Autoencoders

Aprendizaje profundo

Alberto Díaz Álvarez, Edgar Talavera Muñoz y Guillermo Iglesias Hernández
Departamento de Sistemas Informáticos - Universidad Politécnica de Madrid

17 de abril de 2023

License CC BY-NC-SA 4.0

¿Qué son los *autoencoders*?

Son una técnica de reducción de dimensionalidad y compresión de los datos

- Su objetivo es aprender una representación compacta de los datos de entrada

Para funcionar los autoencoders constan de dos partes que **se entrenan a la vez**:

- **Codificador**: Transforma la entrada en una representación de menor dimensión
- **Decodificador**: Transforma la codificación en una aproximación de la entrada

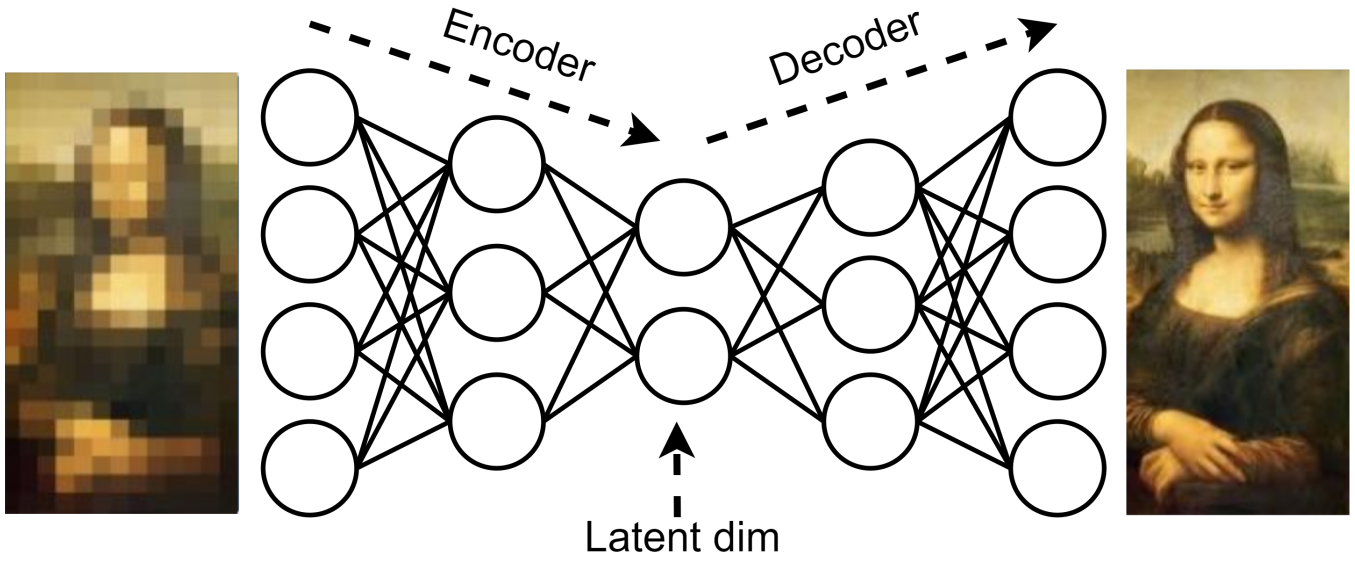
Son muy útiles en una gran variedad de aplicaciones:

- Reducción de la dimensionalidad de datos de alta dimensión
- Eliminación de ruido de señales
- Generación de datos nuevos y similares a los de entrada[1]

El propio Ian GoodFellow menciona que son la primera **red generadora**

Funcionamiento

Un autoencoder no difiere demasiado de un Perceptron multicapa (MLP)



