Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

ÁREA DE CIENCIA DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL

Aprendiendo perfiles de usuario: distinción entre intereses consolidados e intereses recientes

Pérez Núñez, Pablo

TUTOR: D. Jorge Díez Peláez

FECHA: 26 de julio de 2018

Índice

Índice de figuras

Índice de tablas

Índice de algoritmos

1. Introducción

La inmensidad de contenido disponible a través de la web en la actualidad, hace que los usuarios dispongan de un amplio abanico de posibilidades entre las cuales seleccionar lo más adecuado para sus gustos y aficiones. Esta ingente cantidad de opciones hace necesaria la existencia de sistemas que ayuden al usuario a la hora de decidir y explorar el contenido disponible. Los algoritmos que nos ayudan en esta tarea se denominan Sistemas de Recomendación (SR) [1, 2].

Podemos encontrar ejemplos de estos sistemas en la mayor parte de las plataformas electrónicas que utilizamos hoy en día, siendo algunos de ellos grandes compañías como Amazon [3], Netflix [4, 5], Spotify [6] o incluso el propio buscador de Google [7].

En un inicio, las recomendaciones realizadas por estas aplicaciones se centraban únicamente en sugerir a todos los usuarios los productos más populares o más consumidos de la plataforma, lo cual puede resultar suficiente en ciertas ocasiones. En el caso de las webs de compras, sugerir productos que se compran junto al actual es otra metodología de recomendación muy utilizada.

Con el paso del tiempo, surgieron nuevas formas de recomendar más efectivas, destacando entre ellas la creación de un perfil para cada uno de los usuarios. Este perfil individualizado se obtiene a partir del historial completo de consumo del usuario y permite realizar recomendaciones personalizadas y más precisas en la mayor parte de las ocasiones, siendo en la actualidad una de las técnicas más utilizadas para la resolución de este tipo de problemas.

Otra de las posibilidades que ofrece la obtención de perfiles de usuario es la búsqueda de consumidores similares, permitiendo realizar recomendaciones del tipo *"Otros usuarios también han comprado/escuchado/visto"* o realizar tareas de agrupación de usuarios según gustos.

Actualmente, la mayoría de plataformas hacen uso de todas estas técnicas, ya sea en combinación o de manera individualizada, lo que permite ofrecer al usuario un amplio abanico de ítems donde escoger.

Para poder llevar a cabo algún tipo de recomendación, es necesario disponer de datos que, o bien describan al objeto (imagen, vídeo, audio, texto...) o bien representen la opinión de los diferentes usuarios sobre el mismo. En función de la disponibilidad de estos datos, se pueden aplicar diferentes estrategias de recomendación [8]:

- **Sistemas basados en contenido:** Si se dispone de información acerca del contenido de cada uno de los ítems a recomendar, se crea un perfil para cada uno de los ellos así como otro para cada uno de los usuarios. El objetivo es poder relacionar usuarios e ítems para llevar a cabo la recomendación.
- **Filtros colaborativos:** En el caso de que no se disponga de información detallada de cada uno de los ítems pero si exista una valoración de los ítems por parte de los usuarios (más habitual), se crea un perfil de usuario en función de sus gustos y se recomienda en base a usuarios similares.
- **Sistemas híbridos:** Finalmente, esta última categoría combina las anteriores utilizando información del contenido del ítem así como las valoraciones realizadas por otros usuarios. Los sistemas de este tipo son menos habituales dado que requieren de mucha información (no siempre disponible) además de elevar la complejidad.

2. Objetivo

La generación de perfiles de usuario se lleva a cabo empleando el historial de consumo al completo de cada uno de los ellos. Este tipo de perfiles no tienen en cuenta la posible "evolución personal" del usuario o cambio de gustos del mismo.

El objetivo principal de esta investigación es el de, a partir de un conjunto de datos de una plataforma de musica on-line, generar 2 perfiles diferentes por usuario, siendo el primero de ellos el generado a partir del historial al completo (también denominado perfil consolidado) y el segundo un perfil que contenga solamente la actividad reciente del mismo (perfil reciente).

Para generar estos perfiles, se utilizarán métodos ideados para realizar esta tarea en el ámbito de los documentos y palabras. De esta forma, las canciones pasarán a tratarse como palabras y los usuarios como documentos. Si bien esto puede parecer extraño, ambos mundos comparten muchas similitudes, ya que, un usuario es un conjunto de reproducciones (canciones) y un documento es un conjunto de palabras (ambos con un orden preestablecido y coherente).

Una vez conocidos estos perfiles, se utilizarán para realizar recomendaciones basadas en filtros colaborativos en diferentes tareas. Cada una de estas tareas se evaluará empleando cada perfil individualmente o en combinación con el fin de comparar y conocer si resulta interesante este tipo de planteamiento.

3. Trabajo relacionado

Si bien no se han encontrado trabajos previos donde se realice una separación temporal entre perfiles de usuario, existen numerosos estudios relacionados con las recomendaciones en el ámbito musical.

Destacan numerosos trabajos donde se hace uso de la factorización de matrices[1] con el fin de proyectar ítems en un nuevo espacio. Un ejemplo de ello es [9] donde los autores tratan de, a partir de una canción dada, predecir que artistas han podido interpretarla y a partir de un artista, predecir las canciones que ha podido interpretar. Todo ello se lleva a cabo proyectando artistas y canciones en un nuevo espacio empleando la ya citada factorización de matrices. Otras tareas realizadas en [9] son la búsqueda de canciones similares a una dada y la búsqueda de artistas semejantes a uno preestablecido.

La generación de listas de reproducción de forma automatizada, es el objetivo para el cual los autores de [10] generan proyecciones de usuarios y canciones. En este caso se hace uso de un algoritmo de machine learning denominado LME (Latent Markov Embedding) para aprender representaciones de *playlist* existentes y poder generar nuevas listas personalizadas para usuarios concretos. Un trabajo con objetivo similar se puede encontrar en [11] donde además se hace uso de tags.

Otra aplicación semejante, fuera del contexto musical, se puede ver en [12], donde se realizó un aprendizaje de perfiles de consumidores de carne. El objetivo de este aprendizaje era aprender y analizar las preferencias de los consumidores.

Finalmente, [11] es uno de los trabajos que más relación posee con el que se pretende abordar. En él, los autores hacen uso de datos muy similares a los que se utilizarán a lo largo de este documento con un fin semejante; explorar como cambian las preferencias de escucha de los usuarios a lo largo del tiempo. Para ello, los autores dividieron el conjunto de reproducciones de cada usuario en años y cada año en 4 partes de 3 meses. Con cada uno de esos cuartos, se va modificando y representando el perfil de cada usuario permitiendo finalmente generar una trayectoria visualmente trazable de la evolución de cada perfil.

[1] Proceso empleado típicamente para transformar un espacio de entrada a una nueva dimensión con el fin de obtener una mejor representación de los datos.

4. Métodos a utilizar

Con el fin de conseguir los objetivos de esta investigación, se utilizarán diversos métodos y técnicas existentes las cuales se detallarán a continuación.

4.1. Codificación de palabras

Tradicionalmente, los sistemas de procesamiento del lenguaje natural han representado las palabras mediante un vector resultado de aplicar el método *One-Hot*[2] a un ID de las mismas.

La representación *One-Hot* se ha utilizado en numerosas ocasiones dada su simplicidad y robustez, pero provoca que cada palabra se comporte como una entidad única aislada del resto y sin relación aparente con otras similares.

Palabra	**Casa**	**Perro**	**Manzana**	**Mascota**	**Hogar**	**Limón**
ID	2	3	1	0	5	4
One-Hot	0	0	0	1	0	0
	0	0	1	0	0	0
	1	0	0	0	0	0
	0	1	0	0	0	0
	0	0	0	0	0	1
	0	0	0	0	1	0

Vector One-Hot asociado a "Perro"

Tabla 1 – Codificación *One-Hot*

Utilizando dicha codificación, sinónimos como "casa" y "hogar" tendrían vectores totalmente diferentes y no poseerían ninguna relación (cuando en realidad tendrían que ser muy similares). Este comportamiento se refleja en la Tabla 1.

En el año 2013 surge una nueva forma de codificar estas palabras con el fin de suplir las carencias de los métodos anteriores denominada Word2Vec (W2V) [13].

4.2. Word2Vec

La principal virtud de este nuevo método es la capacidad de representar cada una de las palabras en función de su semántica de tal forma que, dos palabras similares o sinónimas posean una representación casi idéntica.

Para llevar a cabo esta tarea, el vector que representa a cada una de las palabras no se obtiene directamente aplicando un *One-Hot* al ID de las mismas sino que se crea un modelo que aprende estos vectores o embeddings haciendo uso de las palabras de su contexto.

[2]Método que genera un vector único de ceros y un solo uno ubicado en la posición correspondiente al ID de la palabra actual.

Estos embeddings están formados por propiedades o variables latentes que el modelo se encarga de buscar para poder distinguir unas palabras de otras. El significado de estas variables latentes es totalmente desconocido para el usuario una vez entrenado el modelo, solo se conoce de ellas los valores que poseen para cada una de las palabras.

A modo de ejemplo, en la Tabla 2 se indican unos nombres para las variables latentes a fin de ilustrar el funcionamiento del método, pero como se mencionó previamente, sólo se conocen los valores que forman los embeddings.

	Palabra	**Casa**	**Perro**	**Manzana**	**Mascota**	**Hogar**	**Limón**
	ID	40	96	2	585	788	12
Variables latentes	Tamaño	0.8	-0.2	-0.6	0	0.9	-0.7
	Género	0.0	-0.1	-0.02	-0.2	0.0	0.01
	Comida	0.01	0.2	1	0.3	-0.1	0.9
	Edificio	0.9	0.01	-0.1	0.02	0.8	0.1
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	Coste	0.7	-0.5	-0.9	-0.6	0.9	-0.8

Embedding asociado a "Perro"

Tabla 2 – Embeddings y variables latentes Word2Vec

Con esta representación, se puede ver de forma clara que, tanto el embedding asociado a "casa" y el asociado a "hogar" son muy similares, así como los que representan "manzana" y "limón" o "perro" y "mascota". Este comportamiento no sucedía en casos anteriores donde simplemente se utilizaba la codificación *One-Hot*.

Este nuevo tipo de representación posee numerosas propiedades que no poseían la codificación *One-Hot*. La propiedad más destacada es la capacidad de operar con los vectores de forma que se puedan extraer analogías entre palabras como la presentada en la Figura 1 donde a partir de 4 embeddings y aplicando la ecuación

$$\boldsymbol{r} = \boldsymbol{e}_{España} - \boldsymbol{e}_{Madrid} + \boldsymbol{e}_{París} \simeq \boldsymbol{e}_{Francia} \tag{1}$$

donde $\mathbf{e}_x$ se refiere al embedding de la palabra x, se obtiene una relación entre un país y su correspondiente capital.

