



# Sistemas de Recomendación

**Arturo Sánchez Palacio**

**24, 27 y 28 de Enero de 2020**

# Factorización Matricial y Deep Learning

# Estructura sección

- Justificación de la Factorización Matricial
- Modelo de Factorización Matricial
- Introducción al Deep Learning
- Factorización Matricial en Keras
- Residual Learning
- Autoencoders

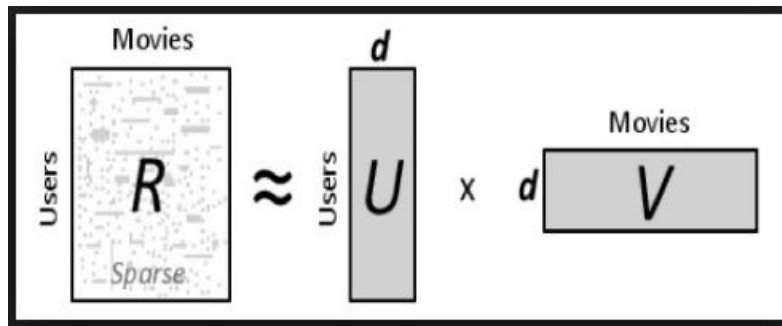
# Introducción

- El tamaño de los datos desborda nuestra capacidad computacional (y estamos trabajando con datos pequeños comparados con la realidad).
- Trabajamos con matrices dispersas enormes.
- Veremos distintos modelos para la factorización matricial.

# Factorización Matricial

**Def.** La Factorización Matricial consiste en dividir una matriz en un producto de dos matrices de menor tamaño.

$$R = W \times U^T$$



**Nota.** Hasta ahora nunca hemos almacenado la matriz. Demasiado grande.

# ¿Por qué nuestro almacenamiento es óptimo?

La matriz de ratings tendría  $N \times M$  posiciones: 3.380.000.000 posiciones

Realmente solo disponemos de 20.000.000 de evaluaciones.

Estamos economizando a un ratio de:  $20.000.000 / 3.380.000.000 = 0.006$

**Def.** A este tipo de representaciones las llamamos representaciones sparse. 

# Definición de las matrices de descomposición

- $W$  ( $N \times K$ ) matriz de usuarios.
- $U$  ( $M \times K$ ) matriz de productos.

La dimensión de las matrices variará según el  $K$  que seleccionemos.  
Normalmente elegiremos un  $K$  entre 10 y 50.

En este curso trabajaremos con un  $K = 10$ .

# Matrices de descomposición

Matriz de usuarios  $W$  ( $N \times K$ ) =  $W$  (130,000 x 10) = 1.300.000 elementos

Matriz de productos  $U$  ( $M \times K$ ) =  $U$  (26.000 x 10) = 260.000 elementos

Total: 1.560.000 elementos.

Ratio de ahorro:  $1.560.000 / 3.380.000.000 = 0.0005$



# Predicciones con matrices de descomposición

Si calculáramos  $R = W \times U^T$  el ordenador colapsaría. Pero si queremos calcular una predicción concreta de una película  $j$  para un usuario  $i$ :

$$r_{i,j} = W_i^T \times U_j$$

# Descomposición en autovalores

Def. Una descomposición en valores singulares de una matriz

Una DVS de

$\{ \displaystyle A \}$ ,

es una factorización del tipo

$\{ \displaystyle A = U \Sigma V^T \}$ ,

con

$\{ \displaystyle U \in \mathbb{R}^{m \times m} \}$

,

$\{ \displaystyle V \in \mathbb{R}^{n \times n} \}$

ortogonales y

$\{ \displaystyle \Sigma \in \mathbb{R}^{m \times n} \}$

una matriz formada con los Valores Singulares de

$\{ \displaystyle A \}$ ,

en su diagonal principal ordenados de mayor a menor.

# Descomposición en autovalores

**Def.** Una descomposición en valores singulares de una matriz  $A$  es una factorización del tipo  $A = U\Sigma V^T$  con  $U$  y  $V$  matrices ortogonales y  $\Sigma$  una matriz diagonal con los autovalores no nulos ordenados de mayor a menor en la diagonal.

**Def.** Los **vectores propios** o **autovectores** de una matriz son los vectores no nulos que cuando son multiplicados por la matriz dan un múltiplo escalar de sí mismos (no cambian la dirección).

**Def.** Al escalar antes mencionados se le denomina **valor propio** o **autovalor**.

# Descomposición matricial

Si se puede descomponer en tres matrices se puede descomponer en dos (producto de dos de ellas).

Es importante entender que nosotros no descomponemos en valores singulares.