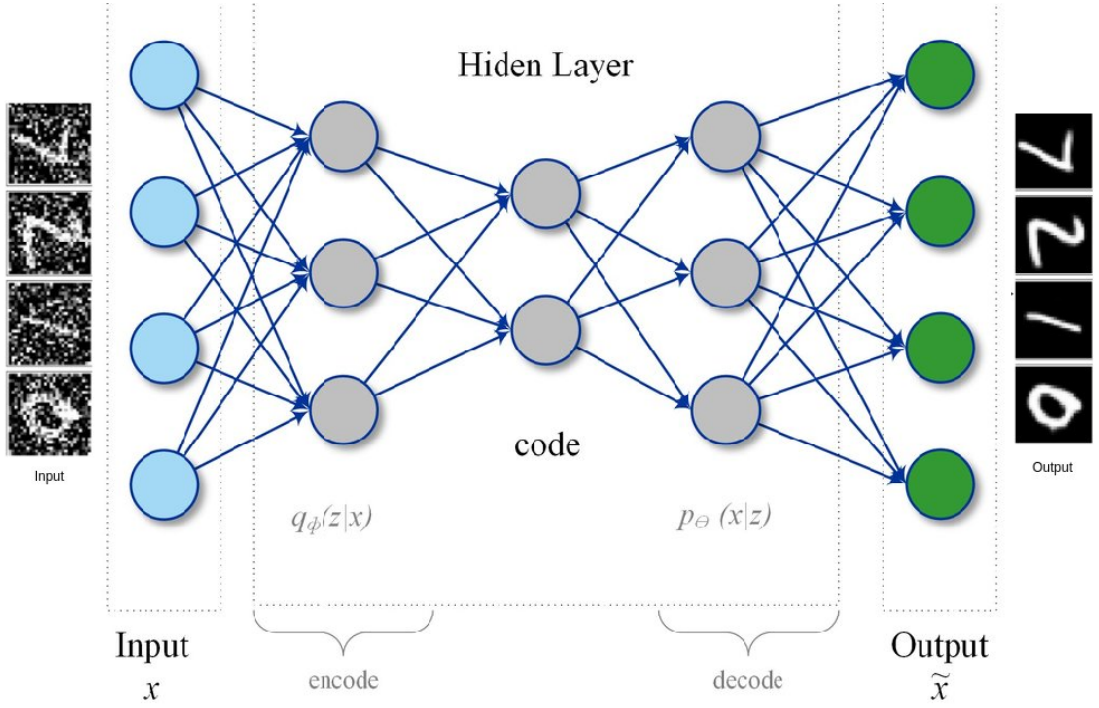
Su entrenamiento se realiza mediante descenso del gradiente

- El backpropagation ajustará la salida de la red lo máximo a la entrada
- El entrenamiento tenderá a eliminar los datos que menos aportan a la salida

Encoder y decoder

Como ya hemos hablado, este tipo de redes consta de dos partes:

- **Encoder**: Comprime la entrada a un espacio latente de menor dimensión
- **Decoder**: Reconstruye la salida a partir del espacio latente



