CURSO MODULAR

BIG DATA Y DATA SCIENCE APLICADOS A LA ECONOMÍA Y A LA ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS

| Modulo: MINERÍA DE DATOS II

Autores:

Pablo Sánchez, Angel Rodríguez y Alfonso Carabantes

Organizadores

UNED

aeca

Asociación Española de Contabilidad y Administración de Empresas

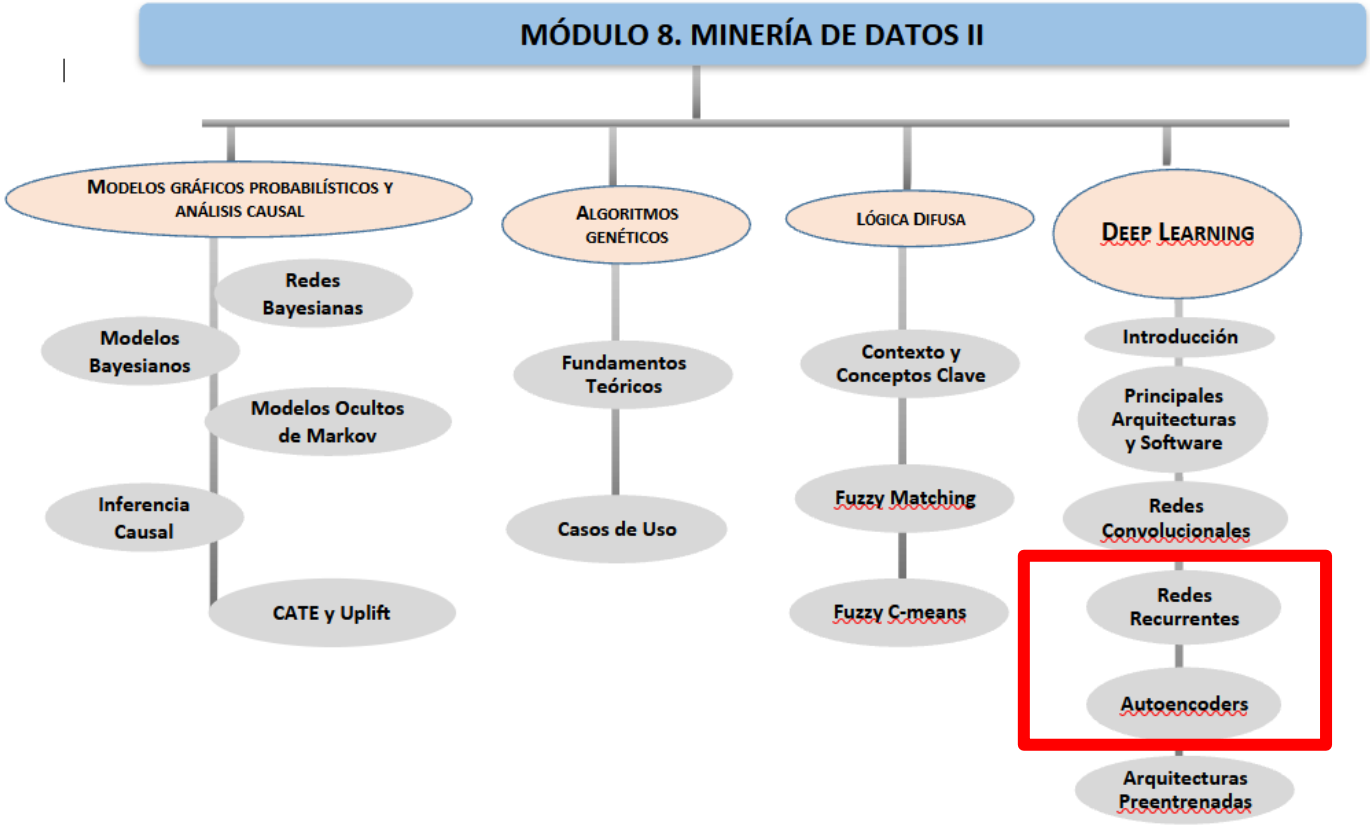
Colaborador

GRUPO DE USUARIOS DE R DE MADRID

Primera parte

5 de julio de 2024

MINERÍA DE DATOS II



ÍNDICE

- Autoencoders
  - Definición
  - Idea intuitiva y tipos de autoencoders
  - Código en Keras de varios ejemplos
- Redes Recurrentes

