto, en situaciones que respetan estos supuestos, el aprendizaje semisupervisado debería mejorar el rendimiento.

Un problema de investigación importante se refiere al hecho de que la mayoría de las observaciones están formadas por un número extremadamente grande de causas subyacentes. Supongamos que h_i , pero el alumno no supervisado no sabe qué h_i . La solución de fuerza bruta es que un alumno sin supervisión aprenda una representación que capture todos los factores generativos razonablemente destacados h_j y los desenreda unos de otros, lo que facilita la predicción de y independientemente de lo que h_i esté asociado con y .

En la práctica, la solución de fuerza bruta no es factible porque no es posible capturar todos o la mayoría de los factores de variación que influyen en una observación. Por ejemplo, en una escena visual, ¿la representación debería codificar siempre todos los objetos más pequeños del fondo? Es un fenómeno psicológico bien documentado que los seres humanos no perciben cambios en su entorno que no son inmediatamente relevantes para la tarea que están realizando; ver, por ejemplo, [Simons y Levin \(1998\)](#). Una importante frontera de investigación en el aprendizaje semisupervisado es determinar qué codificar en cada situación. Actualmente, dos de las principales estrategias para hacer frente a un gran número de causas subyacentes son utilizar una señal de aprendizaje supervisado al mismo tiempo que la señal de aprendizaje no supervisado para que el modelo opte por capturar los factores de variación más relevantes, o utilizar representaciones mucho más grandes si se usa aprendizaje puramente no supervisado.

Una estrategia emergente para el aprendizaje no supervisado es modificar la definición de qué causas subyacentes son las más destacadas. Históricamente, los codificadores automáticos y los modelos generativos se han entrenado para optimizar un criterio fijo, a menudo similar al error cuadrático medio. Estos criterios fijos determinan qué causas se consideran destacadas. Por ejemplo, el error cuadrático medio aplicado a los píxeles de una imagen especifica implícitamente que una causa subyacente solo es destacada si cambia significativamente el brillo de una gran cantidad de píxeles. Esto puede ser problemático si la tarea que deseamos resolver implica interactuar con objetos pequeños. Ver figura 15.5 para un ejemplo.

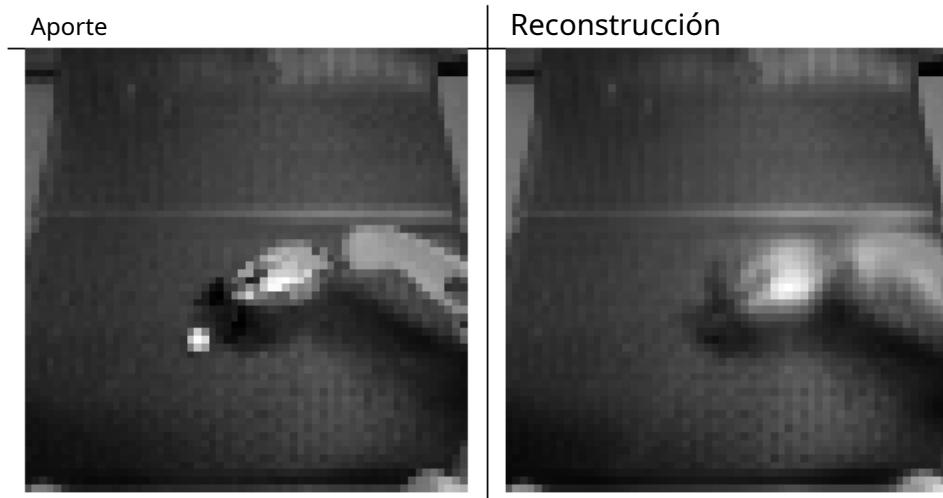


Figura 15.5: Un codificador automático entrenado con el error cuadrático medio para una tarea de robótica no pudo reconstruir una pelota de ping pong. La existencia de la pelota de ping pong y todas sus coordenadas espaciales son importantes factores causales subyacentes que generan la imagen y son relevantes para la tarea de robótica. Desafortunadamente, el codificador automático tiene una capacidad limitada y el entrenamiento con el error cuadrático medio no identificó la pelota de ping pong como lo suficientemente destacada para codificar. Imágenes amablemente proporcionadas por Chelsea Finn.

de una tarea de robótica en la que un codificador automático no ha podido aprender a codificar una pequeña pelota de ping pong. Este mismo robot es capaz de interactuar con éxito con objetos más grandes, como pelotas de béisbol, que son más sobresalientes según el error cuadrático medio.

Son posibles otras definiciones de prominencia. Por ejemplo, si un grupo de píxeles sigue un patrón altamente reconocible, incluso si ese patrón no implica un brillo u oscuridad extremos, ese patrón podría considerarse extremadamente destacado. Una forma de implementar tal definición de prominencia es utilizar un enfoque desarrollado recientemente llamado **redes adversarias generativas** ([Buen compañero et al., 2014c](#)). En este enfoque, se entrena un modelo generativo para engañar a un clasificador feedforward. El clasificador feedforward intenta reconocer todas las muestras del modelo generativo como falsas y todas las muestras del conjunto de entrenamiento como reales. En este marco, cualquier patrón estructurado que la red feedforward pueda reconocer es muy importante. La red antagónica generativa se describirá con más detalle en la sección [20.10.4](#). Para los propósitos de la presente discusión, es suficiente entender que ellos *aprender* cómo determinar lo que es sobresaliente. [lotero et al. \(2015\)](#) demostraron que los modelos entrenados para generar imágenes de cabezas humanas a menudo descuidan la generación de oídos cuando se entranan con el error cuadrático medio, pero generan con éxito los oídos cuando se entranan con el marco adversario. Debido a que las orejas no son extremadamente brillantes u oscuras en comparación con la piel circundante, no son especialmente sobresalientes de acuerdo con la pérdida de error cuadrático medio, pero su alto

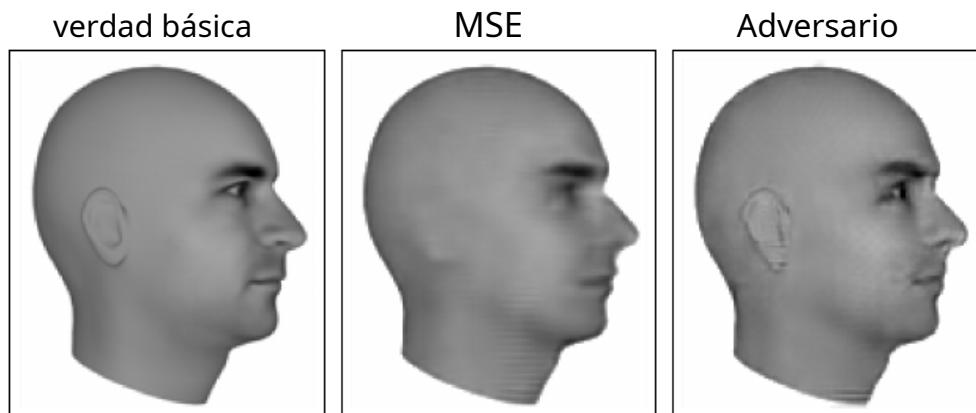


Figura 15.6: Las redes generativas predictivas brindan un ejemplo de la importancia de aprender qué características son las más destacadas. En este ejemplo, la red generativa predictiva ha sido entrenada para predecir la apariencia de un modelo tridimensional de una cabeza humana en un ángulo de visión específico. (Izquierda) Verdad fundamental. Esta es la imagen correcta, que la red debe emitir. (Centro) Imagen producida por una red generativa predictiva entrenada solo con el error cuadrático medio. Debido a que las orejas no causan una diferencia extrema en el brillo en comparación con la piel vecina, no eran lo suficientemente prominentes para que el modelo aprendiera a representarlas. (Derecha) Imagen producida por un modelo entrenado con una combinación de error cuadrático medio y pérdida adversaria. Usando esta función de costo aprendida, las orejas sobresalen porque siguen un patrón predecible. Aprender qué causas subyacentes son lo suficientemente importantes y relevantes para modelar es un área activa importante de investigación. Cifras proporcionadas gentilmente por [Iotero et al. \(2015\)](#).

la forma reconocible y la posición constante significa que una red de retroalimentación puede aprender fácilmente a detectarlos, lo que los hace muy destacados en el marco de confrontación generativa. Ver figura 15.6 por ejemplo imágenes. Las redes antagónicas generativas son solo un paso para determinar qué factores deben estar representados. Esperamos que la investigación futura descubra mejores formas de determinar qué factores representar y desarrollar mecanismos para representar diferentes factores según la tarea.

Un beneficio de aprender los factores causales subyacentes, como lo señaló [Scholkopf et al. \(2012\)](#), es que si el verdadero proceso generativo tiene X como efecto y y como causa, luego modelar $pag(X/y)$ es robusto a los cambios en $pag(y)$. Si se invirtiera la relación causa-efecto, esto no sería cierto, ya que por la regla de Bayes, $pag(X/y)$ sería sensible a los cambios en $pag(y)$. Muy a menudo, cuando consideramos cambios en la distribución debido a diferentes dominios, no estacionariedad temporal o cambios en la naturaleza de la tarea, *los mecanismos causales permanecen invariantes* (las leyes del universo son constantes), mientras que la distribución marginal sobre las causas subyacentes puede cambiar. Por lo tanto, una mejor generalización y robustez a todo tipo de cambios puede

esperarse a través del aprendizaje de un modelo generativo que intente recuperar los factores causales $shyp(X/h)$.

15.4 Representación Distribuida

Las representaciones distribuidas de conceptos, representaciones compuestas de muchos elementos que se pueden configurar por separado, son una de las herramientas más importantes para el aprendizaje de representaciones. Las representaciones distribuidas son poderosas porque pueden usar n características con k valores para describir k diferentes conceptos. Como hemos visto a lo largo de este libro, tanto las redes neuronales con múltiples unidades ocultas como los modelos probabilísticos con múltiples variables latentes hacen uso de la estrategia de representación distribuida. Ahora introducimos una observación adicional. Muchos algoritmos de aprendizaje profundo están motivados por la suposición de que las unidades ocultas pueden aprender a representar los factores causales subyacentes que explican los datos, como se analiza en la sección 15.3. Las representaciones distribuidas son naturales para este enfoque, porque cada dirección en el espacio de representación puede corresponder al valor de una variable de configuración subyacente diferente.

Un ejemplo de una representación distribuida es un vector de n características binarias, que pueden tomar 2^n configuraciones, cada una potencialmente correspondiente a una región diferente en el espacio de entrada, como se ilustra en la figura 15.7. Esto se puede comparar con una representación simbólica, donde la entrada está asociada con un solo símbolo o categoría. Si hay n símbolos en el diccionario, uno puede imaginar n detectores de características, cada uno correspondiente a la detección de la presencia de la categoría asociada. En ese caso solo n son posibles diferentes configuraciones del espacio de representación, tallando n diferentes regiones en el espacio de entrada, como se ilustra en la figura 15.8. Tal representación simbólica también se llama representación one-hot, ya que puede ser capturada por un vector binario con n bits que son mutuamente excluyentes (solo uno de ellos puede estar activo). Una representación simbólica es un ejemplo específico de la clase más amplia de representaciones no distribuidas, que son representaciones que pueden contener muchas entradas pero sin un control separado significativo sobre cada entrada.

Ejemplos de algoritmos de aprendizaje basados en representaciones no distribuidas incluyen:

- Los métodos de agrupamiento, incluido el k -algoritmo de medios: cada punto de entrada se asigna exactamente a un grupo.
- k -Algoritmos de vecinos más cercanos: una o unas pocas plantillas o ejemplos de prototipos están asociados con una entrada dada. En el caso de $k > 1$, hay varios

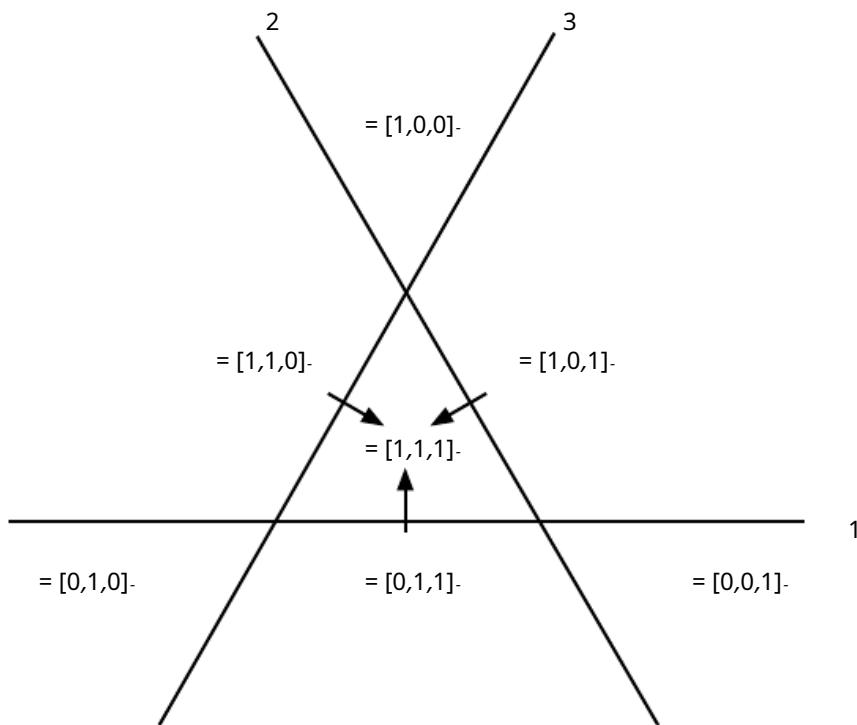


Figura 15.7: Ilustración de cómo un algoritmo de aprendizaje basado en una representación distribuida divide el espacio de entrada en regiones. En este ejemplo, hay tres características binarias. $h_1, h_2, y h_3$. Cada característica se define mediante el umbral de la salida de un aprendizaje lineal transformación. Cada característica divideR₂en dos semiplanos. Dejar h_+ ser el conjunto de entrada puntos por los cuales $h=1$ h_- Sea el conjunto de puntos de entrada para los cuales $h=0$. En esto ilustración, cada línea representa el límite de decisión para una h_i , con el correspondiente flecha que apunta a la h_+ lado de la frontera. La representación en su conjunto toma en un valor único en cada posible intersección de estos semiplanos. por ejemplo, el valor de representación $[1,1,1]$ -corresponde a la regiónh₊ 1+ 2+ 3. Compara esto con el representaciones no distribuidas en figura15.8. En el caso general deddimensiones de entrada, una representación distribuida divideR_dmediante la intersección de semiespacios en lugar de semiplanos. La representación distribuida connortecaracterísticas asigna códigos únicos a O(*norte*_d)diferentes regiones, mientras que el algoritmo vecino más cercano connortee ejemplos asigna códigos únicos a solo *norte*regiones. La representación distribuida es así capaz de distinguir exponencialmente muchas más regiones que la no distribuida. Tenga en cuenta que no todos hvalores son factibles (no hay $h=0$ en este ejemplo) y que un clasificador lineal encima de la representación distribuida no puede asignar diferentes identidades de clase a cada región vecina; incluso una red profunda de umbral lineal tiene una dimensión de VC de solo $O(w \log w)$ dónde w es el número de pesos (Sontag, 1998). La combinación de una capa de representación poderosa y una capa clasificadora débil puede ser un regularizador fuerte; un clasificador que intenta aprender el concepto de "persona" versus "no una persona" no necesita asignar una clase diferente a una entrada representada como "mujer con anteojos" que la que asigna a una entrada representada como "hombre sin anteojos". Esta restricción de capacidad alienta a cada clasificador a concentrarse en unos pocos h y alienta a aprender a representar las clases de forma linealmente separable.

valores que describen cada entrada, pero no se pueden controlar por separado, por lo que esto no califica como una verdadera representación distribuida.