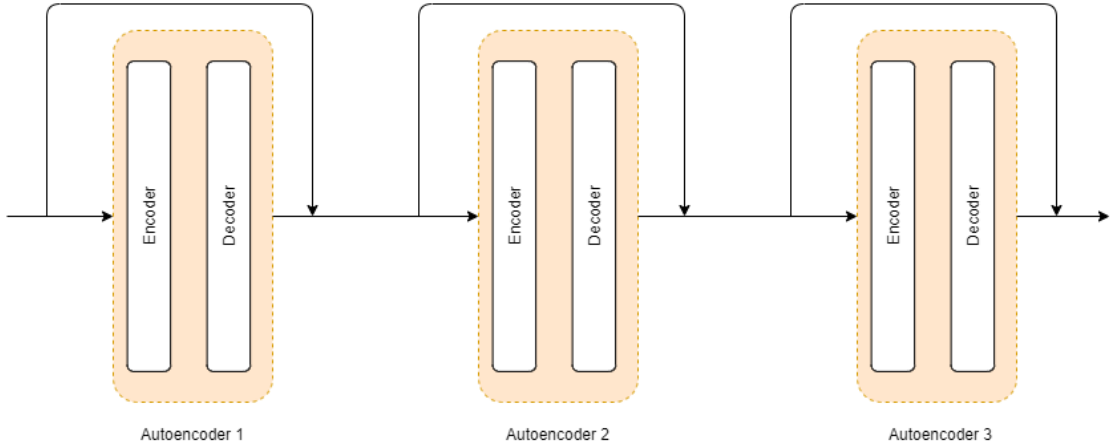
Ejemplo de *autoencoders*

Notebook: `Trabajando con autoencoders.ipynb`

Stacked Autoencoders (SAE)

En sí el espacio latente creado es una reducción de dimensionalidad

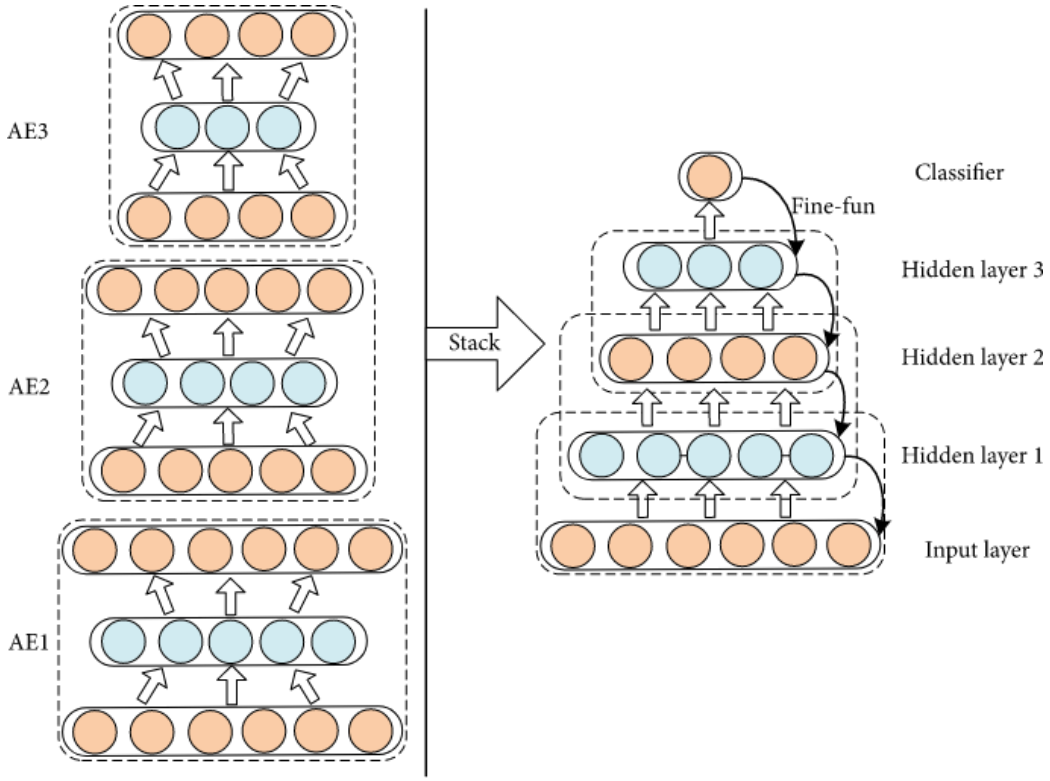
- Pero a veces los datasets son demasiados complejos
- Y a veces un sólo *autoencoder* no es capaz de realizar correctamente el proceso
- Una solución a esto es el uso de autoencoders apilados para mejorar los resultados



SAE como método de entrenamiento (I)

Los SAE de hecho se pueden usar como técnica para entrenar MLP

- Usar un autoencoder para entrenar cada capa con su propia entrada
- Así cada capa aprende características relevantes de la entrada



SAE como método de entrenamiento (y II)

La estructura de la imagen anterior se puede dividir en tres pasos:

1. Se entrena el primer autoencoder (AE1) y se obtiene el vector de características
2. Este vector de características se utiliza como entrada para el siguiente modelo (AE2), y este procedimiento se repite.
3. Una vez entrenadas todas las capas ocultas, se apilan los modelos

Tras este proceso, disponemos de un MLP cuyas capas contienen reducciones dimensionales de las características de entrada

- En este punto, el MLP se puede entrenar para minimizar la función de coste
- Se apoya en la suposición de que las primeras capas de una red aprenden las características de las entradas

Esta fue una de las primeras técnicas para entrenar redes neuronales profundas

Limitaciones de los *autoencoders*

Los *autoencoders* tienen mucho potencia, pero en la generación de datos sintéticos no son muy eficientes

El **principal problema**: El espacio latente generado **no es continuo**

- Está formado por regiones separadas unas de otras que agrupan las características de ejemplos similares
- Entre estas regiones no hay un espacio continuo de representaciones intermedias
  - En realidad sí, pero no tiene sentido
  - ¿De verdad no hay un espacio intermedio entre un 1 y un 7? ¿o entre un 3 y un 8?
- Esto hace que la interpolación entre ejemplos sea imposible

En definitiva, **si el espacio intermedio no es continuo, las salidas del decoder no son realistas**

Variational autoencoders (VAE)

Son una variante de los autoencoders que permiten generar datos sintéticos

- Mezclan las redes neuronales con distribuciones de probabilidad
- Permiten que los datos generados sigan el mismo patrón de los datos de entrada

Así la red aprende los parámetros de una distribución de probabilidad

- Así construyen explícitamente un **espacio latente continuo**
- No una función arbitraria como pasa en redes neuronales convencionales

VAE - Funcionamiento

En los VAE el espacio latente se define por dos vectores de tamaño n:

- $\vec{\mu} = (\mu_1, \ldots, \mu_n)$: Vector de medias
- $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$: Vector de desviaciones típicas

Forman un vector de distribuciones normales: $(N(\mu_1, \sigma_1), \ldots, N(\mu_n, \sigma_n))$

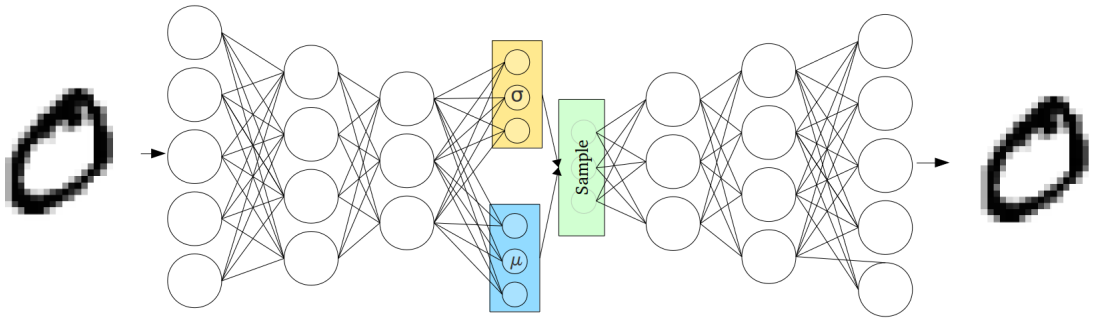
- Cada (μ) controlará el centro aproximado donde codificar el dato de entrada
- Cada σ controlará cuánto se puede desviar en cualquiera de sus muestreos

Con este modelo el decoder asocia áreas completas (no solo puntos aislados), a ligeras variantes de la misma salida

- Esto hace que se genere un espacio mucho más suave e interpolado
- Así es capaz de producir nuevas salidas que comparten propiedades