**No se pueden calcular valores singulares para una matriz con huecos.**

# ¿Por qué tiene sentido?

Ejemplo práctico:

Predicción:  $w_i^T u_j = ||w_i|| \times ||u_j|| \times \cos(\theta)$

K = 5. Acción, comedia, romántica, humor y animación:

$w_i$  (1) cuánto le gusta al usuario i la acción.

$w_i$  (2) cuánto le gusta al usuario i la comedia.

$u_j$  (1) cuánta acción hay en la película j.

$u_j$  (2) cuánta comedia hay en la película j.

# ¿Por qué tiene sentido?

$w_i = (1, 0.8, -1, 0.1, 1)$  (gustos del usuario i)

$u_j = (1, 1.5, -1.3, 0, 1.2)$  (contenidos de la película j)

Resultado =  $1 \cdot 1 + 0.8 \cdot 1.5 + 1 \cdot 1.3 + 0.1 \cdot 0 + 1 \cdot 1.2 = 4.7$

Nota. Cuando descomponemos una matriz no sabemos lo que es cada característica. Trabajamos con **variables latentes**.

# ¿ Por qué es una aproximación y no exacto?

No estamos tomando los valores de toda la matriz solo una parte de ellos. Intentaremos entrenar el modelo de manera que elija los valores que más información nos vayan a aportar.

# Modelo de Factorización Matricial

Para construir el modelo empezamos fijando una función de pérdida. Empleamos error cuadrático:

$$J = \sum_{i,j \in \Omega} (r_{ij} - \hat{r}_{ij})^2 = \sum_{i,j \in \Omega} (r_{ij} - w_i^T u_j)^2$$

Con  $\Omega$  el conjunto de pares  $(i,j)$  donde el usuario  $i$  ha evaluado el producto  $j$ .



# Modelo de Factorización Matricial

Derivamos la función de pérdida para intentar minimizarla:

$$\frac{\partial J}{\partial w_i} = 2 \sum_{j \in \Psi_i} (r_{ij} - w_i^T u_j)(-u_j) = 0$$

Vamos a despejar  $w$ :

$$\sum_{j \in \Psi_i} (w_i^T u_j) u_j = \sum_{j \in \Psi_i} r_{ij} u_j$$

# Modelo Factorización Matricial

El producto escalar es conmutativo: (ojo dimensiones tranposición)

$$\sum_{j \in \Psi_i} (u_j^T w_i) u_j = \sum_{j \in \Psi_i} r_{ij} u_j$$

Vector x Escalar = Escalar x Vector

$$\sum_{j \in \Psi_i} u_j (u_j^T w_i) = \sum_{j \in \Psi_i} r_{ij} u_j$$

# Modelo Factorización Matricial

Propiedad asociativa:

$$\left( \sum_{j \in \Psi_i} u_j u_j^T \right) w_i = \sum_{j \in \Psi_i} r_{ij} u_j$$

Despejando  $w_i$ :

$$w_i = \left( \sum_{j \in \Psi_i} u_j u_j^T \right)^{-1} \sum_{j \in \Psi_i} r_{ij} u_j$$

# Modelo Factorización Matricial

La función de pérdida es simétrica para U así que con un razonamiento análogo:

$$u_j = \left( \sum_{i \in \Omega_j} w_i w_i^T \right)^{-1} \sum_{i \in \Omega_j} r_{ij} w_i$$

# Modelo Factorización Matricial

## Algoritmo de Mínimos Cuadrados Alternos

- Iniciamos las matrices U y W de manera aleatoria y fijamos un número T de iteraciones.
- For t in range(T):

$$w_i = \left( \sum_{j \in \Psi_i} u_j u_j^T \right)^{-1} \sum_{j \in \Psi_i} r_{ij} u_j$$
$$u_j = \left( \sum_{i \in \Omega_j} w_i w_i^T \right)^{-1} \sum_{i \in \Omega_j} r_{ij} w_i$$

**Nota.** Está demostrado que la pérdida decrece en cada iteración.


# Expansión del modelo

- Ya tenemos un primero modelo para la descomposición matricial.
- El modelo es demasiado naïve. No considera los sesgos.
- Expandimos nuestro modelo añadiendo tres coeficientes:

$$\hat{r}_{i,j} = w_i^T u_j + b_i + c_j + \mu$$

# Expansión del modelo

Sesgos:

- $b_i$  Sesgo de usuario (optimista o pesimista)
- $c_j$  Sesgo de película (es bastante intuitivo, nos habla de la calidad de la película) 
- $\mu$  Media global. La usamos para centrar en torno a 0.