- Árboles de decisión: solo una hoja (y los nodos en el camino de la raíz a la hoja) se activa cuando se proporciona una entrada.
- Mezclas gaussianas y mezclas de expertos: las plantillas (cluster centers) o expertos ahora se asocian a un grado de activación. Al igual que con el k -Algoritmo de vecinos más cercanos, cada entrada se representa con múltiples valores, pero esos valores no se pueden controlar fácilmente por separado.
- Máquinas kernel con un kernel gaussiano (u otro kernel local similar): aunque el grado de activación de cada "vector de soporte" o ejemplo de plantilla ahora tiene un valor continuo, surge el mismo problema que con las mezclas gaussianas.
- Lenguaje o modelos de traducción basados en n-gramos. El conjunto de contextos (secuencias de símbolos) se divide según una estructura de árbol de sufijos. Una hoja puede corresponder a las dos últimas palabras siendo $w_1 w_2$. Por ejemplo. Se estiman parámetros separados para cada hoja del árbol (es posible que se compartan).

Para algunos de estos algoritmos no distribuidos, la salida no es constante por partes, sino que se interpola entre regiones vecinas. La relación entre el número de parámetros (o ejemplos) y el número de regiones que pueden definir sigue siendo lineal.

Un importante concepto relacionado que distingue una representación distribuida de una simbólica es que la generalización surge debido a los atributos compartidos entre diferentes conceptos. Como puros símbolos, "gato" y "perro" están tan lejos el uno del otro como cualquier otro símbolo. Sin embargo, si uno los asocia con una representación distribuida significativa, muchas de las cosas que se pueden decir sobre los gatos se pueden generalizar a los perros y viceversa. Por ejemplo, nuestra representación distribuida puede contener entradas como "tiene_piel" o "número_de_patas" que tienen el mismo valor para la incrustación de ambos "gato" y "perro." Los modelos de lenguaje neural que operan sobre representaciones distribuidas de palabras generalizan mucho mejor que otros modelos que operan directamente sobre representaciones de palabras one-hot, como se analiza en la sección 12.4. Las representaciones distribuidas inducen una rica *espacio de similitud*, en el que los conceptos (o entradas) cercanos semánticamente están cerca en la distancia, una propiedad que está ausente de las representaciones puramente simbólicas.

¿Cuándo y por qué puede haber una ventaja estadística al usar una representación distribuida como parte de un algoritmo de aprendizaje? Las representaciones distribuidas pueden

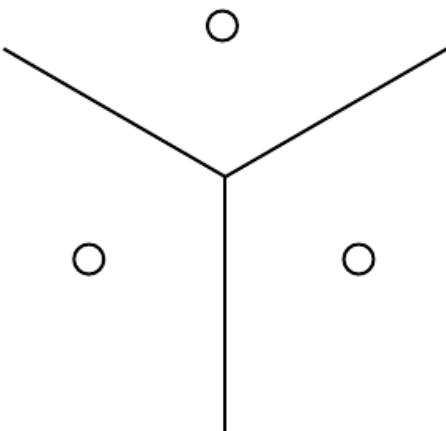


Figura 15.8: Ilustración de cómo el algoritmo del vecino más cercano divide el espacio de entrada en diferentes regiones. El algoritmo del vecino más cercano proporciona un ejemplo de un algoritmo de aprendizaje basado en una representación no distribuida. Diferentes algoritmos no distribuidos pueden tener una geometría diferente, pero normalmente dividen el espacio de entrada en regiones, *con un conjunto separado de parámetros para cada región*. La ventaja de un enfoque no distribuido es que, dados suficientes parámetros, puede ajustarse al conjunto de entrenamiento sin resolver un algoritmo de optimización difícil, porque es sencillo elegir una salida diferente *independientemente* para cada región. La desventaja es que dichos modelos no distribuidos se generalizan solo localmente a través de la suavidad previa, lo que dificulta el aprendizaje de una función complicada con más picos y valles que el número disponible de ejemplos. Contraste esto con una representación distribuida, figura 15.7.

tienen una ventaja estadística cuando una estructura aparentemente complicada se puede representar de forma compacta utilizando un pequeño número de parámetros. Algunos algoritmos de aprendizaje tradicionales no distribuidos generalizan solo debido a la suposición de suavidad, que establece que si $tu \approx v$, entonces la función objetivo f que el aprendido tiene la propiedad de que $f(tu) \approx f(v)$, en general. Hay muchas formas de formalizar tal suposición, pero el resultado final es que si tenemos un ejemplo (x, y) por lo que sabemos que $f(x) \approx y$, entonces elegimos un estimador f que satisfaga aproximadamente estas restricciones cambiando lo menos posible cuando nos movemos a una entrada cercana $x + \epsilon$. Esta suposición es claramente muy útil, pero adolece de la maldición de la dimensionalidad: para aprender una función objetivo que aumenta y disminuye muchas veces en muchas regiones diferentes,¹ es posible que necesitemos un número de ejemplos que sea al menos tan grande como el número de regiones distinguibles. Uno puede pensar en cada una de estas regiones como una categoría o símbolo: al tener un grado de libertad separado para cada símbolo (o región), podemos aprender una asignación de decodificador arbitraria de símbolo a valor. Sin embargo, esto no nos permite generalizar a nuevos símbolos para nuevas regiones.

Si tenemos suerte, puede haber cierta regularidad en la función objetivo, además de ser suave. Por ejemplo, una red convolucional con max-pooling puede reconocer un objeto independientemente de su ubicación en la imagen, aunque la traducción espacial del objeto no se corresponda con transformaciones uniformes en el espacio de entrada.

Examinemos un caso especial de un algoritmo de aprendizaje de representación distribuida, que extrae características binarias mediante el umbral de funciones lineales de la entrada. Cada característica binaria en esta representación divide \mathbb{R}^d en un par de medios espacios, como se ilustra en la figura 15.7. El número exponencialmente grande de intersecciones de n de los semiespacios correspondientes determina cuántas regiones puede distinguir este alumno de representación distribuida. ¿Cuántas regiones se generan por un arreglo de n hiperplanos en \mathbb{R}^d ? Aplicando un resultado general relativo a la intersección de hiperplanos (Zaslavsky, 1975), uno puede mostrar (Pascanuet et al., 2014b) que el número de regiones que esta representación binaria de características puede distinguir es

$$\sum_{j=0}^{2^n - 1} j = O(n^d). \quad (15.4)$$

Por lo tanto, vemos un crecimiento exponencial en el tamaño de entrada y polinomial en el número de unidades ocultas.

¹Potencialmente, es posible que queramos aprender una función cuyo comportamiento sea distinto en muchas regiones exponencialmente: en un d -espacio dimensional con al menos 2 valores diferentes para distinguir por dimensión, podríamos querer diferir en 2^d diferentes regiones, que requieren $O(2^d)$ ejemplos de entrenamiento.

Esto proporciona un argumento geométrico para explicar el poder de generalización de la representación distribuida: con $O(\text{Dakota del Norte})$ parámetros (para n nortes) características de umbral lineal en R_d) podemos representar claramente $O(norte_d)$ regiones en el espacio de entrada. Si, en cambio, no hiciéramos ninguna suposición sobre los datos y usáramos una representación con un símbolo único para cada región y parámetros separados para cada símbolo para reconocer su porción correspondiente de R_d , luego especificando $O(norte_d)$ regiones requerirían $O(norte_d)$ ejemplos. En términos más generales, el argumento a favor de la representación distribuida podría extenderse al caso en que en lugar de utilizar unidades de umbral lineales, utilicemos extractores de características no lineales, posiblemente continuas, para cada uno de los atributos de la representación distribuida. El argumento en este caso es que si una transformación paramétrica con k los parámetros pueden aprender sobre r regiones en el espacio de entrada, con $k - r$, y si obtener tal representación fuera útil para la tarea de interés, entonces podríamos generalizar mucho mejor de esta manera que en un entorno no distribuido donde necesitaríamos $O(r)$ ejemplos para obtener las mismas características y la partición asociada del espacio de entrada en r regiones. Usar menos parámetros para representar el modelo significa que tenemos menos parámetros para ajustar y, por lo tanto, necesitamos muchos menos ejemplos de entrenamiento para generalizar bien.

Otra parte del argumento de por qué los modelos basados en representaciones distribuidas se generalizan bien es que su capacidad sigue siendo limitada a pesar de poder codificar claramente tantas regiones diferentes. Por ejemplo, la dimensión VC de una red neuronal de unidades de umbral lineal es solo $O(w \cdot \text{registro})$, donde w es el número de pesos (Sontag, 1998). Esta limitación surge porque, si bien podemos asignar muchos códigos únicos al espacio de representación, no podemos usar absolutamente todo el espacio de código, ni podemos aprender el mapeo de funciones arbitrarias del espacio de representación. *ha la salida* utilizando un clasificador lineal. El uso de una representación distribuida combinada con un clasificador lineal expresa así una creencia previa de que las clases a reconocer son linealmente separables en función de los factores causales subyacentes capturados por h . Por lo general, querremos aprender categorías como el conjunto de todas las imágenes de todos los objetos verdes o el conjunto de todas las imágenes de automóviles, pero no categorías que requieran lógica XOR no lineal. Por ejemplo, normalmente no queremos dividir los datos en el conjunto de todos los autos rojos y camiones verdes como una clase y el conjunto de todos los autos verdes y camiones rojos como otra clase.

Las ideas discutidas hasta ahora han sido abstractas, pero pueden validarse experimentalmente. Zhou et al. (2015) encuentran que las unidades ocultas en una red convolucional profunda entrenada en los conjuntos de datos de referencia de ImageNet y Places aprenden características que muy a menudo son interpretables, correspondientes a una etiqueta que los humanos asignarían naturalmente. En la práctica, ciertamente no siempre es el caso de que las unidades ocultas aprendan algo que tiene un nombre lingüístico simple, pero es interesante ver que esto emerge cerca de los niveles superiores de las mejores redes profundas de visión artificial. ¿Qué tienen tales características en

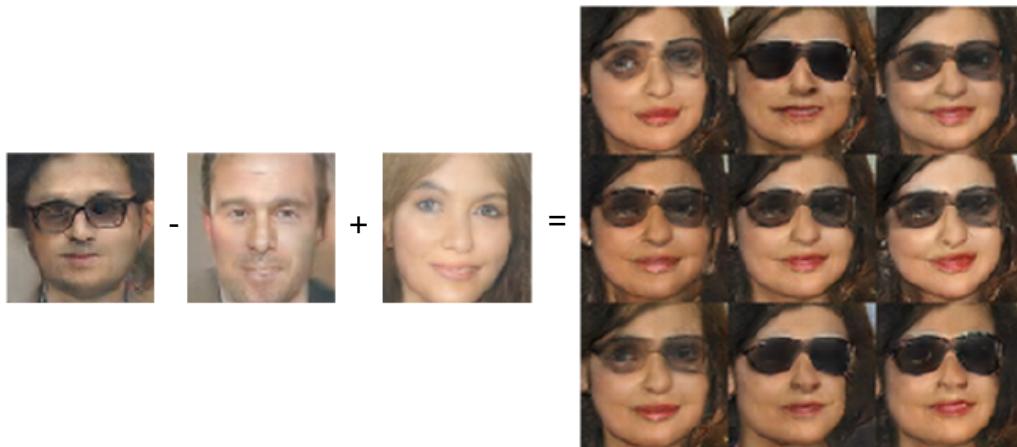


Figura 15.9: Un modelo generativo ha aprendido una representación distribuida que separa el concepto de género del concepto de llevar gafas. Si comenzamos con la representación del concepto de hombre con anteojos, luego restamos el vector que representa el concepto de hombre sin anteojos, y finalmente sumamos el vector que representa el concepto de mujer sin anteojos, obtenemos el vector que representa el concepto de una mujer con gafas. El modelo generativo decodifica correctamente todos estos vectores de representación en imágenes que pueden reconocerse como pertenecientes a la clase correcta. Imágenes reproducidas con permiso de [Radford et al.\(2015\)](#).

común es que uno se pueda imaginar *aprender sobre cada uno de ellos sin tener que ver todas las configuraciones de todos los demás*. [Radford et al.\(2015\)](#) demostraron que un modelo generativo puede aprender una representación de imágenes de caras, con direcciones separadas en el espacio de representación capturando diferentes factores subyacentes de variación. Cifra 15.9 demuestra que una dirección en el espacio de representación corresponde a si la persona es hombre o mujer, mientras que otra corresponde a si la persona lleva gafas. Estas características se descubrieron automáticamente, no se fijaron a priori. No es necesario tener etiquetas para los clasificadores de unidades ocultas: el descenso de gradiente en una función objetiva de interés aprende naturalmente características semánticamente interesantes, siempre que la tarea requiera dichas características. Podemos aprender sobre la distinción entre hombre y mujer, o sobre la presencia o ausencia de anteojos, sin tener que caracterizar todas las configuraciones del norte -1 otras características mediante ejemplos que cubren todas estas combinaciones de valores. Esta forma de separabilidad estadística es lo que permite generalizar a nuevas configuraciones de las características de una persona que nunca se han visto durante el entrenamiento.

15.5 Ganancias exponenciales de profundidad

Hemos visto en la sección 6.4.1 que los perceptrones multicapa son aproximadores universales y que algunas funciones pueden representarse mediante redes profundas exponencialmente más pequeñas en comparación con las redes superficiales. Esta disminución en el tamaño del modelo conduce a una mayor eficiencia estadística. En esta sección, describimos cómo resultados similares se aplican de manera más general a otros tipos de modelos con representaciones ocultas distribuidas.

En la sección 15.4, vimos un ejemplo de un modelo generativo que aprendió sobre los factores explicativos que subyacen a las imágenes de rostros, incluido el sexo de la persona y si lleva gafas. El modelo generativo que logró esta tarea se basó en una red neuronal profunda. No sería razonable esperar que una red superficial, como una red lineal, aprenda la complicada relación entre estos factores explicativos abstractos y los píxeles de la imagen. En esta y otras tareas de IA, es más probable que los factores que se pueden elegir de forma casi independiente entre sí pero que aún corresponden a entradas significativas sean de muy alto nivel y estén relacionados de manera altamente no lineal con la entrada. Argumentamos que esto exige *profundas* representaciones distribuidas, donde las características de nivel superior (vistas como funciones de la entrada) o factores (vistas como causas generativas) se obtienen a través de la composición de muchas no linealidades.