VAE - Estructura

Para que funcione debemos dividir el espacio latente en dos vectores

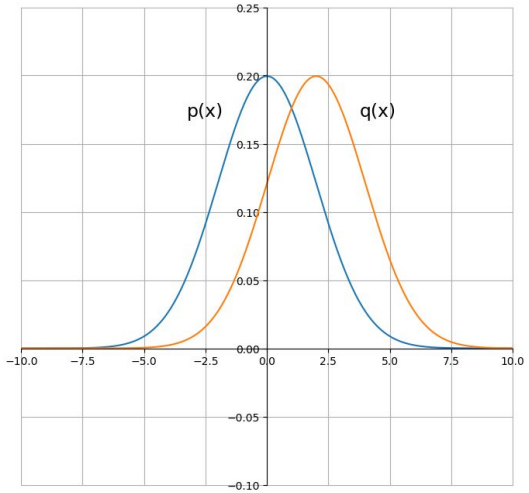


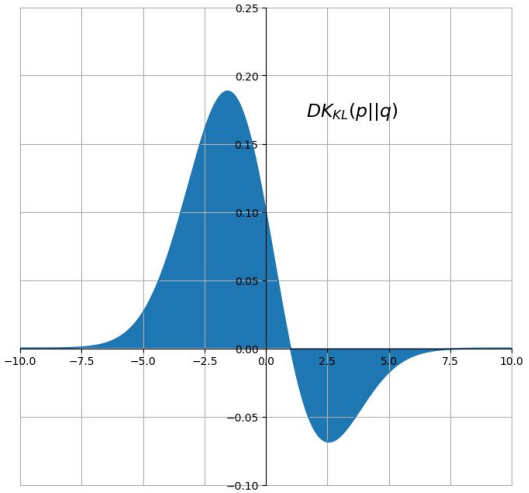
Después debemos ajustar las funciones de perdida individualmente tal que:

- Una función pérdida tradicional que calcule la distancia del objeto generado
- La divergencia KL (Divergencia de Kullback-Leibler) entre la distribución latente aprendida y la distribución previa, que actúa como término de regularización

VAE: *KL-divergence*

Mide la diferencia existente entre dos distribuciones de probabilidad





P.ej. en las distribuciones de la figura dos distribuciones:

- Una distribución normal y conocida $p(x)$
- Una distribución normal y desconocida $q(x)$

Es una **divergencia**, no una distancia ya que no es simétrica

VAE: *KL-divergence* (y II)

Forzando una distribución normal estándar ($\mu = 0$ y $\sigma = 1$) para nuestra distribución de datos tenemos que la divergencia KL se puede calcular como:

$$KL = \sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - log(\sigma_i) - 1$$

Nuestra función de pérdida se compondrá de dos términos:

- Función de pérdida tradicional $\mathcal{L}_r$: Ajustará los datos de salida
- Función de divergencia KL: Ajustará el espacio latente a la distribución estándar

Por tanto, la expresión de la función de pérdida será:

$$\mathcal{L}(y, \hat{y}) = \mathcal{L}_r(y, \hat{y}) + \mathcal{L}_{\mathrm{KL}}(y, \hat{y})$$

Ejemplo de *variational autoencoders*

Notebook: `Generando datos con variational autoencoders.ipynb`

¿Aprendizaje supervisado o no supervisado?

Traidicionalmente se han clasificado dentro del **aprendizaje no supervisado**

- Después de todo, no trabajan con datos etiquetados
- ¡Pero no se pueden optimizar autoencoders sin el propio feedback de la reconstrucción!

En el **aprendizaje supervisado**, se aprende con el feedback de los datos

- Se espera que, al proporcionar algunos ejemplos, el algoritmo descubra la función que asigna las entradas a las salidas deseadas con el menor error

Yann LeCun inventó el self-supervised learning para hablar de estos modelos.

I now call it "self-supervised learning", because "unsupervised" is both a loaded and confusing term. ... Self-supervised learning uses way more supervisory signals than supervised learning, and enormously more than reinforcement learning. That's why calling it "unsupervised" is totally misleading.

- Yann LeCun - Recent Advances in Deep Learning (2019) -

Gracias