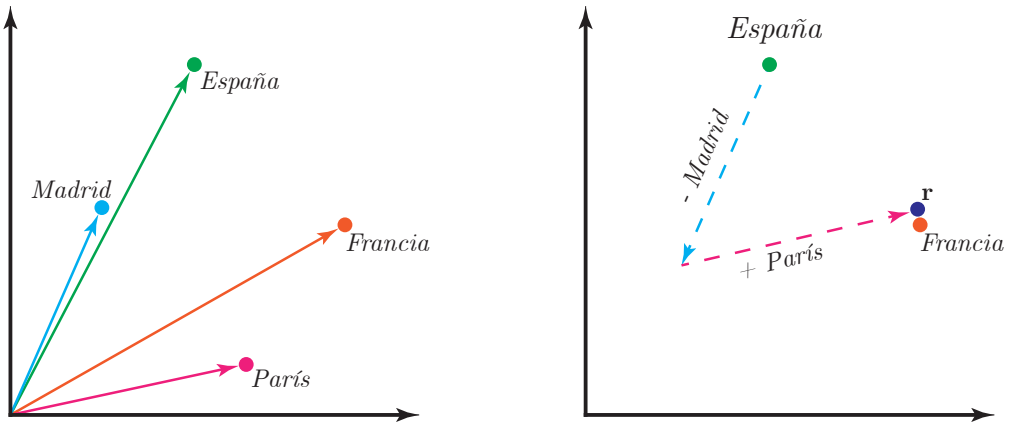

Figura 1 – Operaciones con embeddings en 2 dimensiones

Otros ejemplos de este comportamiento serían los relacionados con el género de las palabras, donde se cumplirían ecuaciones del tipo

$$\boldsymbol{e}_{Rey} - \boldsymbol{e}_{Hombre} + \boldsymbol{e}_{Mujer} \simeq \boldsymbol{e}_{Reina} \tag{2}$$

u otras como

$$\boldsymbol{e}_{Prima} - \boldsymbol{e}_{Mujer} + \boldsymbol{e}_{Hombre} \simeq \boldsymbol{e}_{Primo} \ . \tag{3}$$

Hay que tener en cuenta que Mikolov et al. [13], creadores del método Word2Vec, no fueron los primeros en utilizar este tipo de representaciones para cada una de las palabras [14] [15] [16], pero sí los primeros en plantear unos métodos más simples y menos costosos computacionalmente.

4.2.1. Arquitectura del modelo

Una vez conocidas las bases sobre las que se fundamenta el método así como las capacidades del mismo, es necesario indicar ahora la arquitectura planteada por Mikolov et al.

En el momento en el que se ideó el método Word2Vec, el objetivo principal era el de crear un modelo más simple y computacionalmente menos costoso que los ya existentes. Para conseguir esto los autores crearon un modelo basado en redes neuronales que, a diferencia de los ya existentes, poseía un menor número de capas ocultas así como un menor número de capas no lineales. Este modelo resultante se ilustra en la Figura 2 y consta de las siguientes capas:

- **Capa de entrada (x):** Capa de dimensión n (número de ítems del vocabulario). En ella se introducirán las palabras codificadas en formato *One-Hot*.
- **Capa oculta o de embeddings (e):** Obtenida a partir de una relación lineal de la anterior y una matriz de pesos, esta capa posee una dimensión (m) normalmente diferente (menor) al espacio de entrada. Esta capa contendrá el embedding asociado a la palabra introducida en la capa anterior.
- **Capa de salida ($\hat{\mathbf{y}}$):** Con la misma dimensión que la capa de entrada, esta capa retornará el resultado del modelo que variará en función de método de aprendizaje de embeddigs utilizado (Sección 4.2.2).

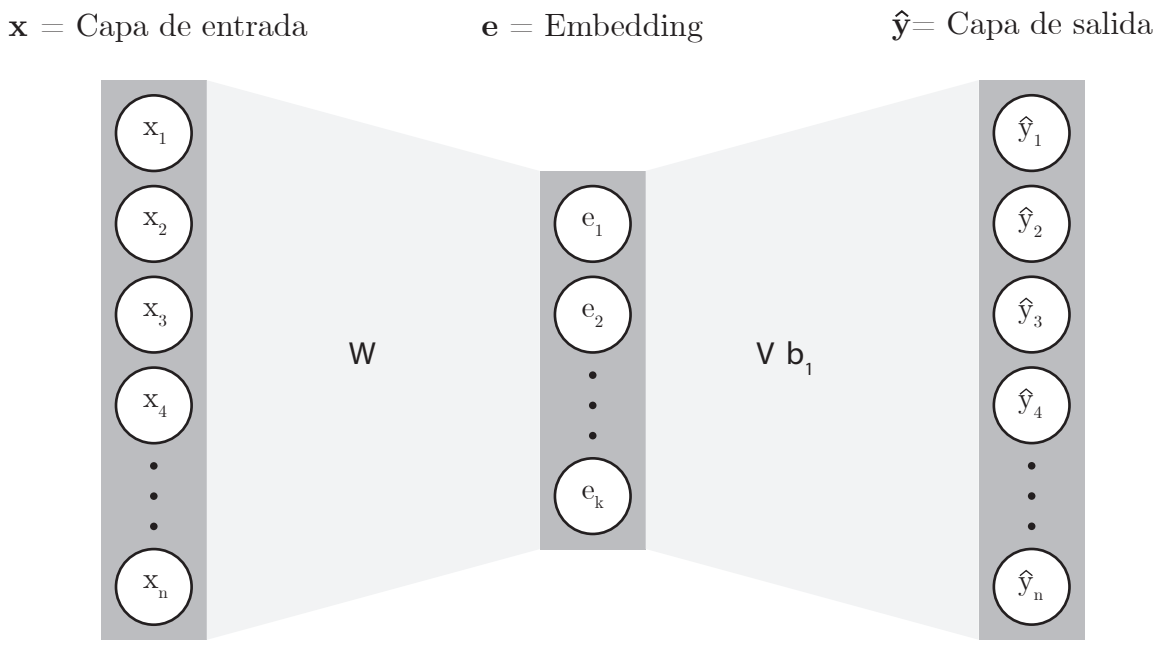

Figura 2 – Arquitectura de la red Word2Vec

Para obtener el resultado $\hat{\mathbf{y}}$ del modelo, primero es necesario multiplicar el vector de entrada $\mathbf{x}$ por la matriz de pesos W para obtener el embedding asociado $\mathbf{e}$. Esta operación se realiza aplicando la siguiente ecuación:

$$\boldsymbol{e} = \boldsymbol{W}\boldsymbol{x} \ . \tag{4}$$

Una vez obtenido el vector $\boldsymbol{e}$, el vector final $\hat{\boldsymbol{y}}$ se obtiene aplicando la ecuación

$$\boldsymbol{v} = \boldsymbol{V}\boldsymbol{e} + \boldsymbol{b}_1 \qquad \hat{\boldsymbol{y}} = g(\boldsymbol{v}) \tag{5}$$

sobre el resultado de la capa oculta.

Todas las ecuaciones previas se pueden agrupar de tal forma que nos permita obtener la ecuación del modelo al completo:

$$f(\boldsymbol{x}) = g(\boldsymbol{V}(\boldsymbol{W}\boldsymbol{x}) + \boldsymbol{b}_1) \ . \tag{6}$$

La función denominada g es la encargada de transformar en probabilidades los resultados "en crudo" retornados por el modelo (conocidos como logits). Esta función es la SoftMax y su funcionamiento se ilustra en la Figura 3.

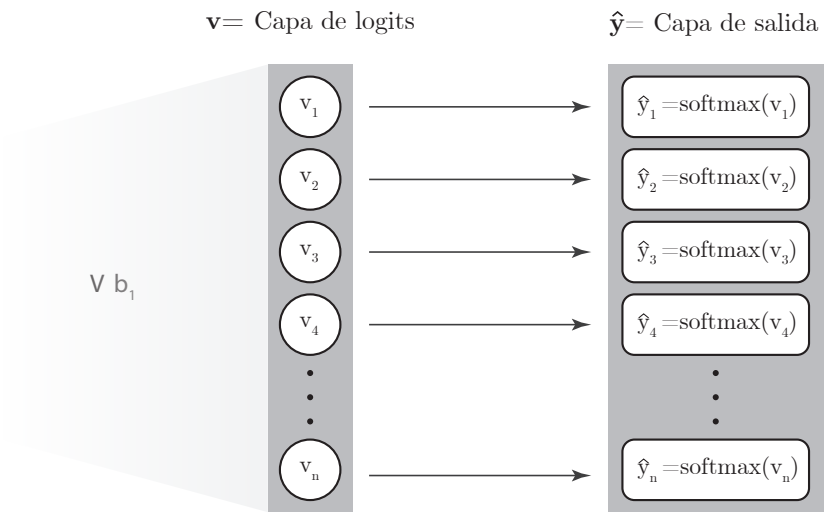

Figura 3 – Funcionamiento de función Softmax

Como se puede ver, para cada una de las salidas del modelo se aplica la función Softmax de la siguiente forma:

$$\hat{y}_i = \frac{\exp(v_i)}{\sum_{j=1}^{j=n} \exp(v_j)} \ , \tag{7}$$

donde n es el número total de palabras del corpus.

En cuanto a la función encargada de evaluar la pérdida[3] del modelo (*loss*), se utilizará la Cross Entropy. A continuación, en la Figura 4 se detalla su ubicación dentro del modelo.

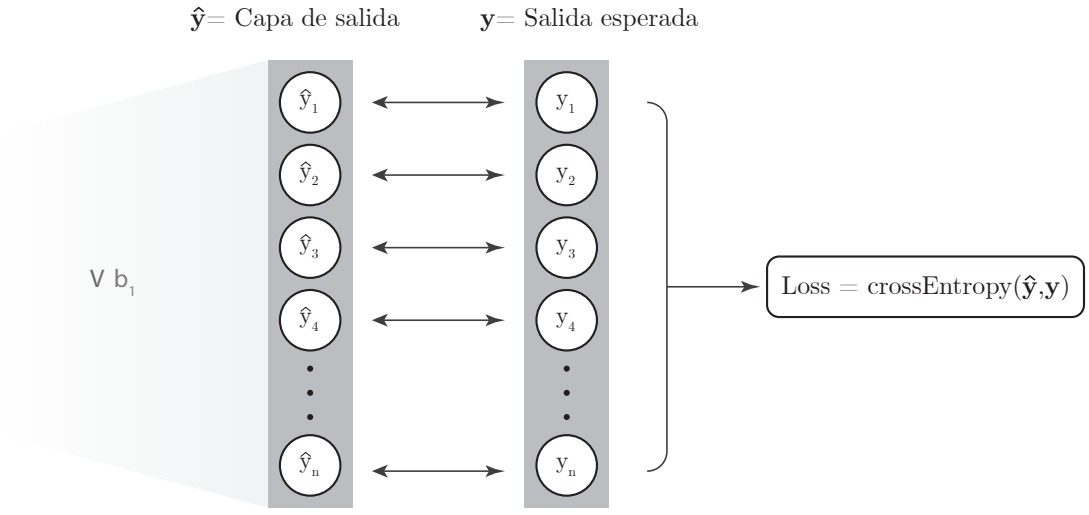

Figura 4 – Cálculo de la pérdida del modelo

[3]La pérdida del modelo se calcula comparando la salida del modelo y el valor verdadero o esperado.

Partiendo de los vectores de salida (el predicho $\hat{\boldsymbol{y}}$ y el esperado $\boldsymbol{y}$), se obtiene el valor de pérdida aplicando la siguiente ecuación:

$$loss = -\sum_{i} \hat{y}_i log(y_i) \ . \tag{8}$$

4.2.2. Aprendizaje de embeddings

En su artículo, Mikolov et al. [13] plantean dos técnicas o métodos para aprender estos embeddings asociados a cada una de las palabras. A continuación de detallarán cada uno de ellos indicando sus principales diferencias.

4.2.2.1. Continuous Bag Of Words (CBOW)

Esta arquitectura está orientada a aprender el embedding asociado a una palabra utilizando como entrada las palabras de su contexto (concretamente la suma de sus vectores *One-Hot*). De esta forma se espera que la capa de salida del modelo $\hat{\mathbf{y}}$ retorne el vector *One-Hot* de la palabra a aprender.