Se ha demostrado en muchos entornos diferentes que la organización de la computación a través de la composición de muchas no linealidades y una jerarquía de características reutilizadas puede dar un impulso exponencial a la eficiencia estadística, además del impulso exponencial proporcionado por el uso de una representación distribuida. Se puede demostrar que muchos tipos de redes (p. ej., con no linealidades saturadas, puertas booleanas, sumas/productos o unidades RBF) con una sola capa oculta son aproximadores universales. Una familia de modelos que es un aproximador universal puede aproximar una gran clase de funciones (incluidas todas las funciones continuas) hasta cualquier nivel de tolerancia distinto de cero, dadas suficientes unidades ocultas. Sin embargo, el número requerido de unidades ocultas puede ser muy grande. k , pero requeriría un número exponencial de unidades ocultas (con respecto al tamaño de entrada) con una profundidad insuficiente (profundidad 2 o profundidad $k - 1$).

En la sección 6.4.1, vimos que las redes feedforward deterministas son aproximadores universales de funciones. Muchos modelos probabilísticos estructurados con una sola capa oculta de variables latentes, incluidas las máquinas de Boltzmann restringidas y las redes de creencias profundas, son aproximadores universales de las distribuciones de probabilidad ([Le Roux y Bengio, 2008, 2010; Montúfar y Ay, 2011; Montúfar, 2014; Krause et al., 2013](#)).

En la sección 6.4.1, vimos que una red feedforward suficientemente profunda puede tener una ventaja exponencial sobre una red demasiado superficial. Estos resultados también se pueden obtener para otros modelos, como los modelos probabilísticos. Uno de estos modelos probabilísticos es el **red de suma-producto** SPN (Poon y Domingos, 2011). Estos modelos utilizan circuitos polinómicos para calcular la distribución de probabilidad sobre un conjunto de variables aleatorias. Delalleau y Bengio (2011) mostró que existen distribuciones de probabilidad para las cuales se requiere una profundidad mínima de SPN para evitar la necesidad de un modelo exponencialmente grande. Más tarde, Martas y medabalimi (2014) mostró que existen diferencias significativas entre cada dos profundidades finitas de SPN, y que algunas de las restricciones utilizadas para hacer manejables los SPN pueden limitar su poder de representación.

Otro desarrollo interesante es un conjunto de resultados teóricos para el poder expresivo de familias de circuitos profundos relacionados con redes convolucionales, destacando una ventaja exponencial para el circuito profundo incluso cuando se permite que el circuito superficial solo se aproxime a la función calculada por el circuito profundo (Cohen et al., 2015). En comparación, el trabajo teórico anterior hizo afirmaciones con respecto solo al caso en el que el circuito superficial debe replicar exactamente funciones particulares.

15.6 Proporcionar pistas para descubrir las causas subyacentes

Para cerrar este capítulo, volvemos a una de nuestras preguntas originales: ¿qué hace que una representación sea mejor que otra? Una respuesta, presentada por primera vez en la sección 15.3, es que una representación ideal es aquella que desenreda los factores causales subyacentes de variación que generaron los datos, especialmente aquellos factores que son relevantes para nuestras aplicaciones. La mayoría de las estrategias para el aprendizaje de representaciones se basan en la introducción de pistas que ayudan al aprendizaje a encontrar estos factores subyacentes de variaciones. Las pistas pueden ayudar al alumno a separar estos factores observados de los demás. El aprendizaje supervisado proporciona una pista muy fuerte: una etiqueta y , presentada con cada X , que suele especificar directamente el valor de al menos uno de los factores de variación. De manera más general, para hacer uso de abundantes datos no etiquetados, el aprendizaje de representación hace uso de otras pistas menos directas sobre los factores subyacentes. Estas sugerencias toman la forma de creencias previas implícitas que nosotros, los diseñadores del algoritmo de aprendizaje, imponemos para guiar al alumno. Resultados como el teorema de que no hay comida gratis muestran que las estrategias de regularización son necesarias para obtener una buena generalización. Si bien es imposible encontrar una estrategia de regularización universalmente superior, un objetivo del aprendizaje profundo es encontrar un conjunto de estrategias de regularización bastante genéricas que sean aplicables a una amplia variedad de tareas de IA, similares a las tareas que las personas y los animales pueden resolver..

Proporcionamos aquí una lista de estas estrategias genéricas de regularización. La lista claramente no es exhaustiva, pero brinda algunos ejemplos concretos de formas en que se puede alentar a los algoritmos de aprendizaje a descubrir características que corresponden a factores subyacentes. Esta lista se introdujo en la sección 3.1 de [bengio et al. \(2013d\)](#) y se ha ampliado parcialmente aquí.

- *Suavidad*: Esta es la suposición de que $R(X + \cdot d) \approx R(X)$ por unidad d pequeña-. Esta suposición le permite al alumno generalizar a partir de ejemplos de entrenamiento a puntos cercanos en el espacio de entrada. Muchos algoritmos de aprendizaje automático aprovechan esta idea, pero es insuficiente para superar la maldición de la dimensionalidad.
- *linealidad*: Muchos algoritmos de aprendizaje asumen que las relaciones entre algunas variables son lineales. Esto permite que el algoritmo haga predicciones incluso muy alejadas de los datos observados, pero a veces puede conducir a predicciones demasiado extremas. La mayoría de los algoritmos de aprendizaje automático simples que no hacen la suposición de suavidad en su lugar hacen la suposición de linealidad. De hecho, se trata de suposiciones diferentes: las funciones lineales con grandes pesos aplicados a espacios de alta dimensión pueden no ser muy suaves. Ver [Buen compañero et al. \(2014b\)](#) para una discusión más detallada de las limitaciones del supuesto de linealidad.
- *Múltiples factores explicativos*: Muchos algoritmos de aprendizaje de representación están motivados por la suposición de que los datos son generados por múltiples factores explicativos subyacentes y que la mayoría de las tareas se pueden resolver fácilmente dado el estado de cada uno de estos factores. Sección [15.3](#) describe cómo esta vista motiva el aprendizaje semisupervisado a través del aprendizaje de representación. Aprender la estructura de $p_{\text{ag}}(X)$ requiere aprender algunas de las mismas características que son útiles para modelar $p_{\text{ag}}(y | X)$ porque ambos se refieren a los mismos factores explicativos subyacentes. Sección [15.4](#) describe cómo esta vista motiva el uso de representaciones distribuidas, con direcciones separadas en el espacio de representación correspondientes a factores de variación separados.
- *Factores casuales*: el modelo está construido de tal manera que trata los factores de variación descritos por la representación aprendida h como las causas de los datos observados X , y no al revés. Como se discutió en la sección [15.3](#), esto es ventajoso para el aprendizaje semisupervisado y hace que el modelo aprendido sea más robusto cuando la distribución sobre las causas subyacentes cambia o cuando usamos el modelo para una nueva tarea.
- *Profundidad, o una organización jerárquica de factores explicativos*: Los conceptos abstractos de alto nivel se pueden definir en términos de conceptos simples, formando una jerarquía. Desde otro punto de vista, el uso de una arquitectura profunda

expresa nuestra creencia de que la tarea debe realizarse a través de un programa de varios pasos, con cada paso refiriéndose a la salida del procesamiento realizado a través de los pasos anteriores.

- *Factores compartidos entre tareas*: En el contexto donde tenemos muchas tareas, correspondientes a diferentes y variables que comparten la misma entrada X donde cada tarea está asociada con un subconjunto o una función $F_i(X)$ de una entrada global X , la suposición es que cada y está asociado con un subconjunto diferente de un grupo común de factores relevantes H . Debido a que estos subconjuntos se superponen, aprender todos los $PAG(y_i/X)$ a través de una representación intermedia compartida $PAG(h/X)$ permite compartir la fuerza estadística entre las tareas.
- *Colectores*: La masa de probabilidad se concentra, y las regiones en las que se concentra están conectadas localmente y ocupan un volumen diminuto. En el caso continuo, estas regiones se pueden aproximar mediante variedades de baja dimensión con una dimensionalidad mucho más pequeña que el espacio original donde residen los datos. Muchos algoritmos de aprendizaje automático se comportan de manera sensata solo en esta variedad ([Buen compañero et al., 2014b](#)). Algunos algoritmos de aprendizaje automático, especialmente los codificadores automáticos, intentan aprender explícitamente la estructura de la variedad.
- *Agrupación natural*: Muchos algoritmos de aprendizaje automático asumen que cada variedad conectada en el espacio de entrada puede asignarse a una sola clase. Los datos pueden estar en muchas variedades desconectadas, pero la clase permanece constante dentro de cada una de ellas. Esta suposición motiva una variedad de algoritmos de aprendizaje, incluida la propagación de tangente, el backprop doble, el clasificador de tangente múltiple y el entrenamiento de confrontación.
- *Coherencia temporal y espacial*: El análisis de características lentes y los algoritmos relacionados asumen que los factores explicativos más importantes cambian lentamente con el tiempo, o al menos que es más fácil predecir los verdaderos factores explicativos subyacentes que predecir observaciones sin procesar, como valores de píxeles. Mira la sección [13.3](#) para una descripción más detallada de este enfoque.
- *escasez*: La mayoría de las funciones presumiblemente no deberían ser relevantes para describir la mayoría de las entradas; no es necesario utilizar una función que detecte la trompa de elefante cuando se representa la imagen de un gato. Por lo tanto, es razonable imponer un principio de que cualquier característica que pueda interpretarse como “presente” o “ausente” debería estar ausente la mayor parte del tiempo.
- *Simplicidad de las dependencias de los factores*: En buenas representaciones de alto nivel, los factores se relacionan entre sí a través de dependencias simples. Lo más simple

posible es la independencia marginal, $PAG(h) = \prod_i PAG_i(h_i)$, pero dependencias lineales o los capturados por un autocodificador superficial también son suposiciones razonables. Esto se puede ver en muchas leyes de la física, y se asume cuando se conecta un predictor lineal o factorizado anterior sobre una representación aprendida.

El concepto de aprendizaje de representación une todas las muchas formas de aprendizaje profundo. Las redes feedforward y recurrentes, los codificadores automáticos y los modelos probabilísticos profundos aprenden y explotan las representaciones. Aprender la mejor representación posible sigue siendo una interesante vía de investigación.

capítulo 16

Modelos probabilísticos estructurados para aprendizaje profundo

El aprendizaje profundo se basa en muchos formalismos de modelado que los investigadores pueden usar para guiar sus esfuerzos de diseño y describir sus algoritmos. Uno de estos formalismos es la idea de **modelos probabilísticos estructurados**. Ya hemos discutido brevemente los modelos probabilísticos estructurados en la sección 3.14. Esa breve presentación fue suficiente para entender cómo usar modelos probabilísticos estructurados como un lenguaje para describir algunos de los algoritmos en parte. Ahora, en parte tercero Los modelos probabilísticos estructurados son un ingrediente clave de muchos de los temas de investigación más importantes en el aprendizaje profundo. Para prepararse para discutir estas ideas de investigación, este capítulo describe modelos probabilísticos estructurados con mucho más detalle. Este capítulo pretende ser independiente; el lector no necesita revisar la introducción anterior antes de continuar con este capítulo.

Un modelo probabilístico estructurado es una forma de describir una distribución de probabilidad, utilizando un gráfico para describir qué variables aleatorias en la distribución de probabilidad interactúan entre sí directamente. Aquí usamos "grafo" en el sentido de la teoría de grafos: un conjunto de vértices conectados entre sí por un conjunto de aristas. Debido a que la estructura del modelo se define mediante un gráfico, estos modelos a menudo también se conocen como **modelos gráficos**.

La comunidad de investigación de modelos gráficos es grande y ha desarrollado muchos modelos diferentes, algoritmos de entrenamiento y algoritmos de inferencia. En este capítulo, brindamos antecedentes básicos sobre algunas de las ideas más centrales de los modelos gráficos, con énfasis en los conceptos que han demostrado ser más útiles para la comunidad de investigación de aprendizaje profundo. Si ya tiene una sólida formación en modelos gráficos, es posible que desee omitir la mayor parte de este capítulo. Sin embargo, incluso un experto en modelos gráficos

puede beneficiarse de la lectura de la sección final de este capítulo, la sección 16.7, en el que destacamos algunas de las formas únicas en que los modelos gráficos se utilizan para los algoritmos de aprendizaje profundo. Los profesionales del aprendizaje profundo tienden a utilizar estructuras de modelo, algoritmos de aprendizaje y procedimientos de inferencia muy diferentes a los que suele utilizar el resto de la comunidad de investigación de modelos gráficos. En este capítulo, identificamos estas diferencias en las preferencias y explicamos las razones de ellas.

En este capítulo, primero describimos los desafíos de construir modelos probabilísticos a gran escala. A continuación, describimos cómo usar una gráfica para describir la estructura de una distribución de probabilidad. Si bien este enfoque nos permite superar muchos desafíos, no deja de tener sus propias complicaciones. Una de las mayores dificultades en el modelado gráfico es comprender qué variables deben poder interactuar directamente, es decir, qué estructuras gráficas son las más adecuadas para un problema dado. Describimos dos enfoques para resolver esta dificultad aprendiendo sobre las dependencias en la sección 16.5. Finalmente, cerramos con una discusión sobre el énfasis único que los profesionales del aprendizaje profundo ponen en enfoques específicos para el modelado gráfico en la sección 16.7.

16.1 El desafío del modelado no estructurado

El objetivo del aprendizaje profundo es escalar el aprendizaje automático a los tipos de desafíos necesarios para resolver la inteligencia artificial. Esto significa ser capaz de comprender datos de alta dimensión con una estructura rica. Por ejemplo, nos gustaría que los algoritmos de IA pudieran comprender imágenes naturales,¹ formas de onda de audio que representan el habla y documentos que contienen varias palabras y caracteres de puntuación.

Los algoritmos de clasificación pueden tomar una entrada de una distribución de alta dimensión tan rica y resumirla con una etiqueta categórica: qué objeto está en una foto, qué palabra se habla en una grabación, de qué tema trata un documento. El proceso de clasificación descarta la mayor parte de la información de la entrada y produce una única salida (o una distribución de probabilidad sobre los valores de esa única salida). El clasificador también suele ignorar muchas partes de la entrada. Por ejemplo, al reconocer un objeto en una foto, normalmente es posible ignorar el fondo de la foto.