AUTOENCODERS

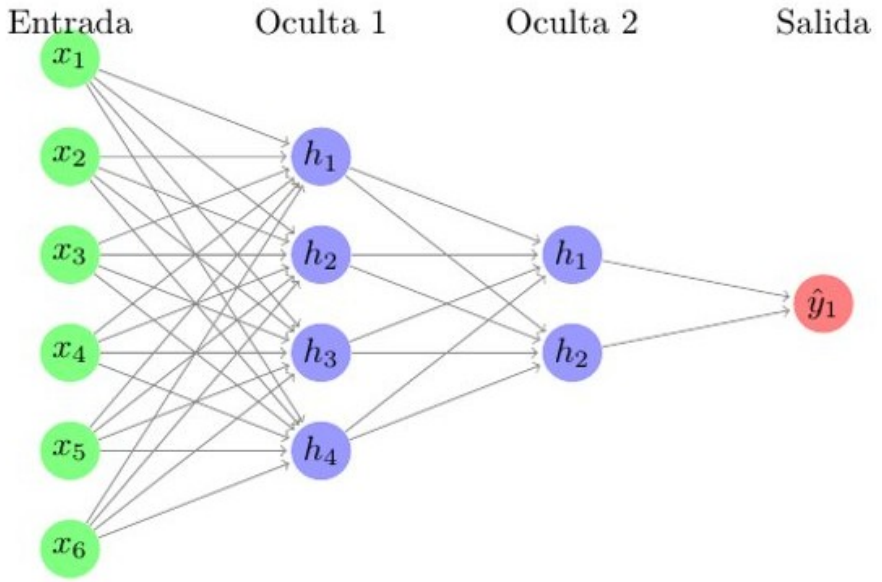
Los **Autoencoders** (AE) son uno de los tipos de redes neuronales que caen dentro del ámbito del **Deep Learning**, en la que nos encontramos con un modelo de **aprendizaje no supervisado**. Ya se empezó a hablar de AE en la década de los 80 (Bourlard and Kamp 1988), aunque es en estos últimos años donde más se está trabajando con ellos.

La **arquitectura** de un AE es una **Red Neuronal Artificial** (ANN por sus siglas en inglés) que se encuentra dividida en **dos partes**, **encoder** y **decoder** (Charte et al. 2018), (Goodfellow, Bengio, and Courville 2016). El **encoder** va a ser la parte de la ANN que va a codificar o **comprimir** los datos de entrada, y el **decoder** será el encargado de **regenerar** de nuevo los datos en la salida. Esta estructura de codificación y decodificación le llevará a tener una **estructura simétrica**.

El AE es **entrenado** para ser capaz de **reconstruir** los **datos** de **entrada** en la capa de salida de la ANN, implementando una serie de **restricciones** (la reducción de elementos en las capas ocultas del encoder) que van a evitar que simplemente se copie la entrada en la salida.

AUTOENCODERS

Si recordamos la estructura de una ANN clásica o también llamada **Red Neuronal Densamente Conectada** (ya que cada neurona conecta con todas las de la siguiente capa) nos encontramos que en esta arquitectura, generalmente, el número de neuronas por capa se va reduciendo hasta llegar a la capa de salida que debería ser normalmente un número (si estamos en un problema regresión), un dato binario (si es un problema de clasificación).