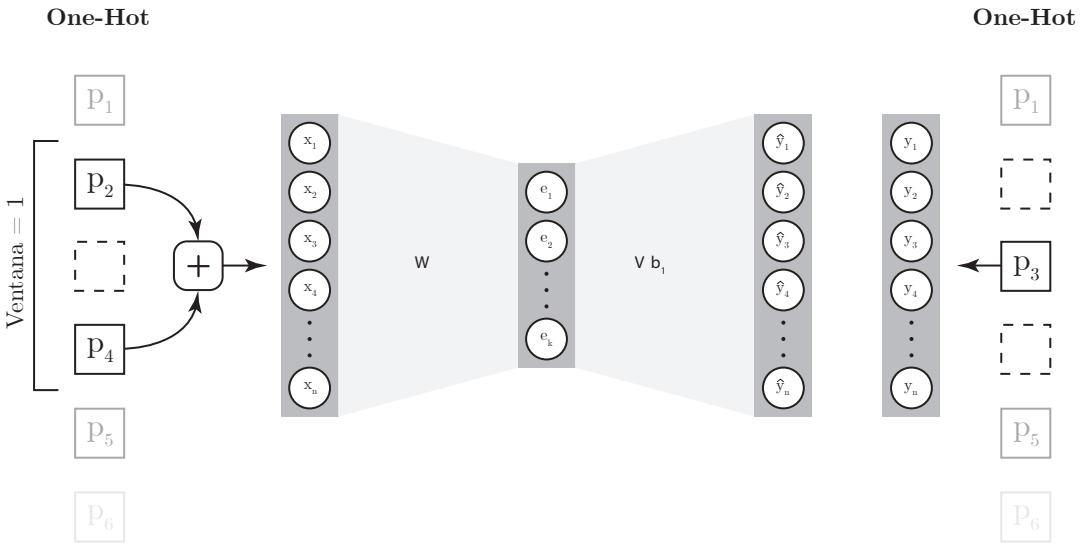

Figura 5 – Funcionamiento del Continuous Bag Of Words (CBOW) con ventana=1

La Figura 5 ilustra el comportamiento de esta arquitectura donde $\mathcal{P}_{doc} = \boldsymbol{p}_1, \boldsymbol{p}_2 \ ... \ \boldsymbol{p}_m$ es una lista de palabras ordenadas (según su aparición en una frase o documento) y codificadas mediante el método *One-Hot*.

Siendo $\mathbf{p}_j$ el ítem a predecir y $\mathcal{S}_j$ el conjunto de todos los ítems $\mathcal{S}_i$ dentro de su ventana, el conjunto de datos de entrenamiento estará formado por pares del tipo

$$\mathcal{D}_{cbow} = \{(\boldsymbol{c}_j, \boldsymbol{p}_j) : \boldsymbol{p}_i \in \mathcal{P}_{doc}\} \ . \tag{9}$$

donde $\boldsymbol{c}_j$ se obtiene como

$$\boldsymbol{c}_j = \sum_{i=1}^{|\mathcal{S}_j|} \mathcal{S}_i \ . \tag{10}$$

4.2.2.2. Continuous Skip-gram Model

A diferencia del modelo anterior, el Skip-gram realiza el aprendizaje a la inversa. El embedding asociado a cada canción, se aprenderá en este caso utilizando cada una de las canciones que se encuentran en su contexto.

Esta nueva forma de aprender los embeddings, retorna mejores resultados según las pruebas realizadas por sus autores [13]. En la Figura 6 se representa en mayor profundidad el funcionamiento del método.

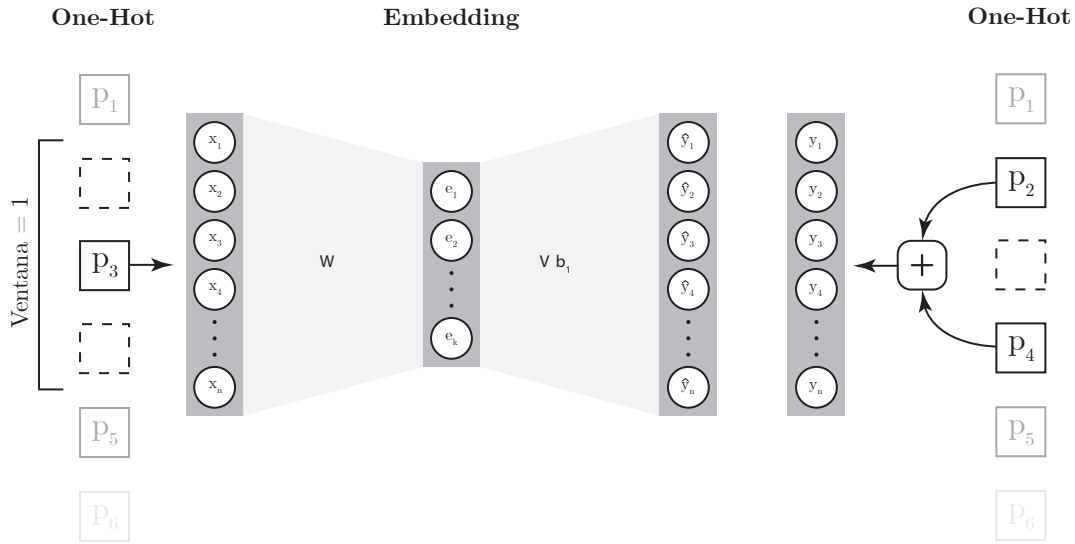

Figura 6 – Funcionamiento del Continuous Skip-gram con ventana=1

Los ejemplos en este caso serían pares del tipo

$$\mathcal{D}_{skgram} = \{(\boldsymbol{p}_j, \boldsymbol{c}_j) : \boldsymbol{p}_i \in \mathcal{P}_{doc}\} \ . \tag{11}$$

A partir de los ejemplos $\mathcal{D}_{cbow}$ o $\mathcal{D}_{skgram}$, se aprenden las matrices $\boldsymbol{W}$ y $\boldsymbol{V}$ junto con el vector $\boldsymbol{b_1}$ correspondiente optimizando la función de *loss* mediante el método del Descenso del Gradiente (GD).

En cada iteración del GD, se actualiza el valor de cada una de las matrices y vectores pasando a ser

$$\boldsymbol{V} \leftarrow \boldsymbol{V} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{V}}\right] \ , \ \boldsymbol{b_1} \leftarrow \boldsymbol{b_1} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{b_1}}\right] \ , \ \boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{W}}\right] \ , \tag{12}$$

donde γ es el *"learning-rate"* del optimizador GD.

El proceso de aprendizaje al completo se resume, para cada uno de los métodos existentes, en los algoritmos 1 y 2 detallados a continuación.

Algoritmo 1 Entrenamiento del W2V para el método CBOW.

input: $\mathcal{D}_{cbow}$, $\gamma > 0$
assign valores aleatorios (distribución gaussiana) a $\boldsymbol{W}$, $\boldsymbol{V}$ y $\boldsymbol{b_1}$
repeat
  Obtener un ejemplo del tipo $(\boldsymbol{c}_j, \boldsymbol{p}_j)$
  $\boldsymbol{v} = \boldsymbol{V}(\boldsymbol{W}\boldsymbol{c_j}) + \boldsymbol{b}_1$, Ecuación (5)
  $\hat{\boldsymbol{y}}_i = \frac{\exp(v_i)}{\sum_{j=1}^{j=n} \exp(v_j)}$, $i = 1 \cdots n$, Ecuación (7)
  $loss = -\sum_i \hat{y}_i \; log(\boldsymbol{p}_{ji})$, Ecuación (8)
  $\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{W}}\right]$
  $\boldsymbol{V} \leftarrow \boldsymbol{V} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{V}}\right]$
  $\boldsymbol{b_1} \leftarrow \boldsymbol{b_1} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{b_1}}\right]$
until *criterio de parada*
return $\boldsymbol{W}$

Algoritmo 2 Entrenamiento del W2V para el método SKGRAM.

input: $\mathcal{D}_{skgram}$, $\gamma > 0$
assign valores aleatorios (distribución gaussiana) a $\boldsymbol{W}$, $\boldsymbol{V}$ y $\boldsymbol{b_1}$
repeat
  Obtener un ejemplo del tipo $(\boldsymbol{p}_j, \boldsymbol{c}_j)$
  $\boldsymbol{v} = \boldsymbol{V}(\boldsymbol{W}\boldsymbol{p_j}) + \boldsymbol{b}_1$, Ecuación (5)
  $\hat{\boldsymbol{y}}_i = \frac{\exp(v_i)}{\sum_{j=1}^{j=n} \exp(v_j)}$, Ecuación (7)
  $loss = -\sum_i \hat{y}_i \; log(\boldsymbol{c}_{ji})$, Ecuación (8)
  $\boldsymbol{W} \leftarrow \boldsymbol{W} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{W}}\right]$
  $\boldsymbol{V} \leftarrow \boldsymbol{V} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{V}}\right]$
  $\boldsymbol{b_1} \leftarrow \boldsymbol{b_1} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{b_1}}\right]$
until *criterio de parada*
return $\boldsymbol{W}$

Utilizando cualquiera de los métodos anteriores, se obtiene como resultado la matriz $\boldsymbol{W}$. Esta matriz contiene un conjunto de embeddings que representan cada una de las palabras del vocabulario en un nuevo espacio de dimensión k.

4.3. Doc2Vec

A la hora de codificar documentos, surge la necesidad de representar cada uno de ellos con vectores de longitud idéntica. Si bien métodos como el *One-Hot* podrían ser válidos en ciertas ocasiones, tradicionalmente este problema se suele solventar mediante el uso de los "n-grams".

Doc 1: *The cat sat on the mat*
Doc 2: *The cat is red*

1-gram (BOW)	Doc 1	Doc 2
the	2	1
cat	1	1
sat	1	0
on	1	0
mat	1	0
is	0	1
red	0	1

2-gram	Doc 1	Doc 2
the cat	1	1
cat sat	1	0
sat on	1	0
on the	1	0
the mat	1	0
cat is	0	1
is red	0	1

3-gram	Doc 1	Doc 2
the cat sat	1	0
cat sat on	1	0
sat on the	1	0
on the mat	1	0
the cat is	0	1
cat is red	0	1

Tabla 3 – Ejemplo de los métodos n-gram y Cbow

En el ámbito de los textos, los "grams" son conjuntos de tamaño n generados a partir de un texto (ver Tabla 3). Una vez generados todos los grams posibles, se codifica cada uno de los documentos con un vector de longitud s (número de grams totales) indicando en la posición de cada gram el número de veces que aparece en cada documento. Si el valor de n es igual a 1, el método pasa a denominarse "bag of words" (BOW).

Cualquiera de las representaciones anteriores presenta problemas similares a los ya vistos en el caso de las palabras y el *One-Hot*. En un caso donde existan dos documentos que hablen del mismo tema utilizando palabras diferentes o sinónimos, con estas codificaciones, pueden poseer representaciones muy distantes.

Un ejemplo de esta problemática sería el documento con el texto "La casa es turquesa" y otro con el texto "El hogar es azul". Ambos tendrían representaciones muy diferentes (solo tendrían la palabra "es" en común) cuando en realidad poseen un significado muy parecido.

Con todos estos problemas en mente y partiendo del método anterior donde se aprendía una representación más certera para cada palabra del vocabulario, surge el Doc2Vec (D2V), donde se realiza el mismo proceso pero orientado a los documentos.