Es posible pedir a los modelos probabilísticos que realicen muchas otras tareas. Estas tareas suelen ser más caras que la clasificación. Algunos de ellos requieren producir múltiples valores de salida. La mayoría requiere una comprensión completa de toda la estructura de

¹A **Imagen naturales** una imagen que podría ser capturada por una cámara en un entorno razonablemente normal, a diferencia de una imagen renderizada sintéticamente, una captura de pantalla de una página web, etc.

la entrada, sin opción de ignorar secciones de la misma. Estas tareas incluyen lo siguiente:

- **Estimación de densidad:** dada una entrada X , el sistema de aprendizaje automático devuelve una estimación de la densidad real $p_{\text{real}}(X)$ bajo la distribución de generación de datos. Esto requiere solo una única salida, pero requiere una comprensión completa de toda la entrada. Si incluso un elemento del vector es inusual, el sistema debe asignarle una probabilidad baja.
- **eliminación de ruido:** dada una entrada dañada o observada incorrectamente X , el sistema de aprendizaje automático devuelve una estimación del original o correcto X . Por ejemplo, se le puede pedir al sistema de aprendizaje automático que elimine el polvo o los rayones de una fotografía antigua. Esto requiere múltiples salidas (cada elemento del ejemplo limpia estimado X) y una comprensión de toda la entrada (ya que incluso un área dañada aún revelará la estimación final como dañada).
- **Imputación de valor faltante:** dadas las observaciones de algunos elementos de X , se le pide al modelo que devuelva estimaciones o una distribución de probabilidad sobre algunos o todos los elementos no observados de X . Esto requiere múltiples salidas. Porque se le podría pedir al modelo que restaurara cualquiera de los elementos de X , debe comprender toda la entrada.
- **Muestreo:** el modelo genera nuevas muestras a partir de la distribución $p_{\text{real}}(X)$. Las aplicaciones incluyen la síntesis de voz, es decir, la producción de nuevas formas de onda que suenan como el habla humana natural. Esto requiere múltiples valores de salida y un buen modelo de toda la entrada. Si las muestras tienen incluso un elemento extraído de una distribución incorrecta, entonces el proceso de muestreo es incorrecto.

Para ver un ejemplo de una tarea de muestreo utilizando pequeñas imágenes naturales, consulte la figura 16.1.

Modelar una distribución rica en miles o millones de variables aleatorias es una tarea desafiante, tanto desde el punto de vista computacional como estadístico. Supongamos que solo quisieramos modelar variables binarias. Este es el caso más simple posible y, sin embargo, ya parece abrumador. Para un pequeño, 32×32 imagen de color de píxel (RGB), hay 2^{3072} posibles imágenes binarias de esta forma. Este número se acabó 10800 veces mayor que el número estimado de átomos en el universo.

En general, si deseamos modelar una distribución sobre un vector aleatorio X que contiene $n_{\text{variables}}$ variables discretas capaces de asumir k valores cada uno, entonces el enfoque ingenuo de representar $P_{\text{real}}(X)$ al almacenar una tabla de búsqueda con un valor de probabilidad por posible resultado requiere $k^{n_{\text{variables}}}$ parámetros!

Esto no es factible por varias razones:

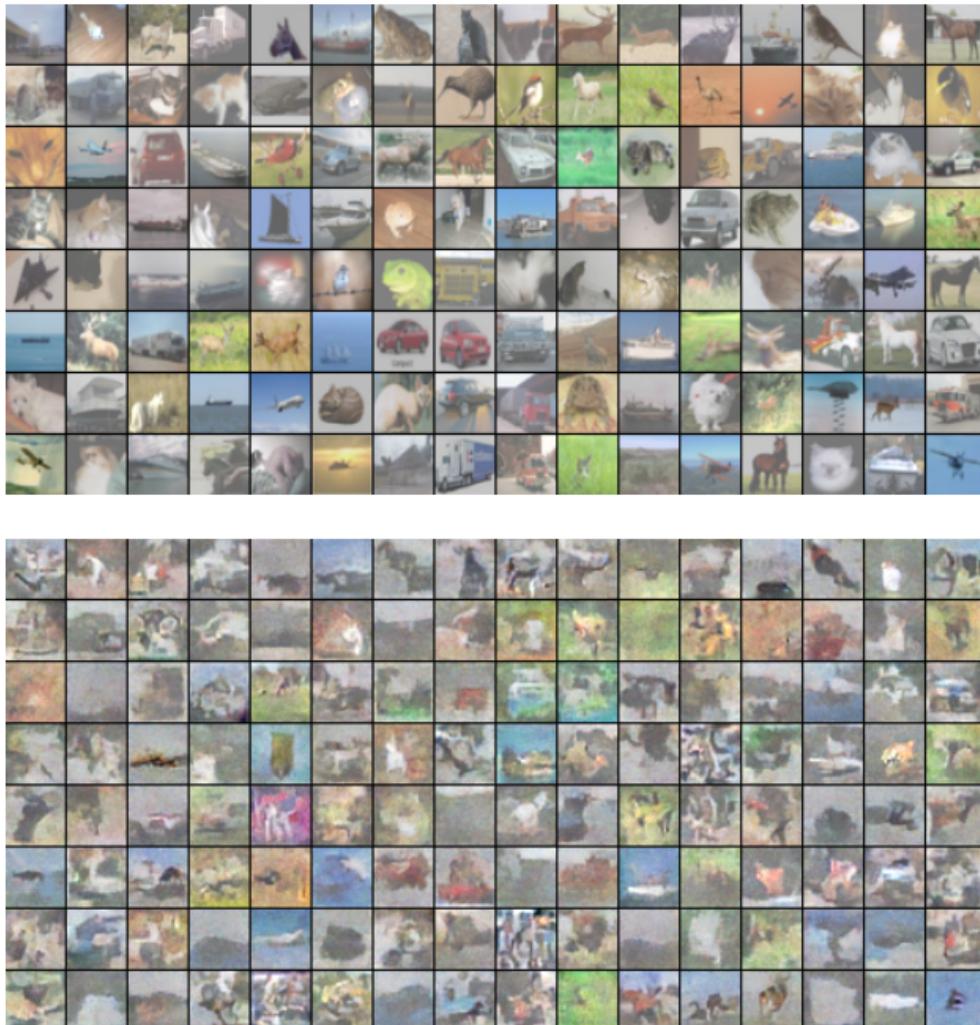


Figura 16.1: Modelado probabilístico de imágenes naturales. (*Arriba*) Ejemplo 32x32 imágenes en color de píxeles del conjunto de datos CIFAR-10 ([Krizhevsky y Hinton, 2009](#)). (*Abajo*) Muestras extraídas de un modelo probabilístico estructurado entrenado en este conjunto de datos. Cada muestra aparece en la misma posición en la cuadrícula que el ejemplo de entrenamiento más cercano en el espacio euclíadiano. Esta comparación nos permite ver que el modelo realmente sintetiza nuevas imágenes, en lugar de memorizar los datos de entrenamiento. El contraste de ambos conjuntos de imágenes se ha ajustado para su visualización. Figura reproducida con permiso de [courville et al. \(2011\)](#).

- *Memoria: el costo de almacenar la representación:* Para todos los valores de n y k , representar la distribución como una tabla requerirá demasiados valores para almacenar.
- *Eficiencia estadística:* A medida que aumenta la cantidad de parámetros en un modelo, también aumenta la cantidad de datos de entrenamiento necesarios para elegir los valores de esos parámetros usando un estimador estadístico. Debido a que el modelo basado en tablas tiene una cantidad astronómica de parámetros, requerirá un conjunto de entrenamiento astronómicamente grande para ajustarse con precisión. Cualquier modelo de este tipo se ajustará muy mal al conjunto de entrenamiento, a menos que se hagan suposiciones adicionales que vinculen las diferentes entradas en la tabla (por ejemplo, como en back-off o suavizado). *norte-modelos de gramo, sección 12.4.1.*
- *Tiempo de ejecución: el costo de la inferencia:* Supongamos que queremos realizar una tarea de inferencia donde usamos nuestro modelo de distribución conjunta $P(X)$ para calcular alguna otra distribución, como la distribución marginal $P(X_1)$ o la distribución condicional $P(X_2|X_1)$. Calcular estas distribuciones requerirá sumar toda la tabla, por lo que el tiempo de ejecución de estas operaciones es tan alto como el costo de memoria intratable de almacenar el modelo.
- *Tiempo de ejecución: el costo del muestreo:* Del mismo modo, supongamos que queremos tomar una muestra del modelo. La forma ingenua de hacer esto es probar algún valor $u \sim \text{Uniform}(0,1)$, luego iterar a través de la tabla, sumando los valores de probabilidad hasta que exceda u y devolver el resultado correspondiente a esa posición en la tabla. Esto requiere leer toda la tabla en el peor de los casos, por lo que tiene el mismo costo exponencial que las otras operaciones.

El problema con el enfoque basado en tablas es que estamos modelando explícitamente cada posible tipo de interacción entre cada posible subconjunto de variables. Las distribuciones de probabilidad que encontramos en tareas reales son mucho más simples que esto. Por lo general, la mayoría de las variables se influyen entre sí solo indirectamente.

Por ejemplo, considere modelar los tiempos de finalización de un equipo en una carrera de relevos. Supongamos que el equipo consta de tres corredores: Alice, Bob y Carol. Al comienzo de la carrera, Alice lleva un bastón y comienza a correr alrededor de una pista. Después de completar su vuelta alrededor de la pista, le entrega el testigo a Bob. Luego, Bob corre su propia vuelta y le entrega el bastón a Carol, quien corre la última vuelta. Podemos modelar cada uno de sus tiempos de finalización como una variable aleatoria continua. El tiempo de finalización de Alice no depende del de nadie más, ya que ella va primero. El tiempo de finalización de Bob depende del de Alice, porque Bob no tiene la oportunidad de comenzar su vuelta hasta que Alice haya completado la de ella. Si Alice termina más rápido, Bob terminará más rápido, siendo todo lo demás

igual. Finalmente, el tiempo de finalización de Carol depende de sus dos compañeros de equipo. Si Alice es lenta, es probable que Bob también termine tarde. Como consecuencia, Carol comenzará bastante tarde y, por lo tanto, es probable que también termine tarde. Sin embargo, el tiempo de finalización de Carol depende sólo *indirectamente* sobre el tiempo de finalización de Alice a través de Bob. Si ya conocemos la hora de finalización de Bob, no podremos estimar mejor la hora de finalización de Carol al averiguar cuál fue la hora de finalización de Alice. Esto significa que podemos modelar la carrera de relevos usando solo dos interacciones: el efecto de Alice sobre Bob y el efecto de Bob sobre Carol. Podemos omitir la tercera interacción indirecta entre Alice y Carol de nuestro modelo.

Los modelos probabilísticos estructurados proporcionan un marco formal para modelar solo interacciones directas entre variables aleatorias. Esto permite que los modelos tengan significativamente menos parámetros y, por lo tanto, se estimen de manera confiable a partir de menos datos. Estos modelos más pequeños también han reducido drásticamente el costo computacional en términos de almacenamiento del modelo, realización de inferencias en el modelo y extracción de muestras del modelo.

16.2 Uso de gráficos para describir la estructura del modelo

Los modelos probabilísticos estructurados usan gráficos (en el sentido de la teoría de gráficos de "nodos" o "vértices" conectados por bordes) para representar interacciones entre variables aleatorias. Cada nodo representa una variable aleatoria. Cada borde representa una interacción directa. Estas interacciones directas implican otras interacciones indirectas, pero solo las interacciones directas deben modelarse explícitamente.

Hay más de una manera de describir las interacciones en una distribución de probabilidad utilizando un gráfico. En las siguientes secciones describimos algunos de los enfoques más populares y útiles. Los modelos gráficos se pueden dividir en gran medida en dos categorías: modelos basados en gráficos acíclicos dirigidos y modelos basados en gráficos no dirigidos.

16.2.1 Modelos dirigidos

Un tipo de modelo probabilístico estructurado es el**modelo gráfico dirigido**, también conocido como el**red de creencias bayesiana**²(Perla, 1985).

Los modelos gráficos dirigidos se denominan "dirigidos" porque sus bordes están dirigidos,

²Judea Pearl sugirió usar el término "red bayesiana" cuando se desea "enfatizar la naturaleza crítica" de los valores calculados por la red, es decir, para resaltar que generalmente representan grados de creencia en lugar de frecuencias de eventos.

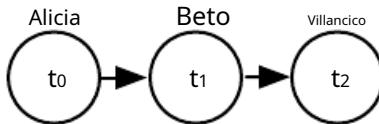


Figura 16.2: Un modelo gráfico dirigido que representa el ejemplo de carrera de relevos. hora de finalización de Alice t0 influye en el tiempo de finalización de Bob t1, porque Bob no puede comenzar a correr hasta que Alice termina. Del mismo modo, Carol solo puede comenzar a correr después de que Bob termina, por lo que el tiempo de finalización de Bob es t1influye directamente en el tiempo de finalización de Carol t2.

es decir, apuntan de un vértice a otro. Esta dirección se representa en el dibujo con una flecha. La dirección de la flecha indica la distribución de probabilidad de qué variable se define en términos de la otra. Dibujar una flecha de a a b significa que definimos la distribución de probabilidad sobre b a través de una distribución condicional, con a como una de las variables en el lado derecho de la barra de condicionamiento. En otras palabras, la distribución sobre b depende del valor de a.

Continuando con el ejemplo de la carrera de relevos de la sección 16.1, supongamos que nombramos el tiempo de finalización de Alice t0, tiempo de finalización de Bob t1, y la hora de finalización de Carol t2. Como vimos anteriormente, nuestra estimación de t1depende de t0. Nuestra estimación de t2depende directamente de t1pero solo indirectamente en t0. Podemos dibujar esta relación en un modelo gráfico dirigido, ilustrado en la figura 16.2.

Formalmente, un modelo gráfico dirigido definido sobre variables X está definido por un gráfico acíclico dirigido $GRAMO$ cuyos vértices son las variables aleatorias del modelo, y un conjunto de **distribuciones de probabilidad condicional locales** $pag(X_i / Pensilvaniagramo(X_i))$ donde $Pensilvania$ $GRAMO(X_i)$ da a los padres de x_i en $GRAMO$. La distribución de probabilidad sobre X es dado por

$$pag(x) = \prod_i pag(X_i / Pensilvaniagramo(X_i)). \quad (16.1)$$

En nuestro ejemplo de carrera de relevos, esto significa que, usando el gráfico dibujado en la figura 16.2,

$$pag(t_0, t_1, t_2) = pag(t_0) pag(t_1 / t_0) pag(t_2 / t_1). \quad (16.2)$$

Esta es la primera vez que vemos un modelo probabilístico estructurado en acción. Podemos examinar el costo de usarlo, para observar cómo el modelado estructurado tiene muchas ventajas en relación con el modelado no estructurado.