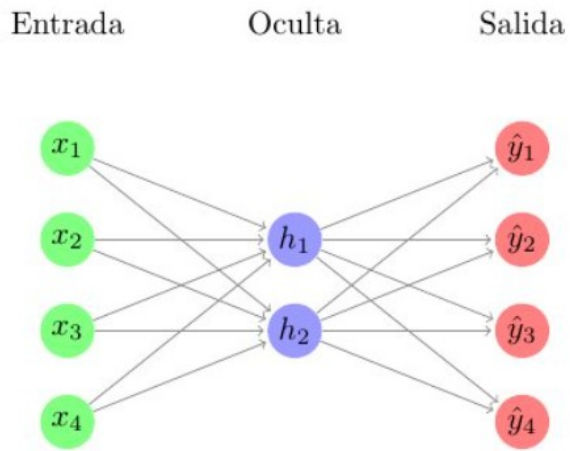


AUTOENCODERS

Si pensamos en una **estructura básica de AE** en la que tenemos una capa de entrada, una capa oculta y una capa de salida, ésta sería su representación:

Donde los valores de x_1, x_2, x_3, x_4 son los datos de entrada y los datos $\hat{y}_1, \hat{y}_3, \hat{y}_4, \hat{y}_4$ son la **reconstrucción de los mismos** después de pasar por la capa oculta que tiene sólo dos dimensiones.

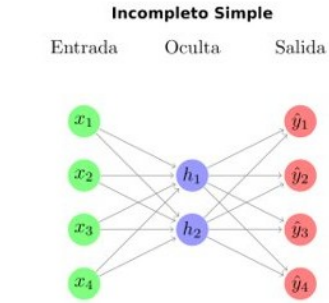
El objetivo del **entrenamiento** de un **AE** será que estos valores de $\hat{y}_i$ sean lo más **parecidos** posibles a los x_i.

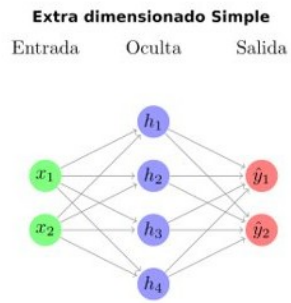


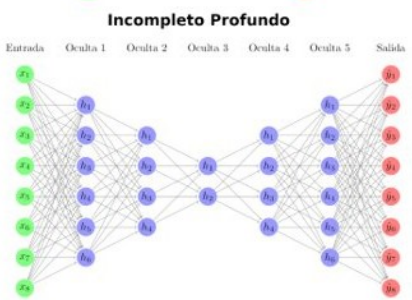
AUTOENCODERS

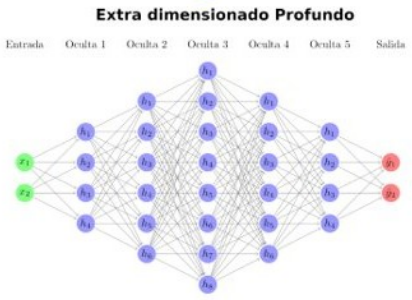
Según (Charteet al. 2018) los AE se puden clasificar según el **tipo de arquitectura en**:

- Incompleto simple
- Incompleto profundo
- Extra dimensionado simple (no se sueleusar)
- Extra dimensionado profundo (no se sueleusar)