Este método fue planteado por los mismos autores que crearon el Word2Vec (Mikolov et al.) en un artículo del año 2014 [17] y posee una arquitectura y funcionamiento similares. El objetivo del D2V es tratar de paliar los problemas citados permitiendo obtener una representación mucho más fiel de cada documento. Para llevar a cabo esta tarea hace uso de diferente información y técnicas para aprender los embeddings.

4.3.1. Aprendizaje de embeddings

El Doc2Vec se aprovecha de información como los embeddings aprendidos previamente para cada una de las palabras (W2V), el orden de las mismas dentro del documento o simplemente la presencia o no de estas. Combinando estos datos, el D2V consigue crear un embedding para cada uno de los documentos.

En función de la información utilizada para el aprendizaje, el método Doc2Vec distingue dos configuraciones diferentes denominadas D2V-DM y D2V-DBOW. Según los autores, es necesario aprender un embedding por cada uno de los métodos anteriores y combinar ambos finalmente para obtener la mejor representación del documento. A continuación se detallarán cada uno de estos métodos.

4.3.1.1. Distributed Memory (D2V-DM)

En la primera de las arquitecturas propuestas, los autores se inspiraron en el funcionamiento del W2V, donde se tiene muy en cuenta el orden de precedencia de las palabras. En este caso, el modelo a entrenar aprenderá el embedding de cada documento mientras intenta predecir, a partir de un usuario y 2 palabras dadas, la siguiente en el documento.

Estas palabras previas, se introducirán en el modelo según su codificación W2V aprendida en la tarea anterior, de esta forma, se están utilizando representaciones mucho más fiables que un simple *One-Hot*. Teniendo todo esto en cuenta, es necesario entrenar un modelo (ver Figura 7) compuesto de:

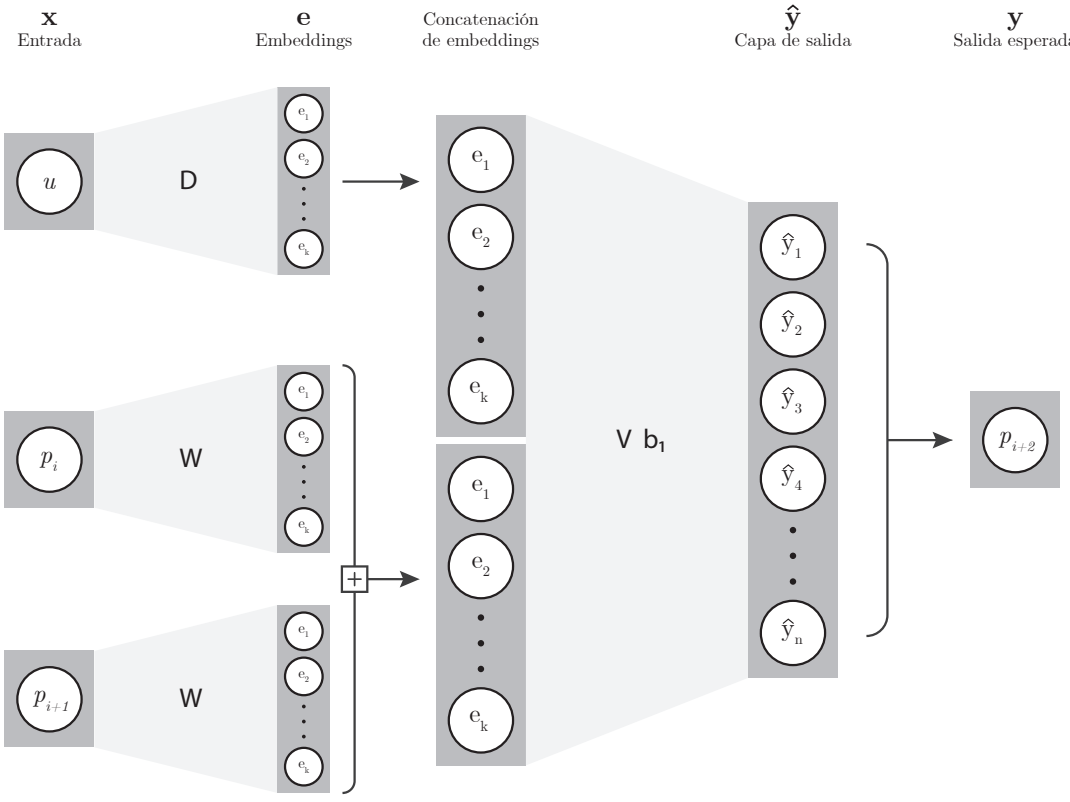

Figura 7 – Estructura del método Doc2Vec-DM

- **Capa de entrada (x):** En esta capa se introducirán, en cada iteración, un ID de documento junto con los IDs de la palabra 1 y 2 (por orden de aparición).
- **Capa de embeddings (e):** Se obtienen los embeddings de las canciones (usando W del W2V) y el del documento.
- **Concatenación de embeddings:** En este punto se concatenan el embedding del documento con la suma de los embeddigs de las canciones 1 y 2
- **Capa de salida ($\hat{\mathbf{y}}$):** Capa que contiene la salida del modelo.
- **Salida esperada (y):** Esta capa contiene el ID real de la palabra 3 (a continuación de la 1 y 2 en el documento).

Destacar de este modelo que, se utilizan 2 palabras previas con el fin de predecir una tercera, pero este número puede ser alterado. Finalmente recalcar que, en este modelo, la matriz W que retorna el embedding de las palabras permanece inalterable durante el entrenamiento dado que ya se ha aprendido en una tarea anterior (W2V).

Dado $\mathcal{U} = \{u_1, u_2, u_3 \ldots\}$ conjunto de documentos y $\mathcal{P}_u = \{p_1, p_2, p_3 \ldots\}$ el conjunto de palabras del documento u, se entrenará el modelo con cuádruplas del tipo

$$\mathcal{D}_{d2v-dm} = \{(u, p_i, p_{i+1}, p_{i+2}) : u \in \mathcal{U}, p_i \in \mathcal{P}_u\} \ . \tag{13}$$

Con cada uno de los ejemplos, el modelo transforma la entrada en la salida aplicando

$$((\boldsymbol{D}_u \oplus (\boldsymbol{W}_{p_i} + \boldsymbol{W}_{p_{i+1}}))\boldsymbol{V}) + \boldsymbol{b_1} \ , \tag{14}$$

donde $\boldsymbol{D}_u$ corresponde a la columna u de la matriz $\boldsymbol{D}$, $\boldsymbol{W}_{p_i}$ es la columna p_i de la matriz $\boldsymbol{W}$, $\boldsymbol{W}_{p_{i+1}}$ corresponde a la columna p_{i+1} de la matriz $\boldsymbol{W}$ y $\boldsymbol{W}$ se ha aprendido previamente utilizando el método W2V.

La función de pérdida

$$loss = NCE(((\boldsymbol{D}_u \oplus (\boldsymbol{W}_{p_i} + \boldsymbol{W}_{p_{i+1}}))\boldsymbol{V}) + \boldsymbol{b_1}, p_{i+2}) \tag{15}$$

se calcula empleando el método NCE (Noise Contrastive Estimation) que se explicará posteriormente.

Esta loss se minimizará mediante GD donde, en cada iteración, se actulizan los valores de $\boldsymbol{D}$,$\boldsymbol{V}$ y $\boldsymbol{b_1}$ de la siguiente forma:

$$\boldsymbol{D} \leftarrow \boldsymbol{D} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{D}}\right] \ , \ \boldsymbol{V} \leftarrow \boldsymbol{V} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{V}}\right] \ , \ \boldsymbol{b_1} \leftarrow \boldsymbol{b_1} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{b_1}}\right] \ . \tag{16}$$

Destacar una vez más que la matriz $\boldsymbol{W}$ proviene del W2V y por tanto permanece inalterable durante el aprendizaje del modelo. A modo de resumen, se indica el procedimiento completo en el Algoritmo 3.

Algoritmo 3 Entrenamiento del D2V para el método DM.

input: $\mathcal{D}_{d2v-dm}$, $\mathcal{W}$, $\gamma > 0$
assign valores aleatorios (distribución gaussiana) a $\boldsymbol{D}$, $\boldsymbol{V}$ y $\boldsymbol{b_1}$
repeat
  Obtener un ejemplo del tipo $(u, p_i, p_{i+1}, p_{i+2})$
  $loss = NCE(((\boldsymbol{D}_u \oplus (\boldsymbol{W}_{p_i} + \boldsymbol{W}_{p_{i+1}}))\boldsymbol{V}) + \boldsymbol{b_1}, p_{i+2})$
  $\boldsymbol{D} \leftarrow \boldsymbol{D} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{D}}\right]$
  $\boldsymbol{V} \leftarrow \boldsymbol{V} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{V}}\right]$
  $\boldsymbol{b_1} \leftarrow \boldsymbol{b_1} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{b_1}}\right]$
until *criterio de parada*
return $\boldsymbol{D}$

4.3.1.2. Distributed Bag Of Words (D2V-DBOW)

Otra forma mucho más simple de aprender la codificación de un documento se puede llevar a cabo sin tener en cuenta el orden que poseen las palabras dentro del mismo. Esto es lo que proponen los autores en este segundo método, donde simplemente se tiene en cuenta las palabras que aparecen en el mismo.

La arquitectura de la red que se ha de emplear en este caso es mucho más simple que la utilizada en el caso anterior, esta se detalla en la Figura 8 y se compone de los siguientes elementos:

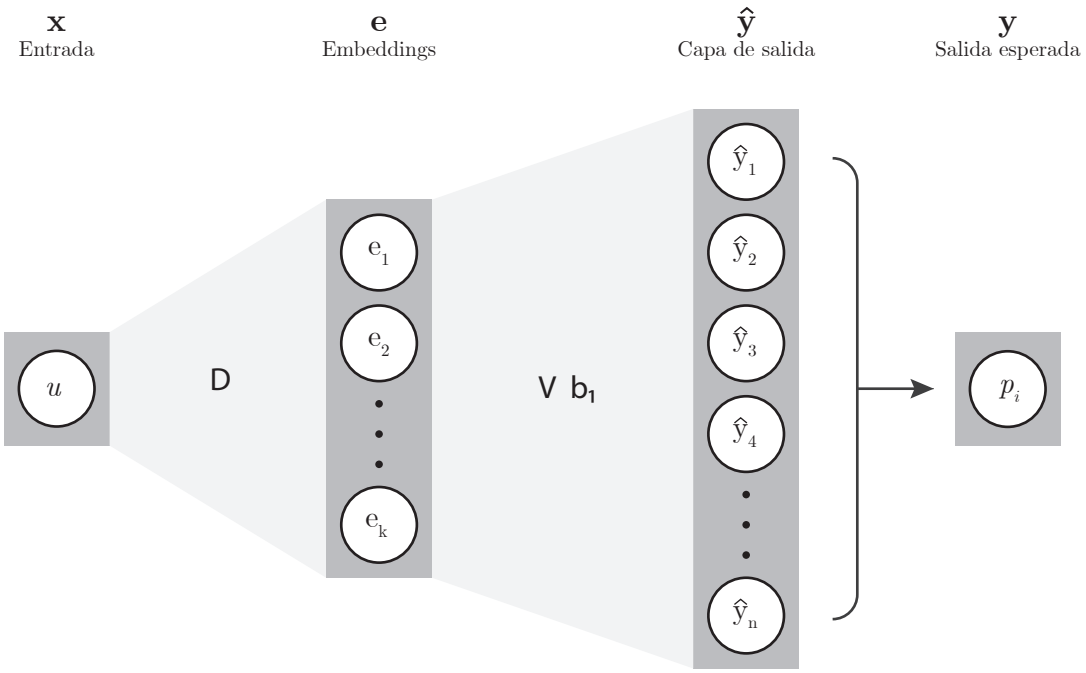

Figura 8 – Estructura del método Doc2Vec-DBOW

- **Capa de entrada (x):** En esta capa se introducirá el ID de un documento.
- **Capa de embeddings (e):** Se obtienen los embeddings de cada documento.
- **Capa de salida (ŷ):** Capa que contiene la salida del modelo (del tamaño de las palabras).
- **Salida esperada (y):** Esta capa contiene el ID real de la palabra (que se encuentra en el documento de entrada).