Supongamos que representamos el tiempo discretizando el tiempo que va desde el minuto 0 al minuto 10 en fragmentos de 6 segundos. Esto haría que t0, t1y T2cada uno sea una variable discreta con 100 valores posibles. Si intentáramos representar $pag(t_0, t_1, t_2)$ con una tabla, necesitaría almacenar 999,999 valores (100 valores de t0×100 valores de t1×100 valores de t2, menos 1, ya que la probabilidad de una de las configuraciones se hace

redundante por la restricción de que la suma de las probabilidades sea 1). Si en cambio, solo hacemos una tabla para cada una de las distribuciones de probabilidad condicional, entonces la distribución sobre t_0 requiere 99 valores, la tabla que define t_1 dado t_0 requiere 9900 valores, al igual que la tabla que define t_2 dado t_1 . Esto llega a un total de 19.899 valores. ¡Esto significa que el uso del modelo gráfico dirigido redujo nuestra cantidad de parámetros en un factor de más de 50!

En general, para modelar n variables discretas, cada una de las cuales tiene k valores, el costo del enfoque de tabla única escala como $O(k^n)$, como hemos observado antes. Supongamos ahora que construimos un modelo gráfico dirigido sobre estas variables. Si m es el número máximo de variables que aparecen (a cada lado de la barra de condicionamiento) en una sola distribución de probabilidad condicional, entonces el costo de las tablas para el modelo dirigido escala como $O(k^m)$. Mientras podamos diseñar un modelo tal que $m < n$, obtenemos ahorros muy dramáticos.

En otras palabras, siempre que cada variable tenga pocos padres en el gráfico, la distribución se puede representar con muy pocos parámetros. Algunas restricciones en la estructura del gráfico, como exigir que sea un árbol, también pueden garantizar que operaciones como calcular distribuciones marginales o condicionales sobre subconjuntos de variables sean eficientes.

Es importante darse cuenta de qué tipo de información puede y no puede codificarse en el gráfico. El gráfico codifica solo suposiciones simplificadoras sobre qué variables son condicionalmente independientes entre sí. También es posible hacer otro tipo de supuestos simplificadores. Por ejemplo, supongamos que asumimos que Bob siempre corre igual sin importar cómo se desempeñó Alice. (En realidad, el desempeño de Alice probablemente influya en el desempeño de Bob; dependiendo de la personalidad de Bob, si Alice corre especialmente rápido en una carrera determinada, esto podría animar a Bob a esforzarse e igualar su desempeño excepcional, o podría volverlo demasiado confiado y perezoso). Entonces, el único efecto que Alice tiene sobre el tiempo de finalización de Bob es que debemos sumar el tiempo de finalización de Alice a la cantidad total de tiempo que creemos que Bob necesita para correr. Esta observación nos permite definir un modelo con $O(k)$ parámetros en lugar de $O(k^2)$. Sin embargo, tenga en cuenta que t_0 y T_1 siguen siendo directamente dependientes de esta suposición, porque t_1 representa el tiempo absoluto en el que Bob termina, no el tiempo total que él mismo pasa corriendo. Esto significa que nuestro gráfico aún debe contener una flecha desde t_0 a t_1 . La suposición de que el tiempo de ejecución personal de Bob es independiente de todos los demás factores no se puede codificar en un gráfico sobre t_0 , t_1 , y T_2 . En cambio, codificamos esta información en la definición de la distribución condicional en sí. La distribución condicional ya no es una $k \times k - 1$ tabla de elementos indexada por t_0 y T_1 pero ahora es una fórmula un poco más complicada que usa solo $k - 1$ parámetros. La sintaxis del modelo gráfico dirigido no impone ninguna restricción sobre cómo definimos

nuestras distribuciones condicionales. Solo define qué variables se les permite tomar como argumentos.

16.2.2 Modelos no dirigidos

Los modelos gráficos dirigidos nos dan un lenguaje para describir modelos probabilísticos estructurados. Otro lenguaje popular es el **modelos no dirigidos**, de otra manera conocido como**Campos aleatorios de Markov**(MRF) o**Redes de Markov**(Kindermann, 1980). Como su nombre lo indica, los modelos no dirigidos usan gráficos cuyos bordes no están dirigidos.

Los modelos dirigidos se aplican de manera más natural a situaciones en las que existe una razón clara para dibujar cada flecha en una dirección particular. A menudo, estas son situaciones en las que entendemos la causalidad y la causalidad solo fluye en una dirección. Una de esas situaciones es el ejemplo de la carrera de relevos. Los corredores anteriores afectan los tiempos de finalización de los corredores posteriores; los corredores posteriores no afectan los tiempos de finalización de los corredores anteriores.

No todas las situaciones que querríamos modelar tienen una dirección tan clara en sus interacciones. Cuando las interacciones parecen no tener una dirección intrínseca u operar en ambas direcciones, puede ser más apropiado utilizar un modelo no dirigido.

Como ejemplo de tal situación, supongamos que queremos modelar una distribución sobre tres variables binarias: si usted está enfermo o no, si su compañero de trabajo está enfermo o no y si su compañero de cuarto está enfermo o no. Como en el ejemplo de la carrera de relevos, podemos hacer suposiciones simplificadoras sobre los tipos de interacciones que tienen lugar. Suponiendo que su compañero de trabajo y su compañero de cuarto no se conocen, es muy poco probable que uno de ellos le transmita al otro una infección como un resfriado directamente. Este evento puede verse como tan raro que es aceptable no modelarlo. Sin embargo, es razonablemente probable que cualquiera de ellos pueda causarle un resfriado y que usted se lo transmita al otro.

En este caso, es tan fácil para usted hacer que su compañero de cuarto se enferme como para que su compañero de cuarto lo haga a usted, por lo que no hay una narrativa limpia y unidireccional en la que basar el modelo. Esto motiva a utilizar un modelo no dirigido. Al igual que con los modelos dirigidos, si dos nodos en un modelo no dirigido están conectados por un borde, las variables aleatorias correspondientes a esos nodos interactúan entre sí directamente. A diferencia de los modelos dirigidos, el borde en un modelo no dirigido no tiene flecha y no está asociado con una distribución de probabilidad condicional.

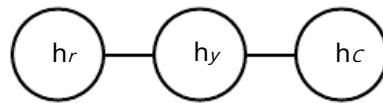


Figura 16.3: Un gráfico no dirigido que representa cómo la salud de su compañero de cuarto h_r , tu salud h_y , y la salud de su compañero de trabajo h_c afectarse unos a otros. Usted y su compañero de cuarto pueden infectarse mutuamente con un resfriado, y usted y su colega de trabajo pueden hacer lo mismo, pero asumiendo que su compañero de cuarto y su colega no se conocen, solo pueden infectarse indirectamente a través de usted.

Denotamos la variable aleatoria que representa su salud como h_y , la variable aleatoria que representa la salud de tu compañero de cuarto como h_r , y la variable aleatoria que representa la salud de su colega como h_c . Ver figura 16.3 para un dibujo del gráfico que representa este escenario.

Formalmente, un modelo gráfico no dirigido es un modelo probabilístico estructurado definido en un gráfico no dirigido *GRAMO*. Para cada camarilla C en el gráfico, ³afactor $\varphi(C)$ (también llamado un**potencial camarilla**) mide la afinidad de las variables en esa camarilla por estar en cada uno de sus posibles estados conjuntos. Los factores están restringidos a ser no negativos. Juntos definen una**distribución de probabilidad no normalizada**

$$p_{\text{ag}}(x) = \prod_{C \in G} \varphi(C). \quad (16.3)$$

Es eficiente trabajar con la distribución de probabilidad no normalizada siempre que todas las camarillas sean pequeñas. Codifica la idea de que los estados con mayor afinidad son más probables. Sin embargo, a diferencia de una red bayesiana, la definición de las camarillas tiene poca estructura, por lo que no hay nada que garantice que al multiplicarlas se obtendrá una distribución de probabilidad válida. Ver figura 16.4 para ver un ejemplo de lectura de información de factorización de un gráfico no dirigido.

Nuestro ejemplo del frío que se extiende entre usted, su compañero de cuarto y su colega contiene dos camarillas. Una camarilla contiene h_y y h_c . El factor para esta camarilla se puede definir mediante una tabla y puede tener valores parecidos a estos:

		$h_y=0$	$h_y=1$
		2	1
$h_c=0$	1	10	
	1		

³Una camarilla del gráfico es un subconjunto de nodos que están todos conectados entre sí por un borde del gráfico.

Un estado de 1 indica buena salud, mientras que un estado de 0 indica mala salud (haber sido infectado con un resfriado). Ambos suelen estar sanos, por lo que el estado correspondiente tiene la mayor afinidad. El estado en el que solo uno de ustedes está enfermo tiene la afinidad más baja, porque este es un estado raro. El estado en el que ambos están enfermos (porque uno de ustedes ha infectado al otro) es un estado de mayor afinidad, aunque aún no es tan común como el estado en el que ambos están sanos.

Para completar el modelo, también necesitaríamos definir un factor similar para la camarilla que contiene h_{YH} .

16.2.3 La función de partición

Si bien se garantiza que la distribución de probabilidad no normalizada no sea negativa en todas partes, no se garantiza que sume o integre a 1. Para obtener una distribución de probabilidad válida, debemos usar la probabilidad normalizada correspondiente distribución:⁴

$$p_{\text{ag}}(x) = \frac{1}{Z} p_{\text{ag}}(X) \quad (16.4)$$

dónde Z es el valor que resulta en la distribución de probabilidad sumando o integrando a 1:

$$Z = \int p_{\text{ag}}(X) dX. \quad (16.5)$$

Tu puedes pensar en Z como una constante cuando el φ las funciones se mantienen constantes. Tenga en cuenta que si el φ las funciones tienen parámetros, entonces Z es una función de esos parámetros. Es común en la literatura escribir Z con sus argumentos omitidos para ahorrar espacio. La constante de normalización Z es conocido como el **función de partición**, un término tomado de la física estadística.

Desde Z es una integral o suma sobre todas las asignaciones conjuntas posibles del estado X a menudo es intratable de calcular. Para poder obtener la distribución de probabilidad normalizada de un modelo no dirigido, la estructura del modelo y las definiciones de las φ funciones deben ser conducentes a la computación Z eficientemente. En el contexto del aprendizaje profundo, Z suele ser intratable. Debido a la intratabilidad de la informática, Z exactamente, debemos recurrir a aproximaciones. Tales algoritmos aproximados son el tema del capítulo 18.

Una consideración importante a tener en cuenta al diseñar modelos no dirigidos es que es posible especificar los factores de tal manera que Z no existe. Esto sucede si algunas de las variables en el modelo son continuas y la integral

⁴Una distribución definida mediante la normalización de un producto de potenciales de camarilla también se denomina distribución.

Distribución de Gibbs.

de $p_{\phi}(x)$ sobre su dominio diverge. Por ejemplo, supongamos que queremos modelar una sola variable escalar $x \in \mathbb{R}$ con un solo potencial de camarilla $\varphi(X) = X_2$. En este caso,

$$Z = \int_{-\infty}^{\infty} \exp(-X_2) dx. \quad (16.6)$$

Dado que esta integral diverge, no existe una distribución de probabilidad correspondiente a esta elección de $\varphi(X)$. A veces la elección de algún parámetro de la φ funciones determina dónde se define la distribución de probabilidad. por ejemplo, para $\varphi(X; \beta) = \exp(-\beta X_2)$, el β parámetro determina si Z existe. Si $\beta > 0$ da como resultado una distribución gaussiana sobre x , pero todos los demás valores de β hacen φ imposible de normalizar.

Una diferencia clave entre el modelado dirigido y el modelado no dirigido es que los modelos dirigidos se definen directamente en términos de distribuciones de probabilidad desde el principio, mientras que los modelos no dirigidos se definen más libremente por φ funciones que luego se convierten en distribuciones de probabilidad. Esto cambia las intuiciones que uno debe desarrollar para poder trabajar con estos modelos. Una idea clave a tener en cuenta al trabajar con modelos no dirigidos es que el dominio de cada una de las variables tiene un efecto dramático en el tipo de distribución de probabilidad que un conjunto dado de φ funciones a las que corresponde. Por ejemplo, considere una n -variable aleatoria de valor vectorial dimensional X y un modelo no dirigido parametrizado por un vector de sesgos b . Supongamos que tenemos una camarilla para cada elemento de X , $\varphi_i(X_i) = \exp(b_i X_i)$. ¿Qué tipo de distribución de probabilidad da esto como resultado? La respuesta es que no tenemos suficiente información, porque aún no hemos especificado el dominio de X . Si $X \in \mathbb{R}^n$, entonces la integral que define Z diverge y no existe distribución de probabilidad. Si $X \in \{0, 1\}^n$, entonces $p_{\phi}(X)$ factoriza en n distribuciones independientes, con $p_{\phi}(X_i=1) = \text{sigmoide}(b_i)$. Si el dominio de X es el conjunto de vectores base elementales ($\{[1, 0, \dots, 0], [0, 1, \dots, 0], \dots, [0, 0, \dots, 1]\}$), entonces $p_{\phi}(x) = \text{softmax}(b)$, por lo que un gran valor de b en realidad reduce $p_{\phi}(X=1)$ para $j=i$. A menudo, es posible aprovechar el efecto de un dominio cuidadosamente elegido de una variable para obtener un comportamiento complicado a partir de un conjunto relativamente simple de φ funciones. Exploraremos una aplicación práctica de esta idea más adelante, en la sección 20.6.

16.2.4 Modelos basados en energía

Muchos resultados teóricos interesantes sobre modelos no dirigidos dependen de la suposición de que $\forall X, p_{\phi}(X) > 0$. Una forma conveniente de hacer cumplir esta condición es usar un **modelo basado en energía** (EBM) donde

$$p_{\phi}(x) = \text{Exp}(-m_l(x)) \quad (16.7)$$

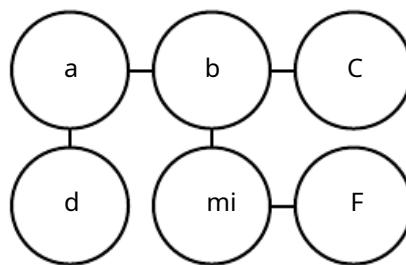


Figura 16.4: Este gráfico implica que $p_{\text{ag}}(a,b,C,d,\text{mi},f)$ se puede escribir como $\frac{1}{Z}\varphi_{a,b}(a,b)\varphi_{b,C}(b,C)\varphi_{a,d}(a,d)\varphi_{b,\text{mi}}(b,\text{mi})\varphi_{\text{mi},F}(\text{mi},f)$ para una adecuada elección del φ funciones

$y\varphi(X)$ es conocido como el**función de energía**. Porque $\text{Exp}(z)$ es positivo para todos z , esto garantiza que ninguna función de energía dará como resultado una probabilidad de cero para cualquier estado X .Ser completamente libre de elegir la función de energía hace que el aprendizaje sea más sencillo. Si aprendiéramos los potenciales de la camarilla directamente, necesitaríamos usar la optimización restringida para imponer arbitrariamente algún valor de probabilidad mínimo específico. Al aprender la función de energía, podemos usar la optimización sin restricciones.⁵

Las probabilidades en un modelo basado en energía pueden acercarse arbitrariamente a cero pero nunca alcanzarlo.