# Expansión del modelo

Nuevas fórmulas:

$$u_j = \left( \sum_{i \in \Omega_j} w_i w_i^T \right)^{-1} \sum_{i \in \Omega_j} (r_{ij} - b_i - c_j - \mu) w_i$$

$$w_i = \left( \sum_{j \in \Psi_i} u_j u_j^T \right)^{-1} \sum_{j \in \Psi_i} (r_{ij} - b_i - c_j - \mu) u_j$$

$$b_i = \frac{1}{|\Psi_i|} \sum_{j \in \Psi_i} (r_{ij} - w_i^T u_j - c_j - \mu) \quad c_j = \frac{1}{|\Omega_j|} \sum_{i \in \Omega_j} (r_{ij} - w_i^T u_j - b_i - \mu)$$




# Regularización

**Def.** La **regularización** es una técnica que ayuda a prevenir el overfitting.

Un signo de overfitting es que los pesos sean muy grandes. 

Ejercemos la regularización construyendo una nueva función de pérdida:

$$J = \sum_{i,j \in \Omega} (r_{ij} - \hat{r}_{ij})^2 \text{  } + \lambda (\|W\|_F^2 + \|U\|_F^2 + \|b\|_2^2 + \|c\|_2^2)$$

# Regularizaón

Norma de Fröbenius: 

$$\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2} = \sqrt{\text{trace}(A^* A)} = \sqrt{\sum_{i=1}^{\min\{m, n\}} \sigma_i^2}$$

Con  $A^*$  la traspuesta de  $A$  y  $\sigma_i$  los valores singulares de la matriz  $A$ .

# Regularización

Nuevos coeficientes:

$$w_i = \left( \sum_{j \in \Psi_i} u_j u_j^T + \lambda I \right)^{-1} \sum_{j \in \Psi_i} (r_{ij} - b_i - c_j - \mu) u_j$$

$$u_j = \left( \sum_{i \in \Omega_j} w_i w_i^T + \lambda I \right)^{-1} \sum_{i \in \Omega_j} (r_{ij} - b_i - c_j - \mu) w_i$$

$$b_i = \frac{1}{|\Psi_i| + \lambda} \sum_{j \in \Psi_i} (r_{ij} - w_i^T u_j - c_j - \mu) \quad c_j = \frac{1}{|\Omega_j| + \lambda} \sum_{i \in \Omega_j} (r_{ij} - w_i^T u_j - b_i - \mu)$$

# Descomposición Matricial

**Construcción del modelo:**

Notebook: `matrix_factorization.ipynb`

# Descomposición Matricial Vectorizada

**Construcción del modelo:**

Notebook: `matrix_factorization_vectorized.ipynb`



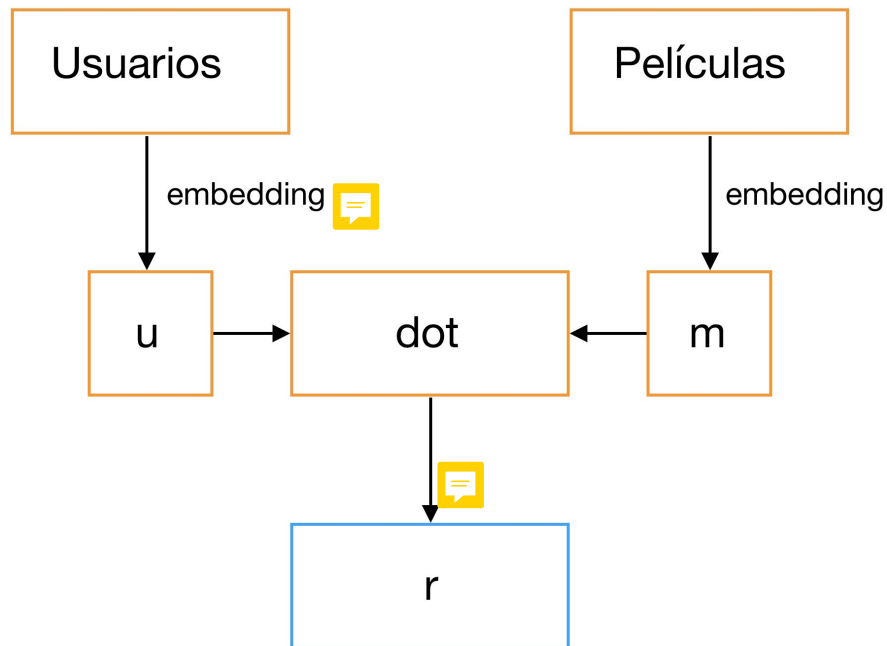
# Factorización de matrices en Keras

Descenso del gradiente:

El método de descenso del gradiente se puede emplear para optimizar cualquier función suave. Es decir, se puede usar para descomposición matricial.