En este caso todas las matrices se aprenden al realizar el entrenamiento y no se usan matrices aprendidas en tareas previas como sucedía previamente con los embeddings del W2V.

Dado $\mathcal{U} = \{u_1, u_2, u_3 \ldots\}$ conjunto de documentos y $\mathcal{P}_u = \{p_1, p_2, p_3 \ldots\}$ el conjunto de palabras del documento u, se entrenará el modelo con tuplas del tipo

$$\mathcal{D}_{d2v-dbow} = \{(u, p_i) : u \in \mathcal{U}, p_i \in \mathcal{P}_u\} \ . \tag{17}$$

Con cada uno de los ejemplos, el modelo transforma la entrada en la salida aplicando

$$\hat{\boldsymbol{y}} = \boldsymbol{D}_u \boldsymbol{V} + \boldsymbol{b_1} \tag{18}$$

donde $\boldsymbol{D}_u$ corresponde a la columna u de la matriz $\boldsymbol{D}$.

La función de pérdida

$$loss = NCE(\hat{\boldsymbol{y}}, p_i) \tag{19}$$

en este problema se calcula empleando una vez más el método Noise Contrastive Estimation.

Esta loss se minimizará mediante GD donde, en cada iteración, se actulizan los valores de $\boldsymbol{D}$,$\boldsymbol{V}$ y $\boldsymbol{b_1}$ de la siguiente forma:

$$\boldsymbol{D} \leftarrow \boldsymbol{D} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{D}}\right] \qquad \boldsymbol{V} \leftarrow \boldsymbol{V} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{V}}\right] \qquad \boldsymbol{b_1} \leftarrow \boldsymbol{b_1} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{b_1}}\right] \ . \tag{20}$$

A modo de resumen, el Algoritmo 4 detalla todo el proceso al completo.

Algoritmo 4 Entrenamiento del D2V para el método DBOW.

input: $\mathcal{D}_{d2v-dbow}$, $\gamma > 0$
assign valores aleatorios (distribución gaussiana) a $\boldsymbol{D}$, $\boldsymbol{V}$ y $\boldsymbol{b_1}$
repeat
    Obtener un ejemplo del tipo (u, p_i)
    $loss = NCE(\boldsymbol{D}_u \boldsymbol{V} + \boldsymbol{b_1}, p_i)$
    $\boldsymbol{D} \leftarrow \boldsymbol{D} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{D}}\right]$
    $\boldsymbol{V} \leftarrow \boldsymbol{V} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{V}}\right]$
    $\boldsymbol{b_1} \leftarrow \boldsymbol{b_1} - \gamma \left[\frac{\partial loss}{\partial \boldsymbol{b_1}}\right]$
until *criterio de parada*
return $\boldsymbol{D}$

Como resultado de este proceso, ya sea utilizando el primer método (D2V-DM) o el segundo (D2V-DBOW) se obtiene una matriz de embeddings donde se representa cada uno de los documentos en un nuevo espacio de dimensión m.

5. Conjunto de datos

Tras analizar diversos conjuntos de datos para llevar a cabo la experimentación, se ha escogido el dataset *"Last.fm Dataset - 1K users"* proporcionado por [18]. Los criterios que se han tenido en cuenta para llevar a cabo la selección han sido diversos:

- El conjunto de datos ha de poseer información de los usuarios.
- Para cada uno de los usuarios, se ha de conocer las canciones reproducidas.
- Cada una de las reproducciones realizadas ha de poseer una fecha y hora de inicio.
- El conjunto de datos ha de abarcar una extensa ventana temporal.

El dataset seleccionado contiene datos extraídos de la web Last.fm [19] mediante el uso de su API. Además de cumplir con todas las exigencias, este conjunto de datos ha sido previamente utilizado por otros investigadores [18] los cuales han realizado un filtrado previo de los mismos.

La estructura del conjunto de datos está formada por dos ficheros donde cada columna se separa de la siguiente mediante un tabulador. Los ficheros se denominan *userid-timestamp-artid-artname-traid-traname.tsv* y *userid-profile.tsv* conteniendo el primero de ellos todas las reproducciones de todos los usuarios y el segundo los datos de cada uno de los estos.

De cada una de las reproducciones se conoce:

- **userid:** Id del usuario (relacionado con la otra tabla)
- **timestamp:** Fecha y hora de inicio de la reproducción
- **musicbrainz-artist-id:** ID único que representa al artista (no disponible en todos)
- **artist-name:** Artista de la canción reproducida
- **musicbrainz-track-id:** ID único que representa a la canción (no disponible en todas)
- **track-name:** Nombre de la canción reproducida

En cuanto a los usuarios, la información que se tiene de cada uno es:

- **userid:** Id del usuario (relacionado con la otra tabla)
- **gender:** Sexo del usuario (m, f o no disponible)
- **age:** Edad (no siempre disponible)
- **country:** País de origen (no siempre disponible)
- **signup :** Fecha de registro (no siempre disponible)

5.1. Estadística descriptiva del conjunto

Los objetivos de llevar a cabo un análisis descriptivo del conjunto de datos son diversos. Conocer los datos sobre los que se va a trabajar y poder tomar decisiones a la hora de realizar una limpieza de los mismos serían los propósitos principales.

El conjunto contiene información sobre **992 usuarios** los cuales han realizado un total de **19.150.868 escuchas**. El número de canciones únicas (sin contar las repeticiones) es de **1.500.649**.

En la Tabla 4 se puede apreciar como disminuye el conjunto de datos en relación al número de escuchas de cada canción, es decir, cómo quedaría el conjunto de datos si solo se cuentan aquellas canciones que se han reproducido X veces o más.

Reproducciones (veces)	**Número de canciones**	**Número de escuchas**
1 o más	1.500.649	19.150.868
10 o más	312.985	16.036.509
20 o más	172.493	14.141.937
30 o más	117.195	12.825.822
40 o más	87.712	11.824.454
50 o más	69.653	11.028.039
60 o más	57.207	10.353.244
70 o más	48.334	9.782.520
80 o más	41.705	9.290.718
90 o más	36.439	8.846.341
100 o más	32.353	8.461.834

Tabla 4 – Tamaño del dataset en función del número de escuchas

Dado que interesa conocer el comportamiento de los usuarios a lo largo del tiempo, se puede ver que existen datos desde el año 2005 hasta el año 2009, siendo el 2008 el año con más reproducciones por parte de los usuarios. Esta información se puede ver reflejada en la Figura 9 donde para cada año se muestra el total de reproducciones.

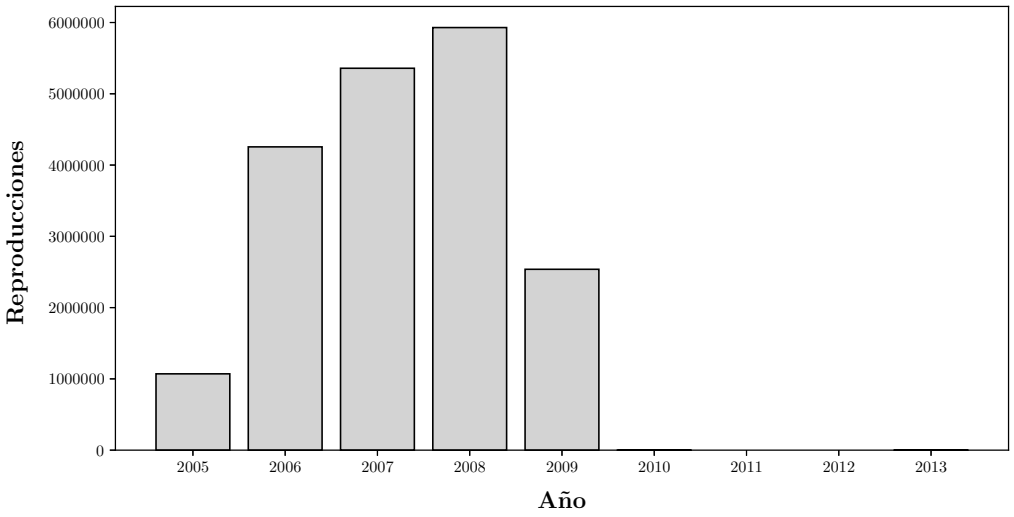

Figura 9 – Reproducciones anuales

Se puede apreciar que el gráfico anterior representa años en los que aparentemente no existen reproducciones pero, en realidad, esto se debe a la existencia de una reproducción en el año 2010 y otra en el 2013 inapreciables dada la escala del eje de ordenadas.

Analizando el número de reproducciones existentes para cada uno de los usuarios, se ha obtenido la Tabla 5. De esta tabla se puede extraer que, el usuario con más reproducciones, es el **user_000949** mientras que el que menos escuchas a realizado es el **user_000332**.

	Reproducciones	**Usuario**
Máximo	183.103	user_000949
Media	19.305	
Mediana	11.551	
Desviación estándar	23.210	
Mínimo	2	user_000332

Tabla 5 – Resumen de reproducciones por usuario

Con el fin de conocer la distribución de las reproducciones de los usuarios, se ha generado un histograma representado en la Figura 10.

En este se puede ver de una forma clara que, más de la mitad de los usuarios, poseen entre 0 y 12206 reproducciones en su historial. En cuanto al resto se usuarios, la mayoría se encuentran entre 12206 y 97648 reproducciones.

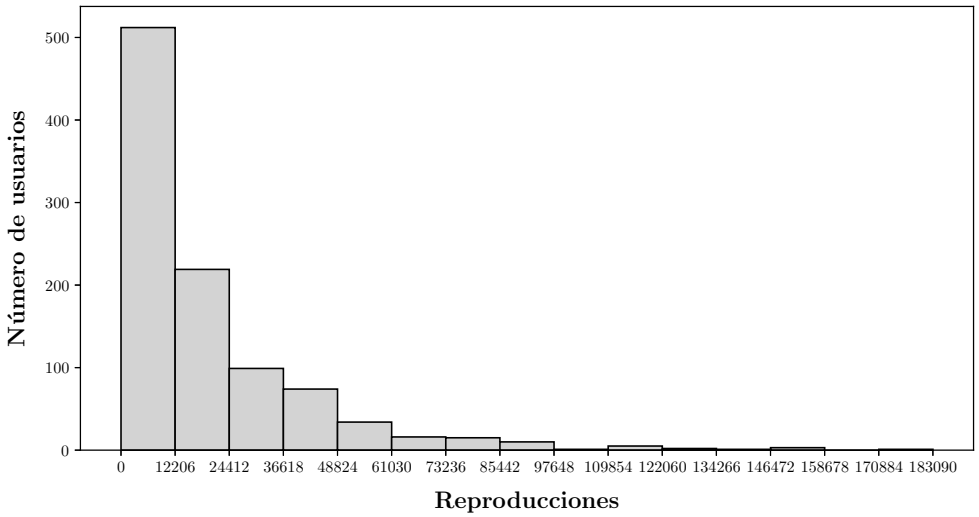

Figura 10 – Histograma de reproducciones por usuario

Otro apartado a tener en cuenta para el objetivo principal de la investigación es conocer el número de días disponibles para cada usuario, es decir, ver cuantos días ha utilizado cada uno de los usuarios la plataforma. Un resumen de estos datos se puede ver en la Tabla 6 junto con un gráfico que representa la distribución de los mismos en la Figura 11.