Cualquier distribución de la forma dada por la ecuación16.7es un ejemplo de un**Distribución de Boltzmann**. Por esta razón, muchos modelos basados en energía se denominan **Máquinas Boltzmann**(Fahlman et al., 1983;Ackley et al., 1985;Hinton et al., 1984;Hinton y Sejnowski, 1986). No existe una directriz aceptada sobre cuándo llamar a un modelo basado en energía y cuándo llamarlo máquina de Boltzmann. El término máquina de Boltzmann se introdujo por primera vez para describir un modelo con variables exclusivamente binarias, pero hoy en día muchos modelos, como la máquina de Boltzmann restringida de covarianza media, también incorporan variables de valor real. Si bien las máquinas de Boltzmann se definieron originalmente para abarcar ambos modelos con y sin variables latentes, el término máquina de Boltzmann se usa hoy en día con mayor frecuencia para designar modelos con variables latentes, mientras que las máquinas de Boltzmann sin variables latentes se denominan más a menudo campos aleatorios de Markov o modelos logarítmicos lineales. .

Las camarillas en un gráfico no dirigido corresponden a factores de la función de probabilidad no normalizada. Porque $\text{Exp}(a)\text{Exp}(b) = \exp(a+b)$, esto significa que diferentes clicas en el gráfico no dirigido corresponden a los diferentes términos de la función de energía. En otras palabras, un modelo basado en energía es solo un tipo especial de red de Markov: la exponenciación hace que cada término en la función de energía corresponda a un factor para una camarilla diferente. Ver figura16.5para ver un ejemplo de cómo leer el

⁵Para algunos modelos, es posible que todavía necesitemos usar optimización restringida para asegurarnos de que existe

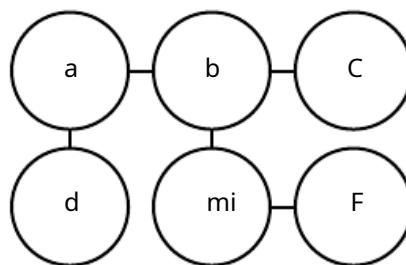


Figura 16.5: Este gráfico implica que $m(a,b,C,d,mi,f)$ se puede escribir como $m_{a,b}(a,b) + m_{b,c}(b,c) + m_{a,d}(a,d) + m_{b,mi}(b,e) + m_{mi,f}(mi,f)$ para una elección apropiada de las funciones de energía por camarilla. Nótese que podemos obtener la φ funciones en la figura 16.4 configurando cada φ a la exponencial de la energía negativa correspondiente, por ejemplo, $\varphi_{a,b}(a,b) = \text{Exp}(-m(a,b))$.

forma de la función de energía a partir de una estructura gráfica no dirigida. Uno puede ver un modelo basado en energía con múltiples términos en su función de energía como un **producto de expertos** (Hinton, 1999). Cada término en la función de energía corresponde a otro factor en la distribución de probabilidad. Cada término de la función de energía se puede considerar como un "experto" que determina si se satisface una restricción suave particular. Cada experto puede hacer cumplir solo una restricción que concierne solo a una proyección de baja dimensión de las variables aleatorias, pero cuando se combinan mediante la multiplicación de probabilidades, los expertos juntos hacen cumplir una restricción complicada de alta dimensión.

Una parte de la definición de un modelo basado en energía no tiene ningún propósito funcional desde el punto de vista del aprendizaje automático: el $-$ iniciar sesión en la ecuación 16.7. Este $-$ signo podría incorporarse a la definición de mi . Para muchas opciones de la función mi , el algoritmo de aprendizaje es libre de determinar el signo de la energía de todos modos. El $-$ signo está presente principalmente para preservar la compatibilidad entre la literatura de aprendizaje automático y la literatura de física. Muchos avances en el modelado probabilístico fueron desarrollados originalmente por físicos estadísticos, para quienes mi se refiere a la energía física real y no tiene un signo arbitrario. Terminología como "energía" y "función de partición" sigue asociada a estas técnicas, aunque su aplicabilidad matemática es más amplia que el contexto físico en el que se desarrollaron. Algunos investigadores de aprendizaje automático (p. ej., Smolensky (1986), quien se refirió a la energía negativa como **armonía**) han optado por emitir la negación, pero esta no es la convención estándar.

Muchos algoritmos que operan en modelos probabilísticos no necesitan calcular $p_{\text{ag}}^{\text{modelo}}(X)$ pero sólo registran $p_{\text{ag}}^{\text{modelo}}(X)$. Para modelos basados en energía con variables latentes h , estos algoritmos a veces se expresan en términos del negativo de esta cantidad,

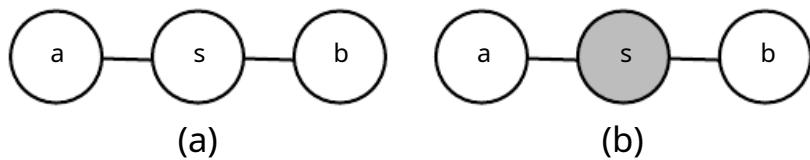


Figura 16.6: (a) El camino entre la variable aleatoria a y la variable aleatoria b a través de s está activo, porque s no se observa. Esto significa que a y b no están separados. (b) Aquí s está sombreado para indicar que se observa. Como el único camino entre a y b es a través de s , y ese camino está inactivo, podemos concluir que a y b están separados dado s .

llamó al **energía gratis**:

$$R(X) = -\text{registro}_h \text{Exp}(-m(x, h)). \quad (16.8)$$

En este libro, por lo general preferimos el más general $\text{registro}_{\text{p}ag_{\text{modelo}}(X)}$ formulación.

16.2.5 Separación y separación D

Los bordes en un modelo gráfico nos dicen qué variables interactúan directamente. A menudo necesitamos saber qué variables *indirectamente* interactúan. Algunas de estas interacciones indirectas se pueden activar o desactivar observando otras variables. Más formalmente, nos gustaría saber qué subconjuntos de variables son condicionalmente independientes entre sí, dados los valores de otros subconjuntos de variables.

Identificar las independencias condicionales en un grafo es muy sencillo en el caso de modelos no dirigidos. En este caso, la independencia condicional implícita en el gráfico se llama **separación**. Decimos que un conjunto de variables **A** es **separado** de otro conjunto de variables **B** dado un tercer conjunto de variables **S** si la estructura del gráfico implica que **A** es independiente de **B** dado **S**. Si dos variables a y b están conectadas por un camino que involucra solo variables no observadas, entonces esas variables no están separadas. Si no existe un camino entre ellos, o si todos los caminos contienen una variable observada, entonces se separan. Nos referimos a las rutas que involucran solo variables no observadas como "activas" y las rutas que incluyen una variable observada como "inactivas".

Cuando dibujamos un gráfico, podemos indicar las variables observadas sombreándolas. Ver figura 16.6 para obtener una descripción de cómo se ven las rutas activas e inactivas en un modelo no dirigido cuando se dibuja de esta manera. Ver figura 16.7 para ver un ejemplo de separación de lectura de un gráfico no dirigido.

Se aplican conceptos similares a los modelos dirigidos, excepto que en el contexto de los modelos dirigidos, estos conceptos se denominan **d-separación**. La "d" significa "dependencia". La separación D para gráficos dirigidos se define igual que la separación

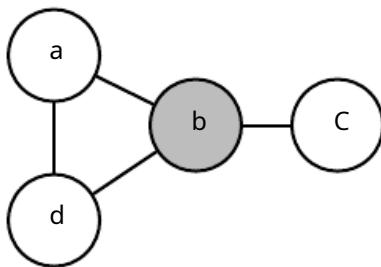


Figura 16.7: Un ejemplo de lectura de propiedades de separación de un gráfico no dirigido. Aquí b está sombreado para indicar que se observa. Debido a que la observación de b bloquea el único camino de a a c, decimos que a y c están separados entre sí dado b. La observación de b también bloquea un camino entre a y d, pero hay un segundo camino activo entre ellos. Por tanto, a y d no están separados dado b.

para grafos no dirigidos: Decimos que un conjunto de variables A está d -separado de otro conjunto de variables B dado un tercer conjunto de variables S si la estructura del gráfico implica que A es independiente de B dado S .

Al igual que con los modelos no dirigidos, podemos examinar las independencias implícitas en el gráfico observando qué caminos activos existen en el gráfico. Como antes, dos variables son dependientes si hay un camino activo entre ellas y están separadas si no existe tal camino. En las redes dirigidas, determinar si un camino está activo es algo más complicado. Ver figura 16.8 para obtener una guía para identificar rutas activas en un modelo dirigido. Ver figura 16.9 para ver un ejemplo de cómo leer algunas propiedades de un gráfico.

Es importante recordar que la separación y la d -separación nos hablan solo de aquellas independencias condicionales que están *implícitas en el gráfico*. No hay ningún requisito de que el gráfico implique todas las independencias que están presentes. En particular, siempre es legítimo usar el gráfico completo (el gráfico con todos los bordes posibles) para representar cualquier distribución. De hecho, algunas distribuciones contienen independencias que no son posibles de representar con la notación gráfica existente.

Independencias específicas del contexto son independencias que están presentes en función del valor de algunas variables en la red. Por ejemplo, considere un modelo de tres variables binarias: a, b y c. Suponga que cuando a es 0, b y c son independientes, pero cuando a es 1, b es determinísticamente igual a c. Codificando el comportamiento cuando a = 1 requiere un borde que conecte b y c. Entonces, la gráfica no indica que b y c son independientes cuando a = 0.

En general, un gráfico nunca implicará que existe una independencia cuando no es así. Sin embargo, un gráfico puede no codificar una independencia.

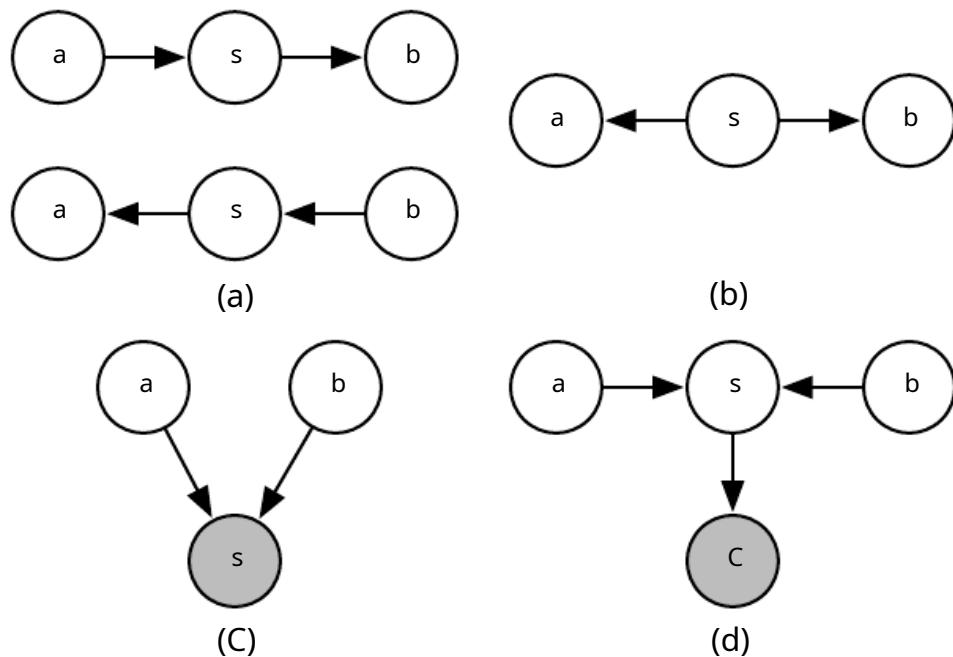


Figura 16.8: Todos los tipos de caminos activos de longitud dos que pueden existir entre las variables aleatorias a y b. (a) Cualquier camino con flechas que procedan directamente de a a b o viceversa. Este tipo de camino se bloquea si se observa s. Ya hemos visto este tipo de camino en el ejemplo de la carrera de relevos. (b) a y b están conectados por *a causa comunes*. Por ejemplo, supongamos que s es una variable que indica si hay un huracán o no y que a y b miden la velocidad del viento en dos puestos de monitoreo meteorológico cercanos diferentes. Si observamos vientos muy fuertes en la estación a, podríamos esperar ver también vientos fuertes en b. Este tipo de camino puede bloquearse observando s. Si ya sabemos que hay un huracán, esperamos ver vientos fuertes en b, independientemente de lo que se observe en a. Un viento más bajo de lo esperado en a (para un huracán) no cambiaría nuestra expectativa de vientos en b (sabiendo que hay un huracán). Sin embargo, si no se observa s, entonces a y b son dependientes, es decir, el camino está activo. (c) a y b son padres de s. Esto se llama **un estructura en V o el caso del colisionador**. La estructura en V hace que a y b estén relacionados por la **explicando el efecto de distancia**. En este caso, la ruta está realmente activa cuando se observa s. Por ejemplo, suponga que s es una variable que indica que su colega no está en el trabajo. La variable a representa que está enferma, mientras que b representa que está de vacaciones. Si observa que ella no está en el trabajo, puede suponer que probablemente esté enferma o de vacaciones, pero no es especialmente probable que ambas cosas hayan sucedido al mismo tiempo. Si te enteras de que ella está de vacaciones, este hecho es suficiente para explicar su ausencia. Puede inferir que probablemente ella no esté también enferma. (d) El efecto de explicación ocurre incluso si se observa cualquier descendiente de s! Por ejemplo, suponga que c es una variable que representa si recibió un informe de su colega. Si nota que no ha recibido el informe, esto aumenta su estimación de la probabilidad de que ella no esté en el trabajo hoy, lo que a su vez aumenta la probabilidad de que esté enferma o de vacaciones. La única forma de bloquear un camino a través de una estructura en V es no observar a ninguno de los descendientes del hijo compartido.

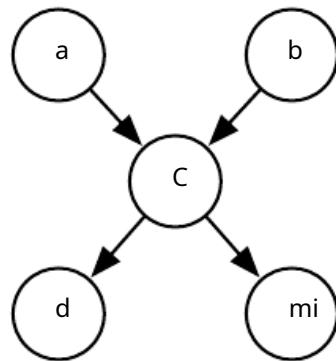


Figura 16.9: A partir de este gráfico, podemos leer varias propiedades de separación d. Ejemplos incluyen:

- a y b están d-separados dado el conjunto vacío.
- a y e están separados por d dado c.
- d y e están separados por d dado c.

También podemos ver que algunas variables ya no están separadas cuando observamos algunas variables:

- a y b no están separados por d dado c.
- a y b no están separados por d dado d.

16.2.6 Conversión entre gráficos dirigidos y no dirigidos

A menudo nos referimos a un modelo de aprendizaje automático específico como no dirigido o dirigido. Por ejemplo, normalmente nos referimos a los RBM como codificación no dirigida y escasa como dirigida. Esta elección de redacción puede ser un tanto engañosa, porque ningún modelo probabilístico es intrínsecamente dirigido o no dirigido. En cambio, algunos modelos son más fáciles *descrito* usando un gráfico dirigido, o describiéndolo más fácilmente usando un gráfico no dirigido.