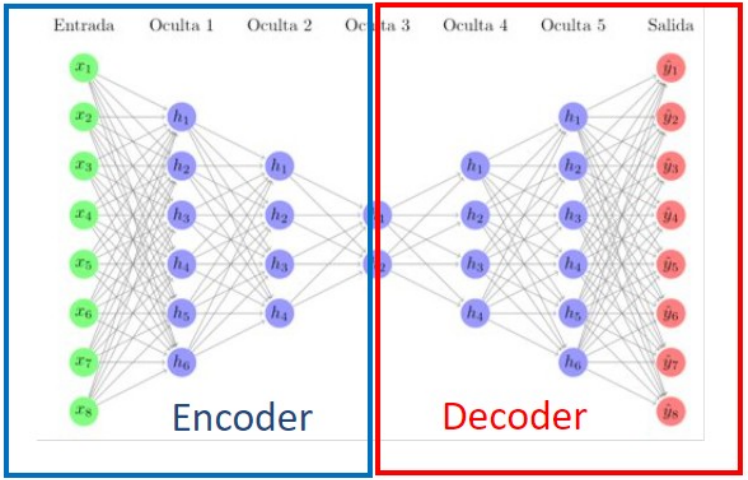


AUTOENCODERS

Idea intuitiva del uso de Autoencoders

Ya hemos comentado que la red neuronal de un AE es simétrica y está formada por un **encoder** y un **decoder**. Además, cuando trabajamos con los AE que son incompletos, se está produciendo una **reducción** del **tamaño** de los **datos** en la fase de codificación y de nuevo una **regeneración** a partir de esos **datos** más pequeños al original.

Si un AE trata de reproducir los datos de entrada mediante un encoder y decoder, ¿**que nos puede aportar si ya tenemos los datos de entrada**?



AUTOENCODERS

Reducción de dimensiones

Ya tenemos uno de los conceptos más importantes de los AE que es la **reducción de dimensiones** de los datos de entrada. Estas nuevas variables que se generan una vez pasado el encoder se les suele llamar el **espacio latente**.

Nos permite una vez entrenado el AE tener un **encoder** que es **capaz de reducir las dimensiones** del origen y luego poder volver a generar los datos.

Este concepto de **reducción de dimensione**s en el mundo de la minería de datos lo podemos asimilar rápidamente a técnicas como el **Análisis de Componentes Principales (PCA)**, que nos permite trabajar con un número más reducido de dimensiones que las originales. Igualmente, esa reducción de los datos y la capacidad de poder reconstruir el original podemos asociarlo al concepto de **compresión** de **datos**.

AUTOENCODERS

Reconocimiento de datos de la misma naturaleza (Detección de Anomalías)

Si nosotros tenemos un conjunto de **datos** de la **misma naturaleza** y los entrenamos con nuestro AE, somos capaces de construir una red neuronal (pesos en la red neuronal) que es capaz de **reproducir esos datos** a través del AE.

Que ocurre si nosotros metemos un **dato** que **no era de la misma naturaleza** que los que entrenaron el AE, lo que tendremos entonces es que al **recrear** los **datos originales no va a ser posible que se parezca a los datos de entrada**. De forma que el **error** que vamos a tener va a ser **mucho mayor** por no ser datos de la misma naturaleza.

Esto nos puede llevar a construir un AE que permita **detectar anomalías**, es decir, que seamos capaces de detectar cuando un dato es una anomalía porque realmente el AE no consigue tener un error lo bastante pequeño.

También nos podría servir para trabajar con conjuntos de **datos** de **clasificación binaria** y **muy desbalanceados** (> 90% en una clase). De forma que entrenamos los datos con la clase mayoritaria y luego podremos detectar cada clase en función de lo grande que sea el error generado en la reconstrucción.

AUTOENCODERS

Reconocimiento de datos de la misma naturaleza (Detección de Anomalías) …

Un AE nos puede ayudar a **detectar estas anomalías** de la siguiente forma:

- Tomamos todos los datos de entrenamiento de la **clase mayoritaria** (o normales) y construimos un AE para ser capaces de reproducirlos. Al ser todos estos datos de la **misma naturaleza** conseguiremos **entrenar** el AE con un **error muy pequeño**.
- Ahora tomamos los **datos** de la **clase minoritaria** (o anomalías) y los pasamos a través del AE obteniendo unos **errores** de **reconstrucción**.
- Definimos el **umbral de error** que nos separará los datos normales de las anomalías, ya que el AE sólo está entrenado con los normales y conseguirá un error más alto con las anomalías al reconstruirlas.
- Cogemos los **datos** de **test** y los vamos pasando por el AE, si el **error es menor del umbral**, entonces será de la **clase mayoritaria**. Si el error es **mayor que el umbral**, entonces estaremos ante una **anomalía**.