	Días	**Usuario**
Máximo	1457	user_000833
Media	394	
Mediana	323	
Desviación estándar	323	
Mínimo	2	user_000332, user_000815, user_000852, user_000856, user_000895

Tabla 6 – Resumen de días disponibles por usuario

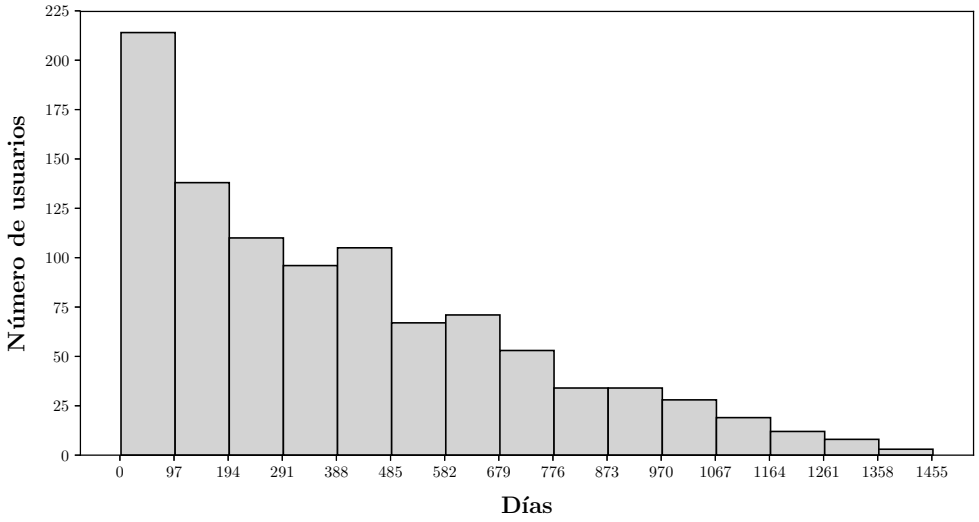

Figura 11 – Histograma de días por usuario

En este segundo histograma se puede observar una gran variedad en los datos. A diferencia de la distribución previamente analizada (Figura 10), en esta podemos ver una división de los datos más uniforme y no extremadamente elevada en los primeros segmentos como ocurría en la anterior.

Relacionando las métricas anteriores, se puede obtener un valor para cada usuario que represente el número de canciones medio diario con el fin de conocer si existen usuarios con muchas reproducciones concentradas en un día (posiblemente usuarios falsos o erróneos) o usuarios que poseen una sola reproducción al día.

	Canciones/Día	**Usuario**
Máximo	337	user_000808
Media	45	
Mediana	35	
Desviación estándar	36	
Mínimo	1	user_000332

Tabla 7 – Resumen del ratio canciones/día por usuario

Como se puede observar en la tabla anterior (Tabla 7) el usuario que más canciones diarias posee es el número 000808 con 337. Por otro lado, el usuario 000332 posee el valor más bajo de todo el conjunto (1 canción al día).

Tanto el valor máximo como el mínimo son valores que se pueden considerar razonables dado que, si se fija como duración media de una canción 3 minutos ($337\ x\ 3 = 1011\ min \simeq 16\ horas$) y se tiene en cuenta que, muchas de estas reproducciones no son completas (el usuario pasa la canción sin escucharla) el valor temporal puede ser razonable.

5.2. Limpieza de datos

Una vez analizado y comprendido el conjunto de forma estadística, es necesario llevar a cabo una limpieza de los datos. El objetivo de esta limpieza es el de mejorar o facilitar la tarea de aprendizaje eliminando el posible ruido y datos que se puedan considerar erróneos.

Para ello se realizará otro análisis sobre los datos, pero esta vez a nivel "visual" de tal forma que permita ver campos erróneos o innecesarios. Como resultado de este segundo análisis, se puede deducir que, tanto el campo *musicbrainz-artist-id* como el campo *musicbrainz-tack-id* de la tabla de reproducciones son innecesarios dada su ausencia en gran parte de los casos. Como representación única de cada una de las canciones se utilizará el nombre del artista concatenado al nombre de la canción (ambos separados por un guión).

Al realizar la concatenación de campos anterior se observa otro problema representado en la Tabla 8. Para la misma canción del mismo artista, existen pequeñas variaciones en los títulos de las canciones que provocan que estas sean consideradas diferentes. En este caso concreto, la diferencia estriba en el apostrofe que acompaña a la ene mayúscula (destacado en negrita).

Nombre de canción
Hello? Is This Thing On? (Thomas **N Eric'S** Rub And Tug Throwdown)
Hello? Is This Thing On? (Thomas **N' Eric'S** Rub And Tug Throwdown)

Tabla 8 – Canciones iguales de distinto nombre

Con el fin de paliar este comportamiento, la solución planteada es realizar un diccionario con, para cada una de las canciones, un ID asociado único de tal forma que, si se encuentra una canción con el título muy similar a otra, se les asigne el mismo ID (Tabla 9).

Nombre de canción	**ID**
Hello? Is This Thing On? (Thomas **N Eric'S** Rub And Tug Throwdown)	15
Hello? Is This Thing On? (Thomas **N' Eric'S** Rub And Tug Throwdown)	15

Tabla 9 – Diccionario (canción, ID)

Este proceso se realiza recorriendo las canciones de cada artista y comparando la similitud entre los nombres de las mismas. En caso de que los nombres de 2 pistas coincidiesen (+ 95 % de similitud) se les asignaría el mismo ID. La similitud entre dos nombres se obtiene utilizando el método *SequenceMatcher.ratio()* de la libreria *difflib* [20] de *Python*.

A modo aclaratorio se detalla en el Algoritmo 5 el proceso completo a realizar para obtener un diccionario del tipo nombre de canción, ID.

Algoritmo 5 Eliminar canciones con nombres duplicados.

```
input: Lista total de canciones (C)
d = Diccionario vacío
group = Agrupar C por artista
for singer, songs in group do
    Comparar cada canción con el resto (del artista)
    if Títulos similares then
        Añadir ambas a d con mismo id
    else
        Añadir ambas a d con distinto id
    end if
end for
return d
```

Teniendo en cuenta todos los análisis previos, la limpieza de datos que se realizará será la siguiente:

1. Combinar canciones con títulos similares.
2. Eliminar reproducciones desde 2010 a 2013 (Solo 2 items; ver Figura 9).
3. Eliminar reproducciones asociadas a canciones con menos de 10 escuchas.
4. Eliminar reproducciones de usuarios que posean menos de 100 escuchas (32 usuarios).

La selección de estos parámetros y no otros tiene como fin eliminar el máximo ruido posible (puntos 1, 2 y 3) y facilitar el aprendizaje de perfiles de usuario (punto 4) principalmente.

Como resultado de este proceso, se han descartado 3.115.739 reproducciones quedando finalmente un total de **16.035.129**. En cuanto a canciones, trabajaremos con **311.946**.

6. Experimentación

La experimentación se dividirá en 4 partes principalmente. Primero se separarán lo datos disponibles, después se obtendrá un embedding para cada una de las canciones existentes, a continuación se crearán los perfiles de cada usuario a partir de estos embeddings y finalmente se realizarán y evaluarán recomendaciones.

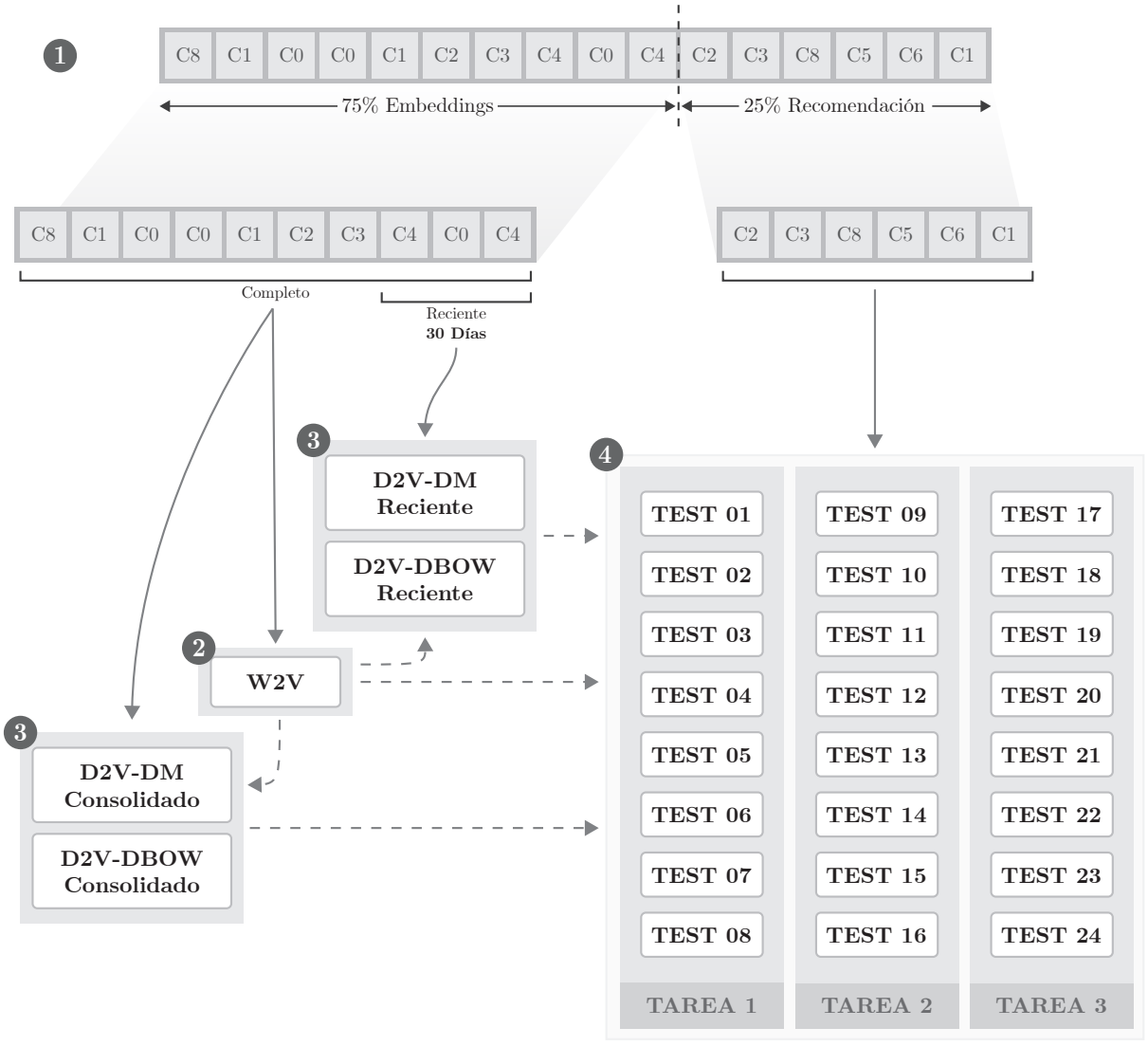

Figura 12 – Flujo global de la experimentación a realizar. Las flechas discontinuas indican una dependencia de embeddings y el resto una dependencia de datos.