Los modelos dirigidos y los modelos no dirigidos tienen sus ventajas y desventajas. Ninguno de los enfoques es claramente superior y universalmente preferido. En cambio, debemos elegir qué idioma usar para cada tarea. Esta elección dependerá en parte de qué distribución de probabilidad deseemos describir. Podemos optar por utilizar un modelo dirigido o un modelo no dirigido en función de qué enfoque puede capturar la mayor cantidad de independencias en la distribución de probabilidad o qué enfoque usa la menor cantidad de bordes para describir la distribución. Hay otros factores que pueden afectar la decisión de qué idioma usar. Incluso mientras trabajamos con una sola distribución de probabilidad, a veces podemos cambiar entre diferentes lenguajes de modelado. A veces, un idioma diferente se vuelve más apropiado si observamos un cierto subconjunto de variables, o si deseamos realizar una tarea computacional diferente. Por ejemplo, la descripción del modelo dirigido a menudo proporciona un enfoque directo para extraer muestras del modelo de manera eficiente (descrito en la sección 16.3) mientras que la formulación del modelo no dirigido suele ser útil para derivar procedimientos de inferencia aproximada (como veremos en el capítulo 19, donde el papel de los modelos no dirigidos se destaca en la ecuación 19.56).

Toda distribución de probabilidad puede representarse mediante un modelo dirigido o mediante un modelo no dirigido. En el peor de los casos, siempre se puede representar cualquier distribución usando un “gráfico completo”. En el caso de un modelo dirigido, el gráfico completo es cualquier gráfico acíclico dirigido en el que imponemos algún orden en las variables aleatorias, y cada variable tiene todas las demás variables que la preceden en el orden como sus ancestros en el gráfico. Para un modelo no dirigido, el gráfico completo es simplemente un gráfico que contiene una sola camarilla que abarca todas las variables. Ver figura 16.10 para un ejemplo.

Por supuesto, la utilidad de un modelo gráfico es que el gráfico implica que algunas variables no interactúan directamente. El gráfico completo no es muy útil porque no implica independencias.

Cuando representamos una distribución de probabilidad con un gráfico, queremos elegir un gráfico que implique tantas independencias como sea posible, sin implicar ninguna independencia que en realidad no exista.

Desde este punto de vista, algunas distribuciones se pueden representar de manera más eficiente

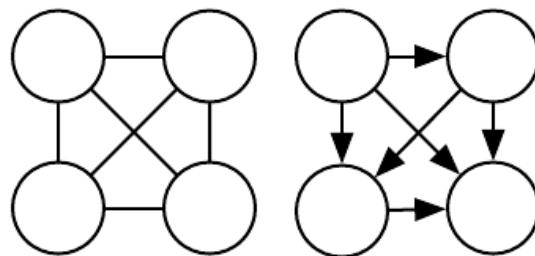


Figura 16.10: Ejemplos de gráficos completos, que pueden describir cualquier distribución de probabilidad. Aquí mostramos ejemplos con cuatro variables aleatorias. (Izquierda) El grafo no dirigido completo. En el caso no dirigido, el grafo completo es único. (Derecha) Un grafo dirigido completo. En el caso dirigido, no hay un único grafo completo. Elegimos un orden de las variables y dibujamos un arco desde cada variable hasta cada variable que viene después en el orden. Por lo tanto, hay un número factorial de gráficos completos para cada conjunto de variables aleatorias. En este ejemplo, ordenamos las variables de izquierda a derecha y de arriba a abajo.

usando modelos dirigidos, mientras que otras distribuciones se pueden representar de manera más eficiente usando modelos no dirigidos. En otras palabras, los modelos dirigidos pueden codificar algunas independencias que los modelos no dirigidos no pueden codificar y viceversa.

Los modelos dirigidos pueden usar un tipo específico de subestructura que los modelos no dirigidos no pueden representar perfectamente. Esta subestructura se denomina **inmoralidad**. La estructura ocurre cuando dos variables aleatorias a y b son padres de una tercera variable aleatoria c , y no hay un borde que conecte directamente a y b en ninguna dirección. (El nombre “inmoralidad” puede parecer extraño; fue acuñado en la literatura de modelos gráficos como una broma sobre los padres solteros). Para convertir un modelo dirigido con gráfico D en un modelo no dirigido, necesitamos crear un nuevo gráfico t . Para cada par de variables x e y , agregamos un borde no dirigido que conecta x e y a t . Si x e y son ambos padres en D de una tercera variable z . La resultante t es conocido como un **gráfico moralizado**. Ver figura 16.11 para ejemplos de conversión de modelos dirigidos a modelos no dirigidos a través de la moralización.

Asimismo, los modelos no dirigidos pueden incluir subestructuras que ningún modelo dirigido puede representar perfectamente. Específicamente, un gráfico dirigido D no puede capturar todas las independencias condicionales implícitas en un gráfico no dirigido t . Si t contiene un **bucle** de longitud superior a tres, a menos que ese bucle también contenga un **acorde**. Un bucle es una secuencia de variables conectadas por aristas no dirigidas, con la última variable de la secuencia conectada de nuevo a la primera variable de la secuencia. Un acorde es una conexión entre dos variables no consecutivas en la secuencia que define un bucle. Si t tiene bucles de cuatro o más longitudes y no tiene cuerdas para estos bucles, debemos agregar las cuerdas antes de poder convertirlo en un modelo dirigido. agregando

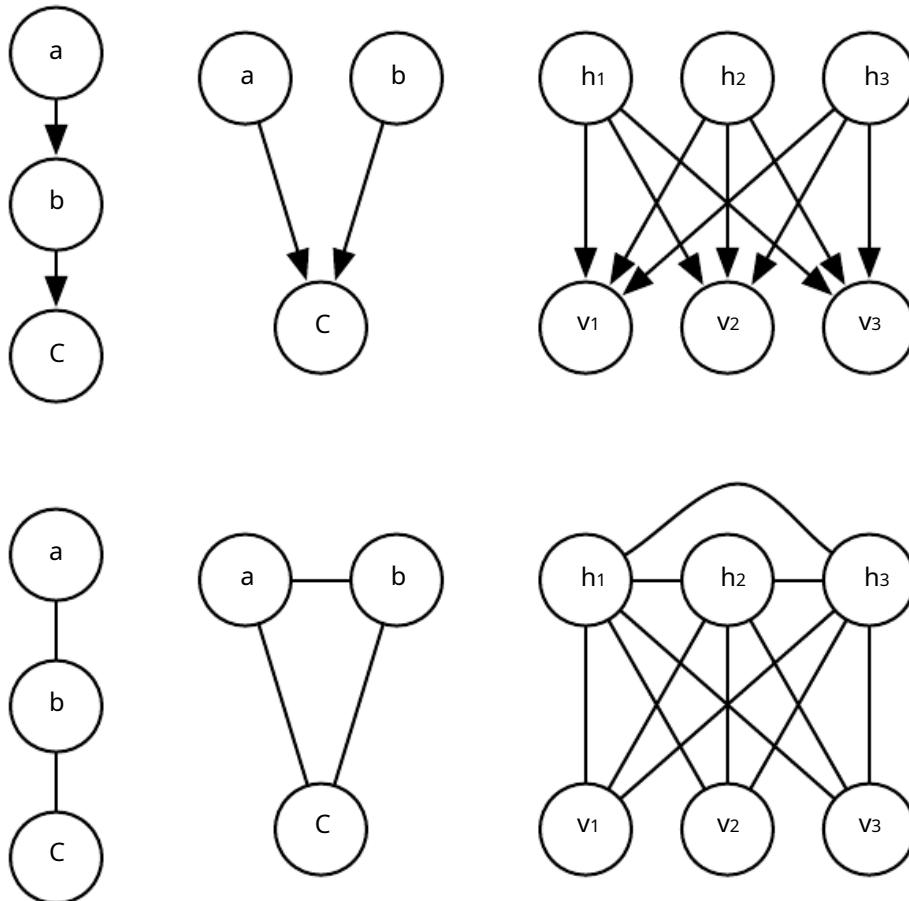


Figura 16.11: Ejemplos de conversión de modelos dirigidos (fila superior) a modelos no dirigidos (fila inferior) mediante la construcción de gráficos moralizados. (Izquierda) Esta cadena simple se puede convertir en un gráfico moralizado simplemente reemplazando sus bordes dirigidos con bordes no dirigidos. El modelo no dirigido resultante implica exactamente el mismo conjunto de independencias e independencias condicionales. (Centro) Este gráfico es el modelo dirigido más simple que no se puede convertir en un modelo no dirigido sin perder algunas independencias. Este gráfico consiste enteramente en una sola inmoralidad. Como a y b son padres de c , están conectados por un camino activo cuando se observa c . Para capturar esta dependencia, el modelo no dirigido debe incluir una camarilla que abarque las tres variables. Esta camarilla no logra codificar el hecho de que $a \perp\!\!\!\perp b$. (Bien) En general, la moralización puede agregar muchos bordes al gráfico, perdiendo así muchas independencias implícitas. Por ejemplo, este gráfico de codificación dispersa requiere agregar bordes moralizantes entre cada par de unidades ocultas, introduciendo así un número cuadrático de nuevas dependencias directas.

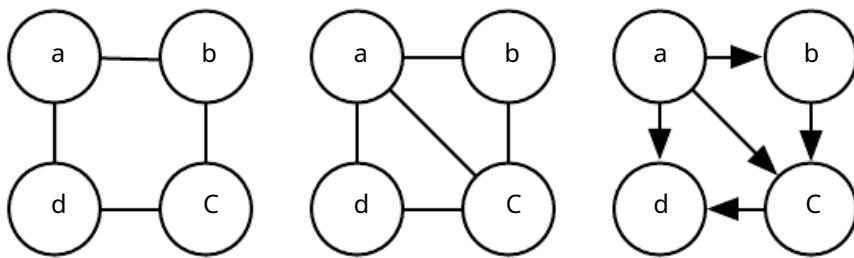


Figura 16.12: Conversión de un modelo no dirigido a un modelo dirigido.(Izquierda)Este modelo no dirigido no se puede convertir dirigido en un modelo dirigido porque tiene un bucle de cuatro longitudes sin cuerdas. Específicamente, el modelo no dirigido codifica dos independencias diferentes que ningún modelo dirigido puede capturar simultáneamente: una $A \perp\!\!\!\perp C / \{b,d\}$ y $B \perp\!\!\!\perp d / \{a,C\}$.(Centro)Para convertir el modelo no dirigido en un modelo dirigido, debemos triangular el gráfico, asegurándonos de que todos los bucles de longitud superior a tres tengan una cuerda. Para hacerlo, podemos agregar un borde que conecte a y c o podemos agregar un borde que conecte b y d. En este ejemplo, elegimos agregar el borde que conecta a y c.(Bien)Para finalizar el proceso de conversión, debemos asignar una dirección a cada borde. Al hacerlo, no debemos crear ningún ciclo dirigido. Una forma de evitar los ciclos dirigidos es imponer un orden sobre los nodos, y siempre apuntar cada borde desde el nodo que viene antes en el orden hasta el nodo que viene después en el orden. En este ejemplo, usamos los nombres de las variables para imponer un orden alfabético.

estos acordes descarta parte de la información de independencia que fue codificada en t_u . El gráfico formado al sumar cuerdas $a \perp\!\!\!\perp u$ es conocido como **uncordalotriangulado** gráfico, porque ahora todos los bucles se pueden describir en términos de bucles triangulares más pequeños. Para construir un gráfico dirigido D del gráfico cordal, también necesitamos asignar direcciones a los bordes. Al hacerlo, no debemos crear un ciclo dirigido en D , o el resultado no define un modelo probabilístico dirigido válido. Una forma de asignar direcciones a los bordes en D es imponer un orden en las variables aleatorias, luego apuntar cada borde desde el nodo que viene antes en el orden hasta el nodo que viene después en el orden. Ver figura 16.12para una demostración.

16.2.7 Gráficos de factores

Gráficos de factoresson otra forma de dibujar modelos no dirigidos que resuelven una ambigüedad en la representación gráfica de la sintaxis del modelo no dirigido estándar. En un modelo no dirigido, el alcance de cada φ la función debe ser una **subconjunto**de alguna camarilla en el gráfico. La ambigüedad surge porque no está claro si cada camarilla en realidad tiene un factor correspondiente cuyo alcance abarca toda la camarilla; por ejemplo, una camarilla que contiene tres nodos puede corresponder a un factor sobre los tres nodos, o puede corresponder a tres factores que contienen cada uno solo un par de nodos.

Los gráficos de factores resuelven esta ambigüedad al representar explícitamente el alcance de cada función. Específicamente, un gráfico de factores es una representación gráfica de un modelo no dirigido que consta de un gráfico bipartito no dirigido. Algunos de los nodos se dibujan como círculos. Estos nodos corresponden a variables aleatorias como en un modelo estándar no dirigido. El resto de los nodos se dibujan como cuadrados. Estos nodos corresponden a los factores y de la distribución de probabilidad no normalizada. Las variables y los factores pueden estar conectados con aristas no dirigidas. Una variable y un factor están conectados en el gráfico si y solo si la variable es uno de los argumentos del factor en la distribución de probabilidad no normalizada. Ningún factor puede estar conectado a otro factor en el gráfico, ni una variable puede estar conectada a una variable. Ver figura 16.13 para ver un ejemplo de cómo los gráficos de factores pueden resolver la ambigüedad en la interpretación de redes no dirigidas.

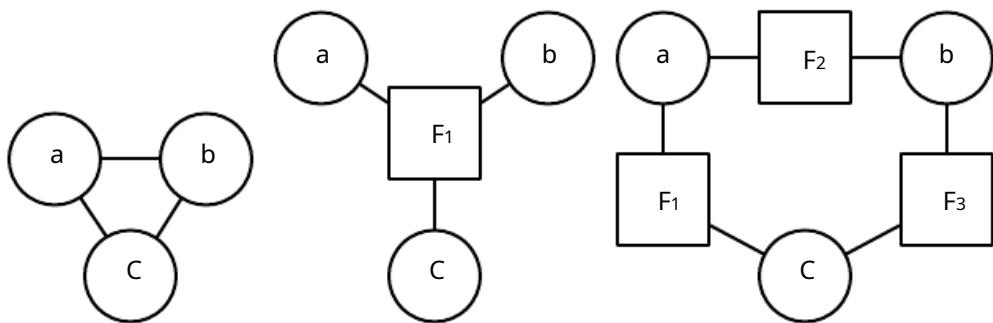


Figura 16.13: Un ejemplo de cómo un gráfico de factores puede resolver la ambigüedad en la interpretación de redes no dirigidas. (Izquierda) Una red no dirigida con una camarilla que involucra tres variables: a, b y c. (Centro) Un gráfico de factores correspondiente al mismo modelo no dirigido. Este gráfico de factores tiene un factor sobre las tres variables. (Bien) Otro gráfico factorial válido para el mismo modelo no dirigido. Este gráfico de factores tiene tres factores, cada uno sobre solo dos variables. La representación, la inferencia y el aprendizaje son todos asintóticamente más baratos en este gráfico de factores que en el gráfico de factores representado en el centro, aunque ambos requieren el mismo gráfico no dirigido para su representación.