# Factorización de matrices en Keras

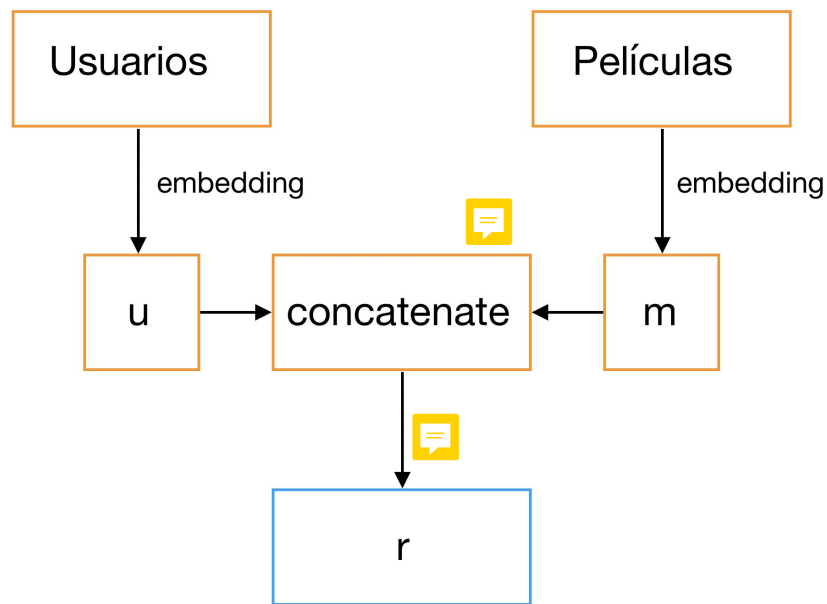
Arquitectura de nuestra red:





# Factorización de matrices en Keras

Red mejorada:



# Aprendizaje residual

- Es una técnica basada en Visión Artificial.
- Red neuronal con distintas ramas cada una encargada de aprender distintas características y concatenarlo todo finalmente.
- La factorización matricial funciona pero ¿podemos mejorarlo?
- La factorización matricial solo aprende relaciones lineales. Esto permite aprender todo tipo de relaciones.

# Aprendizaje Residual

**Construcción del modelo:**

Notebook: `residual_learning.ipynb`

# AutoEncoder

**Def.** Un **autoencoder** es una red neuronal que reproduce su propio output.

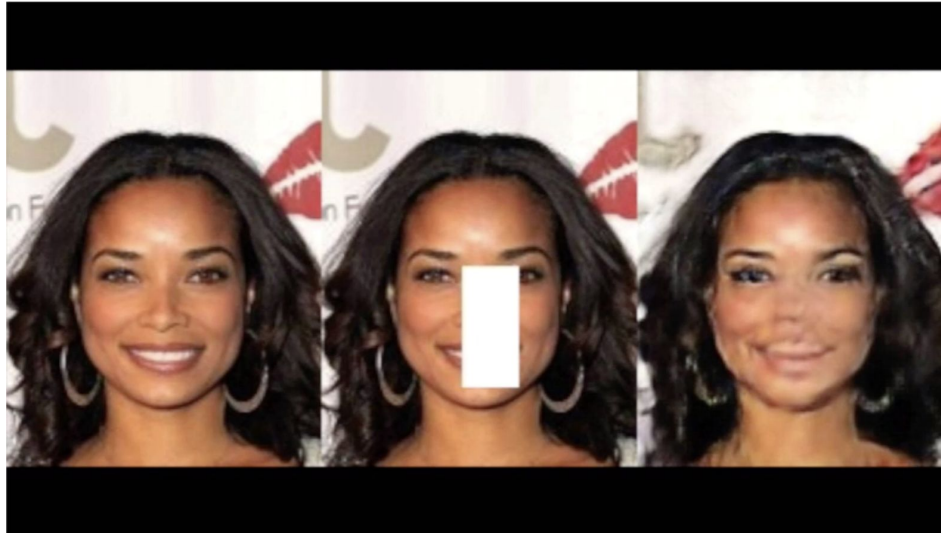
Como función de pérdida empleamos:  **$(\text{output} - \text{input})^2$**

Gran ventaja: No requiere cambios en la implementación, simplemente usar la entrada como etiqueta.

La aplicación principal de los autoencoders es la eliminación de “ruido”.

# AutoEncoder

Ejemplo de eliminación de ruido:



# AutoEncoder

Aplicación a recomendación:

- Rellenamos los huecos de la matriz como rellenaríamos los píxeles.
- El número de usuarios es el número de muestras y el número de películas es el número de características.
- AutoRec recorre los 130.000 usuarios por epoch.
- Para usar AutoEncoders necesitamos cargar la matriz. Usamos sparse matrix de NumPy.