AUTOENCODERS

Eliminación de ruido en imágenes

Otra de las formas de uso de los autoencoders en tratamiento de imágenes es para **eliminar ruido** de las mismas, es decir poder quitar manchas de las imágenes. La forma de hacer esto es la siguiente:

- Partimos de un conjunto de **datos de entrenamiento** (imágenes) a las que le **metemos ruido**, por ejemplo, modificando los valores de cada pixel usando una distribución normal, de forma que obtenemos unos datos de entrenamiento con ruido.

- **Construimos** el **AE** de forma que los **datos** de **entrada** son los que tienen **ruido**, pero los de **salida** vamos a forzar que sean los **originales**. De forma que intentamos que **aprendan a reconstruirse como los que no tienen ruido**.

- Una vez que tenemos el AE y le pasamos datos de test con ruido, seremos capaces de reconstruirlos sin el ruido.

AUTOENCODERS

Modelos generativos

Cuando hablamos de modelos generativos, nos referimos a **AE** que son **capaces** de **generar cosas nuevas** a las que existían.

De forma que mediante técnicas como los **Variational Autoencoders, los Adversarial Autoencoders** seremos capaces de **generar nuevas imágenes** que no teníamos inicialmente.

Es decir, podríamos pensar en poder tener un **AE** que sea **capaz** de **reconstruir imágenes** de **caras**, pero que **además** con toda la información aprendida fuera **capaz** de **generar nuevas caras** que realmente no existen.

AUTOENCODERS

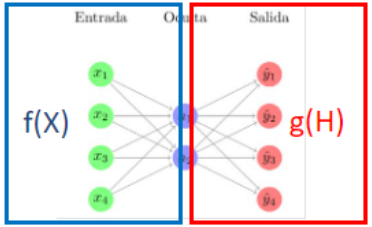
Según lo visto de forma intuitiva vamos a tener el **encoder** $f(X)$ que será el encargado de codificar los datos de entrada y luego tendremos el **decoder** $g(H)$ que será el encargado de realizar la decodificación y conseguir acercarnos al dato original X.

Es decir, intentamos conseguir $g(f(X)) = \hat{Y} \approx X$.

Si suponemos un **Simple Autoencoder** en el que tenemos **una única capa oculta**, con una **función de activación intermedia** $\boldsymbol{\delta^{(1)}}$ y una **función de activación de salida** $\boldsymbol{\delta^{(2)}}$ y los **parámetros** $\boldsymbol{W^{(i)}}$ **y** $\boldsymbol{b^{(i)}}$ representan los **parámetros de la red neuronal** en cada capa, tendríamos la siguiente expresión:

$$f(X) = \delta^{(1)}(W^{(1)} * X + b^{(1)})$$

$$g(H) = \delta^{(2)}(W^{(2)} * H + b^{(2)})$$



$$\text{donde } g(f(X)) = \delta^{(2)}(W^{(2)} * (\delta^{(1)}(W^{(1)} * X + b^{(1)}) + b^{(2)})) = \hat{Y} \approx X$$

Así tendremos que $g(f(X)) = \hat{Y}$ donde $\hat{Y}$ será la reconstrucción de X

29

CODIFICACIÓN AUTOENCODERS

Vamos a ver el Código en python con Keras/Tensorflow de varios Autoencoders

CURSO MODULAR

BIG DATA Y DATA SCIENCE APLICADOS A LA ECONOMÍA Y A LA ADMINISTRACIÓN Y DIRECCIÓN DE EMPRESAS

| Modulo: MINERÍA DE DATOS II

Autores:

Mauricio Beltrán, Pablo Sánchez y Alfonso Carabantes

Organizadores

UNED

aeca
Asociación Española de Contabilidad y Administración de Empresas

Colaborador

GRUPO DE USUARIOS DE R DE MADRID

Gracias por tu Atención!