16.3 Muestreo a partir de modelos gráficos

Los modelos gráficos también facilitan la tarea de extraer muestras de un modelo.

Una ventaja de los modelos gráficos dirigidos es que un procedimiento simple y eficiente llamado **muestreo ancestral** puede producir una muestra de la distribución conjunta representada por el modelo.

La idea básica es ordenar las variables x_i en el gráfico en un ordenamiento topológico, de modo que para todos i, j , si x_j es mayor que si x_i es padre de x_j , las variables

luego se pueden muestrear en este orden. En otras palabras, primero muestreamos $x_1 \sim PAG(X_1)$, luego muestra $PAG(X_2 / Pensilvaniagramo(X_2))$, y así sucesivamente, hasta que finalmente tomamos una muestra $PAG(X_{norte} / Pensilvaniagramo(X_{norte}))$. Siempre que cada distribución condicional $pag(X_i / Pensilvaniagramo(X_i))$ es fácil de muestrear, entonces todo el modelo es fácil de muestrear. La operación de clasificación topológica garantiza que podemos leer las distribuciones condicionales en la ecuación 16.1 y muestra de ellos en orden. Sin la ordenación topológica, podríamos intentar muestrear una variable antes de que sus padres estén disponibles.

Para algunos gráficos, es posible más de un ordenamiento topológico. El muestreo ancestral se puede utilizar con cualquiera de estos ordenamientos topológicos.

El muestreo ancestral es generalmente muy rápido (suponiendo que el muestreo de cada condicional sea fácil) y conveniente.

Un inconveniente del muestreo ancestral es que solo se aplica a modelos gráficos dirigidos. Otro inconveniente es que no admite todas las operaciones de muestreo condicional. Cuando deseamos muestrear un subconjunto de variables en un modelo gráfico dirigido, dadas algunas otras variables, a menudo requerimos que todas las variables condicionantes estén antes que las variables que se muestrearan en el gráfico ordenado. En este caso, podemos tomar muestras de las distribuciones de probabilidad condicional locales especificadas por la distribución del modelo. De lo contrario, las distribuciones condicionales de las que necesitamos muestrear son las distribuciones posteriores dadas las variables observadas. Estas distribuciones posteriores generalmente no se especifican ni parametrizan explícitamente en el modelo. Inferir estas distribuciones posteriores puede ser costoso. En los modelos donde este es el caso,

Desafortunadamente, el muestreo ancestral es aplicable solo a modelos dirigidos. Podemos tomar muestras de modelos no dirigidos convirtiéndolos en modelos dirigidos, pero esto a menudo requiere resolver problemas de inferencia intratables (para determinar la distribución marginal sobre los nodos raíz del nuevo gráfico dirigido) o requiere introducir tantos bordes que el modelo dirigido resultante se vuelve intratable. . El muestreo de un modelo no dirigido sin convertirlo primero en un modelo dirigido parece requerir la resolución de dependencias cíclicas. Cada variable interactúa con todas las demás variables, por lo que no hay un punto de inicio claro para el proceso de muestreo. Desafortunadamente, extraer muestras de un modelo gráfico no dirigido es un proceso costoso de varios pasos. El enfoque conceptualmente más simple es

Muestreo de Gibbs. Supongamos que tenemos un modelo gráfico sobre un *norte*-vector dimensional de variables aleatorias X . Visitamos iterativamente cada variable x_i y extraiga una muestra condicionada a todas las demás variables, $depag(X_i / X_{-i})$. Debido a las propiedades de separación del modelo gráfico, podemos condicionar de manera equivalente solo a los vecinos de x_i . Desafortunadamente, después de haber hecho una pasada por el modelo gráfico y muestreado todos *norte* variables, todavía no tenemos una muestra justa $depag(X)$. En su lugar, debemos repetir el

procesar y volver a muestrear todo *n* variables usando los valores actualizados de sus vecinos. Asintóticamente, después de muchas repeticiones, este proceso converge al muestreo de la distribución correcta. Puede ser difícil determinar cuándo las muestras han alcanzado una aproximación suficientemente precisa de la distribución deseada. Las técnicas de muestreo para modelos no dirigidos son un tema avanzado, cubierto con más detalle en el capítulo 17.

16.4 Ventajas del modelado estructurado

La principal ventaja de usar modelos probabilísticos estructurados es que nos permiten reducir drásticamente el costo de representar distribuciones de probabilidad, así como el aprendizaje y la inferencia. El muestreo también se acelera en el caso de modelos dirigidos, mientras que la situación puede complicarse con modelos no dirigidos. El mecanismo principal que permite que todas estas operaciones usen menos tiempo de ejecución y memoria es elegir no modelar ciertas interacciones. Los modelos gráficos transmiten información dejando fuera los bordes. Dondequiera que no haya un borde, el modelo especifica la suposición de que no necesitamos modelar una interacción directa.

Un beneficio menos cuantificable del uso de modelos probabilísticos estructurados es que nos permiten separar explícitamente la representación del conocimiento del aprendizaje del conocimiento o la inferencia dado el conocimiento existente. Esto hace que nuestros modelos sean más fáciles de desarrollar y depurar. Podemos diseñar, analizar y evaluar algoritmos de aprendizaje y algoritmos de inferencia que son aplicables a amplias clases de gráficos. Independientemente, podemos diseñar modelos que capturen las relaciones que creemos que son importantes en nuestros datos. Entonces podemos combinar estos diferentes algoritmos y estructuras y obtener un producto cartesiano de diferentes posibilidades. Sería mucho más difícil diseñar algoritmos de extremo a extremo para cada situación posible.

16.5 Aprender acerca de las dependencias

Un buen modelo generativo necesita capturar con precisión la distribución sobre las variables observadas o "visibles". Con frecuencia, los diferentes elementos devson altamente dependientes unos de otros. En el contexto del aprendizaje profundo, el enfoque más comúnmente utilizado para modelar estas dependencias es introducir varias variables latentes u "ocultas", H . El modelo puede entonces capturar dependencias entre cualquier par de variables v_i y V_j indirectamente, a través de dependencias directas entre v_i y h_j , y dependencias directas entre h_i y V_j .

un buen modelo devque no contuviera ninguna variable latente tendría que

tener un gran número de padres por nodo en una red bayesiana o camarillas muy grandes en una red de Markov. Solo representar estas interacciones de orden superior es costoso, tanto en un sentido computacional, porque la cantidad de parámetros que deben almacenarse en la memoria aumenta exponencialmente con la cantidad de miembros en una camarilla, pero también en un sentido estadístico, porque esta cantidad exponencial de parámetros requiere una gran cantidad de datos para estimar con precisión.

Cuando el modelo pretende capturar dependencias entre variables visibles con conexiones directas, generalmente no es factible conectar todas las variables, por lo que el gráfico debe diseñarse para conectar aquellas variables que están estrechamente acopladas y omitir los bordes entre otras variables. Todo un campo de aprendizaje automático llamado **estructura de aprendizaje** está dedicado a este problema. Para una buena referencia sobre el aprendizaje de estructuras, consulte ([Koller y Friedman, 2009](#)). La mayoría de las técnicas de aprendizaje estructurado son una forma de búsqueda codiciosa. Se propone una estructura, se entrena un modelo con esa estructura y luego se le asigna una puntuación. La puntuación premia la alta precisión del conjunto de entrenamiento y penaliza la complejidad del modelo. Las estructuras candidatas con un pequeño número de bordes agregados o eliminados se proponen como el siguiente paso de la búsqueda. La búsqueda procede a una nueva estructura que se espera que aumente la puntuación.

El uso de variables latentes en lugar de estructuras adaptativas evita la necesidad de realizar búsquedas discretas y múltiples rondas de entrenamiento. Una estructura fija sobre variables visibles y ocultas puede usar interacciones directas entre unidades visibles y ocultas para imponer interacciones indirectas entre unidades visibles. Usando técnicas simples de aprendizaje de parámetros, podemos aprender un modelo con una estructura fija que imputa la estructura correcta en el margen $\text{pag}(v)$.

Las variables latentes tienen ventajas más allá de su papel en la captura eficiente $\text{pag}(v)$. Las nuevas variables también proporcionan una representación alternativa para v . Por ejemplo, como se discutió en la sección [3.9.6](#), el modelo de mezcla de gaussianas aprende una variable latente que corresponde a la categoría de ejemplos de la que se extrajo la entrada. Esto significa que la variable latente en un modelo mixto de gaussianas se puede utilizar para realizar la clasificación. En el capítulo [14](#) vimos cómo los modelos probabilísticos simples, como la codificación dispersa, aprenden variables latentes que se pueden usar como características de entrada para un clasificador o como coordenadas a lo largo de una variedad. Se pueden usar otros modelos de la misma manera, pero los modelos más profundos y los modelos con diferentes tipos de interacciones pueden crear descripciones aún más ricas de la entrada. Muchos enfoques logran el aprendizaje de características mediante el aprendizaje de variables latentes. A menudo, dado algún modelo h , las observaciones experimentales muestran que $\text{MI}[h / v]$ o $\text{argmax}_h \text{pag}(h, v)$ es un buen mapeo de características para v .

16.6 Inferencia e inferencia aproximada

Una de las principales formas en que podemos usar un modelo probabilístico es hacer preguntas sobre cómo se relacionan las variables entre sí. Dado un conjunto de pruebas médicas, podemos preguntar qué enfermedad podría tener un paciente. En un modelo de variable latente, podríamos querer extraer características MI[h / v] describiendo las variables observadas. A veces necesitamos resolver tales problemas para poder realizar otras tareas. A menudo entrenamos nuestros modelos utilizando el principio de máxima verosimilitud. Porque

$$\text{registro}_{\text{pag}}(v) = \min_{h \sim \text{pag}(h/v)} [\text{registro}_{\text{pag}}(h, v) - \text{registro}_{\text{pag}}(h / v)], \quad (16.9)$$

a menudo queremos calcular $\text{pag}(h/v)$ para implementar una regla de aprendizaje. Todos estos son ejemplos de **inferencia** problemas en los que debemos predecir el valor de algunas variables dado el valor de otras variables, o predecir la distribución de probabilidad sobre algunas variables dado el valor de otras variables.

Desafortunadamente, para la mayoría de los modelos profundos interesantes, estos problemas de inferencia son intratables, incluso cuando usamos un modelo gráfico estructurado para simplificarlos. La estructura del gráfico nos permite representar distribuciones complicadas y de alta dimensión con una cantidad razonable de parámetros, pero los gráficos utilizados para el aprendizaje profundo generalmente no son lo suficientemente restrictivos como para permitir también una inferencia eficiente.

Es sencillo ver que calcular la probabilidad marginal de un modelo gráfico general es #P difícil. La clase de complejidad #P es una generalización de la clase de complejidad NP. Los problemas en NP requieren determinar solo si un problema tiene una solución y encontrar una solución si existe. Los problemas en #P requieren contar el número de soluciones. Para construir un modelo gráfico del peor de los casos, imagine que definimos un modelo gráfico sobre las variables binarias en un problema 3-SAT. Podemos imponer una distribución uniforme sobre estas variables. Luego podemos agregar una variable latente binaria por cláusula que indica si se cumple cada cláusula. Luego podemos agregar otra variable latente que indique si se cumplen todas las cláusulas. Esto se puede hacer sin hacer una gran camarilla, mediante la construcción de un árbol de reducción de variables latentes, con cada nodo en el árbol informando si se cumplen otras dos variables. Las hojas de este árbol son las variables de cada cláusula. La raíz del árbol informa si se satisface todo el problema. Debido a la distribución uniforme sobre los literales, la distribución marginal sobre la raíz del árbol de reducción especifica qué fracción de asignaciones satisfacen el problema. Si bien este es un ejemplo artificial del peor de los casos, los gráficos duros de NP comúnmente surgen en escenarios prácticos del mundo real.

Esto motiva el uso de la inferencia aproximada. En el contexto del aprendizaje profundo, esto generalmente se refiere a la inferencia variacional, en la que aproximamos el

distribución verdadera $p_{\text{true}}(h/v)$ buscando una distribución aproximada $q(h/v)$ que es lo más cercano posible al verdadero. Esta y otras técnicas se describen en profundidad en el capítulo 19.

16.7 El enfoque de aprendizaje profundo para modelos probabilísticos estructurados

Los profesionales del aprendizaje profundo generalmente usan las mismas herramientas computacionales básicas que otros profesionales del aprendizaje automático que trabajan con modelos probabilísticos estructurados. Sin embargo, en el contexto del aprendizaje profundo, generalmente tomamos diferentes decisiones de diseño sobre cómo combinar estas herramientas, lo que da como resultado algoritmos y modelos generales que tienen un sabor muy diferente de los modelos gráficos más tradicionales.

El aprendizaje profundo no siempre involucra modelos gráficos especialmente profundos. En el contexto de los modelos gráficos, podemos definir la profundidad de un modelo en términos del gráfico del modelo gráfico en lugar del gráfico computacional. Podemos pensar en una variable latente h como estar en profundidad j si el camino más corto desde h a una variable observada es j pasos. Por lo general, describimos la profundidad del modelo como la mayor profundidad de cualquiera de tales h_i . Este tipo de profundidad es diferente de la profundidad inducida por el gráfico computacional. Muchos modelos generativos utilizados para el aprendizaje profundo no tienen variables latentes o solo una capa de variables latentes, pero usan gráficos computacionales profundos para definir las distribuciones condicionales dentro de un modelo.

El aprendizaje profundo esencialmente siempre hace uso de la idea de representaciones distribuidas. Incluso los modelos superficiales utilizados con fines de aprendizaje profundo (como modelos superficiales de preentrenamiento que luego se compondrán para formar modelos profundos) casi siempre tienen una sola capa grande de variables latentes. Los modelos de aprendizaje profundo suelen tener más variables latentes que variables observadas. Las interacciones no lineales complicadas entre variables se logran a través de conexiones indirectas que fluyen a través de múltiples variables latentes.

Por el contrario, los modelos gráficos tradicionales suelen contener principalmente variables que se observan al menos ocasionalmente, incluso si muchas de las variables faltan al azar en algunos ejemplos de entrenamiento. Los modelos tradicionales utilizan principalmente términos de orden superior y estructuran el aprendizaje para capturar interacciones no lineales complicadas entre variables. Si hay variables latentes, por lo general son pocas.

La forma en que se diseñan las variables latentes también difiere en el aprendizaje profundo. El practicante de aprendizaje profundo generalmente no tiene la intención de que las variables latentes adopten una semántica específica antes de tiempo: el algoritmo de entrenamiento es libre de inventar los conceptos que necesita para modelar un conjunto de datos en particular. Las variables latentes son