En la Figura 12, se indican cada una de las partes citadas anteriormente así como la dependencia de datos y embeddings entre cada una. A lo largo de las secciones de este apartado se detallarán todas las partes del flujo-grama explicando desde la división de los datos hasta las tareas de recomendación final donde se evaluarán los embeddings aprendidos.

6.1. División de datos

En esta sección inicial se detallará el procedimiento de división de datos correspondiente al paso número 1 de la Figura 12.

Tras haber realizado todo el proceso previo de limpieza, nuestro conjunto de datos posee una forma muy similar a la indicada en la siguiente figura:

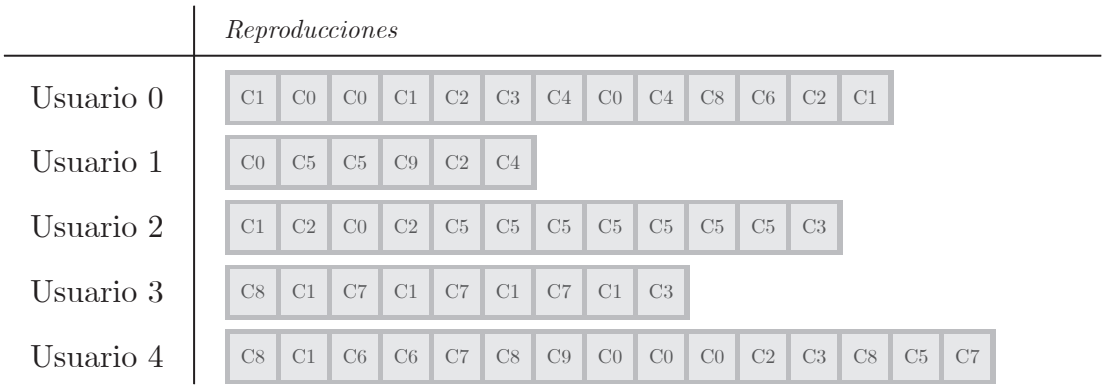

Figura 13 – Ejemplo de conjunto de datos

Este ejemplo ilustrativo solamente posee 5 usuarios y 10 canciones, pero en nuestro caso estos números son mucho mayores. Para poder realizar una tarea de recomendación en el futuro, es necesario dividir este conjunto de datos en otros de menor longitud.

En nuestro caso, se crearán 2 subconjuntos de menor longitud, siendo estos:

- **Conjunto para embeddings:** Se utilizará para aprender los embeddings de las canciones y los perfiles de usuario. Tendrá un tamaño que representa el 75 % del conjunto completo.
- **Conjunto para recomendación:** Se utilizará para realizar las tareas de recomendación finales a partir de los embeddings aprendidos. Lo formarán el 25 % de reproducciones restantes.

Utilizando como ejemplo la figura anterior (Figura 13) y aplicando los porcentajes indicados previamente, se obtendría una separación como la siguiente:

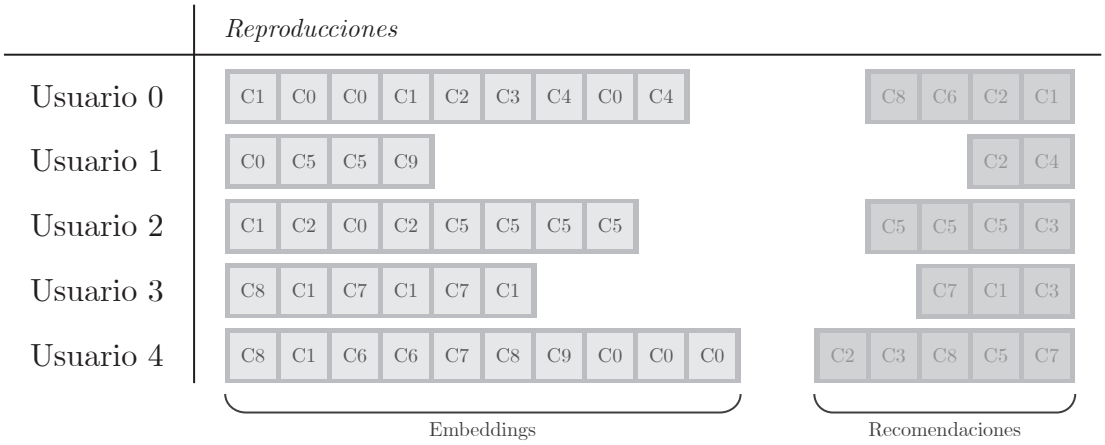

Figura 14 – Conjunto de datos dividido

Destacar que la separación se realiza de forma "vertical" y no al contrario dado que se pretende simular un caso de recomendación real donde los usuarios escucharon canciones hasta una fecha. Realizando la separación en el otro sentido solamente se estarían eliminado usuarios, lo cual no interesa en este caso concreto.

Se ha de tener en cuenta que no se realiza ningún tipo de mezcla de los datos. Realizar este barajado provocaría desordenar las reproducciones de los usuarios alterando por completo el aprendizaje.

A partir de este punto, se aprenderán los embeddings de las canciones y los perfiles de usuario utilizando los datos destinados para ello. Una vez conocidos dichos perfiles de usuario, se realizarán recomendaciones haciendo uso de los datos restantes con el fin de evaluar su utilidad.

6.2. Aprendizaje de embeddings de canciones

Esta parte de la experimentación correspondiente con el paso número 2 de la Figura 12, tiene como objetivo obtener los embeddings de cada una de las canciones empleando el Word2Vec. Un embedding (Figura 2) no es más que una representación de un objeto (una canción en este caso) en una nueva dimensión y que se ha obtenido como resultado de una función extraída de una tarea previa de aprendizaje.

Antes de poder realizar el entrenamiento del modelo Word2Vec es necesario codificar los datos de una forma adecuada así como realizar un ajuste de hiperparámetros probando diferentes configuraciones.

6.2.1. Codificación de canciones

Partiendo de los datos obtenidos en la Sección 5.2 es necesario realizar una codificación de cada una de las canciones disponibles a un formato que permita utilizar cada una de ellas como entrada del modelo Word2vec.

Para llevar a cabo esta tarea se utilizará el método *One-Hot* explicado previamente (Sección 4.1) donde la longitud n del vector es igual al número total de canciones (únicas).

ID Canción	2	3	1	0	5	4
One-Hot	0	0	0	1	0	0
	0	0	1	0	0	0
	1	0	0	0	0	0
	0	1	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮
	0	0	0	0	1	0

Vector One-Hot de n componentes (canción 3)

Tabla 10 – Codificación de canciones mediante el método OneHot

Como resultado de este proceso, se obtiene una matriz sparse de dimensión (Número de ejemplos × Número de canciones) donde cada una de las filas contendría un vector como los representados en la Tabla 10. Es esta figura solo se representan las 6 primeras canciones pero el método de codificación sería equivalente para los items restantes.

6.2.2. Codificación de ejemplos

Con el conjunto de canciones codificado en formato *One-Hot*, el siguiente paso es crear los ejemplos con los que se entrenará el modelo Word2Vec. Un ejemplo no es más que cada uno de los casos que se utilizarán para entrenar el modelo y obtener los embeddings. En función del método de generación o aprendizaje de embeddigs utilizado (Sección 4.2.2) será necesario generar un tipo de ejemplos u otro.

El Skip-gram será el método empleado en este caso dados sus buenos resultados [13], por tanto cada uno de los ejemplos de entrenamiento han de seguir el siguiente patrón:

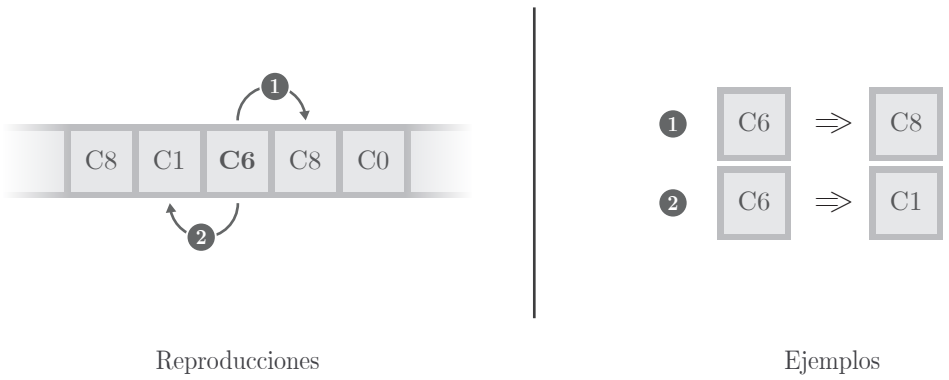

Figura 15 – Creación de ejemplos para entrenar el modelo utilizando Skip-gram con ventana=1

La figura anterior representa como se generaría cada uno de los ejemplos de entrenamiento (parte derecha) a partir de la lista de reproducciones[4] (izquierda) y un tamaño de ventana = 1.

Los ejemplos en este caso serían los pares formados por la canción de entrada del modelo ($\mathbf{c}_6$) y las canciones que se desea que este retorne cuando esté correctamente entrenado. Con el tamaño de ventana indicado, se generarían por tanto los pares ($\mathbf{c}_6$, $\mathbf{c}_8$) y ($\mathbf{c}_6$, $\mathbf{c}_1$).

Realizando este procedimiento para cada una de las reproducciones existentes resultan

$$(n - (w * 2)) * (w * 2) \quad \forall\ n\ \geq (w * 2) + 1 \tag{21}$$

ejemplos, donde n sería el número de reproducciones y w el tamaño de la ventana.

El tamaño de la ventana representa al número de ítems que rodean a la canción por cada uno de los extremos (anterior y posterior), por tanto, al aumentar el tamaño de la ventana, aumenta la influencia que poseen las canciones del contexto en el embedding de la canción actual.

Para la resolución de este problema en particular, se ha estimado como óptimo un valor de ventana de 2. Este valor implica que el embedding de cada canción se generará utilizando 4 canciones de su contexto (las 2 anteriores y las 2 posteriores). Utilizando este valor, los ejemplos asociados a aprender el embedding de la canción ($\mathbf{c}_6$) de la Figura 16 serían los pares ($\mathbf{c}_6$, $\mathbf{c}_8$), ($\mathbf{c}_6$, $\mathbf{c}_1$), ($\mathbf{c}_6$, $\mathbf{c}_8$) y ($\mathbf{c}_6$, $\mathbf{c}_0$).

[4] La lista de reproducciones contiene la concatenación de todas las reproducciones de usuarios en una única lista, uniendo el final de un usuario con el inicio del siguiente.

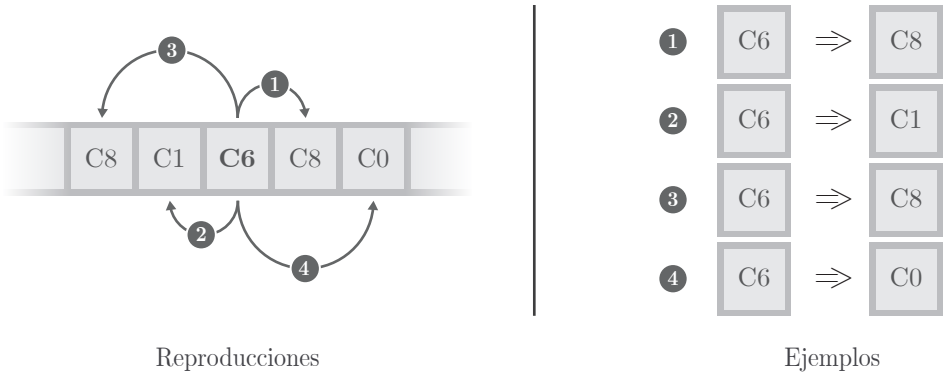

Figura 16 – Creación de ejemplos para entrenar el modelo utilizando Skip-gram con ventana=2

Teniendo en cuenta que en el ámbito de las canciones se puede cambiar rápidamente de contexto (escuchar 1 canción de un genero y la siguiente de otro completamente diferente), no tiene sentido escoger una ventana mayor. Al aumentar el tamaño de la ventana se estaría potenciando la relación entre canciones más alejadas en el tiempo (pasado o futuro) con la canción actual y en este caso tal relación no suele existir.

6.2.2.1. Partición de ejemplos

Una vez creado el conjunto de ejemplos, estos se han de dividir en tres subconjuntos de menor dimensión:

- **Conjunto de entrenamiento:** Se utilizará para entrenar el modelo Word2Vec y tendrá un tamaño que representa el 98 % del conjunto completo
- **Conjunto de validación:** Este subconjunto se utilizará para ajustar los hiperparámetros del modelo con el fin de minimizar el valor de la función de pérdida. Su tamaño corresponde con el 1 % de los ejemplos restantes.
- **Conjunto de test:** Unicamente se utilizará para evaluar el resultado final del modelo. Está formado por los ejemplos no utilizados en ninguno de los datasets anteriores.

Antes de llevar a cabo esta división en varios conjuntos, se realiza un mezclado aleatorio de todos los ejemplos disponibles con el fin de aumentar la diversidad en los datos que formen cada uno de los subconjuntos.

Destacar que, tanto el conjunto de validación como el de test, no se utilizarán en ningún caso para entrenar al modelo, simplemente para optimizarlo y evaluarlo.

La optimización o ajuste de hiperparámetros no se realiza con el propio subconjunto de entrenamiento para evitar un sobreajuste[5] del modelo.

El objetivo de utilizar un subconjunto específico para test es el de evaluar el modelo con un conjunto de datos desconocidos para las etapas de entrenamiento y optimización obteniendo un resultado final lo más imparcial posible.

[5]El sobreajuste u overfitting de un modelo se aprecia cuando este solo reporta buenos resultados con los datos de entrenamiento (y no con datos nuevos o desconocidos).

6.2.3. Optimización del Word2Vec

Tas realizar varias ejecuciones de prueba con el modelo y los datos obtenidos en pasos anteriores se han observado diversos problemas relacionados principalmente con el rendimiento:

- La carga de los ejemplos de entrenamiento es muy costosa en términos de memoria RAM y de tiempo (aproximadamente 60.000.000 ejemplos formados cada uno por 2 vectores *One-Hot* de dimensión superior a 300.000).
- La memoria de la GPU no soporta, una vez cargados los datos, introducir todo el subconjunto de entrenamiento al completo para llevar a cabo el entrenamiento.
- La función de pérdida utilizada en el modelo (Softmax) es muy costosa dado el alto número de canciones (clases) existentes.

Con el fin de paliar en la mayor medida posible cada una de las problemáticas citadas, se han realizado diversas optimizaciones sobre el modelo utilizado. A continuación se describirán cada una de ellas.

6.2.3.1. Exclusión de *One-Hot*

Dada la alta dimensionalidad del conjunto de ejemplos se ha decidido almacenar solamente, para cada canción, el ID que la representa y no la codificación *One-Hot* de la misma. Este cambio supone pasar de cargar una matriz de 40 billones de elementos a cargar aproximadamente 128 millones.

Este cambio en los datos de entrada implica realizar modificaciones en el modelo W2V con el fin de conservar el funcionamiento inicial. Antes de realizar estos cambios se analizará el comportamiento de la ecuación (4) que transforma el vector *One-Hot* de la entrada en el vector de embeddings.

A modo de ejemplo, en un caso real donde se tiene el vector *One-Hot* de entrada $\mathbf{x}$ (representando a la palabra con $id = 4$ en un vocabulario de 5 palabras) y la matriz de pesos W con los siguientes valores:

$$\mathbf{x} = \underset{5\times1}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}} \quad W = \underset{3\times5}{\begin{bmatrix} 1 & -0.7 & 0.6 & 0.7 & -0.1 \\ 0 & 0.3 & 0.1 & 0.8 & 0.6 \\ 0.2 & 0.4 & 1 & 0.5 & -0.9 \end{bmatrix}} \tag{22}$$

se obtiene el vector $\mathbf{e}$ aplicando la ecuación (4) de la siguiente forma:

$$\underset{3\times5}{\begin{bmatrix} 1 & -0.7 & 0.6 & 0.7 & -0.1 \\ 0 & 0.3 & 0.1 & 0.8 & 0.6 \\ 0.2 & 0.4 & 1 & 0.5 & -0.9 \end{bmatrix}} \cdot \underset{5\times1}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}} = \underset{3\times1}{\begin{bmatrix} 0.7 \\ 0.8 \\ 0.5 \end{bmatrix}} \tag{23}$$

Con este resultado se puede ver que el vector de embedding **e** de tamaño 3 se obtiene simplemente extrayendo la columna de la matriz de pesos W asociada con el *id* de la canción. Por tanto, eliminando el vector *One-Hot* se puede emular este comportamiento aplicando la siguiente ecuación:

$$\mathbf{e} = \boldsymbol{W_{id}} \tag{24}$$

donde $\boldsymbol{W_{id}}$ representa la columna número *id* de la matriz de pesos. Realizando este cambio resulta el siguiente modelo:

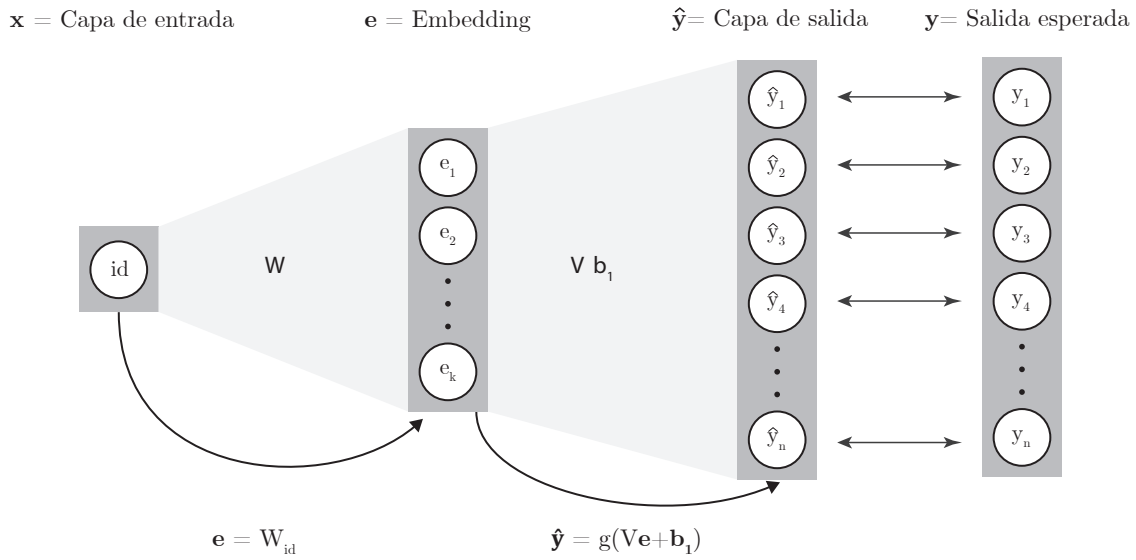

Figura 17 – Modelo resultante tras eliminar la codificación *One-Hot* de la capa de entrada

Este cambio elimina por completo la necesidad de codificar las canciones en la entrada del modelo, pero aún conserva la necesidad de realizar este proceso en el vector **y** que representa la salida esperada. Esta restricción se suprimirá en parte al aplicar la siguiente optimización.

6.2.3.2. Noise Contrastive Estimation (NCE)

La función que se utiliza por defecto para transformar la salida "en crudo" del modelo a una adecuada para un problema multiclase[6] como este es la ya citada SoftMax (Sección 4.2.1). Como se puede comprobar analizando su ecuación (7), la Softmax realiza más cálculos cuanto mayor sea el tamaño del vocabulario, siendo cada uno de estos de una cierta complejidad.

En un caso donde el "vocabulario" está formado por 311.946 canciones se ha de buscar otro método que permita realizar el mismo procedimiento con un coste computacional mucho menor. Es en este punto donde aparece el método NCE [13, 21] que transforma el problema multiclase actual en uno de clasificación binaria.

Inicialmente el problema se planteó como una clasificación multilabel, a continuación, la codificación de los ejemplos lo convirtió en un problema multiclase y finalmente se transforma en una simple clasificación binaria (ver Figura 18).

[6]El problema inicial se plantea como una clasificación multilabel dado que a partir de una canción se predicen todas las de su contexto. Debido a la estructura utilizada a la hora de crear los casos de ejemplo se puede comprobar que en realidad estamos ante un problema multiclase.

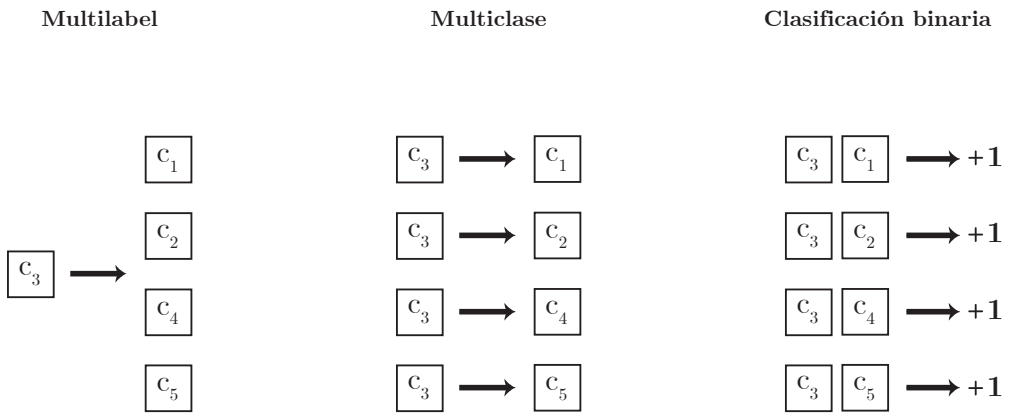

Figura 18 – Evolución del problema

Esta transformación a una clasificación binaria implica que tanto la Softmax como la Cross Entropy ya no sean útiles y necesiten ser remplazadas.

En cuanto al funcionamiento, el NCE obtiene para cada par de entrenamiento (siendo un par $(c3, c1)$ por ejemplo) un número l de pares negativos generados de forma aleatoria (siendo un par negativo $(c3, x)$ $\forall x \neq c1$).

De esta forma el clasificador aprende a distinguir los pares positivos de los negativos y aprende finalmente los embeddings asociados a cada canción. Esto produce un cambio en la filosofía del problema, pasando de predecir la siguiente canción a predecir que pares de canciones será bueno o malo.

Destacar finalmente que el NCE se encuentra optimizado para crear pares negativos con aquellas palabras que aparecen con menor frecuencia, reduciendo la probabilidad de que se genere un par negativo que en realidad sea de la clase positiva.

Con este método no es necesario codificar las canciones de la salida en formato *One-Hot*, dado que internamente gestiona las comparaciones y el cálculo de la loss. Una vez aplicados estos cambios, el modelo final sería:

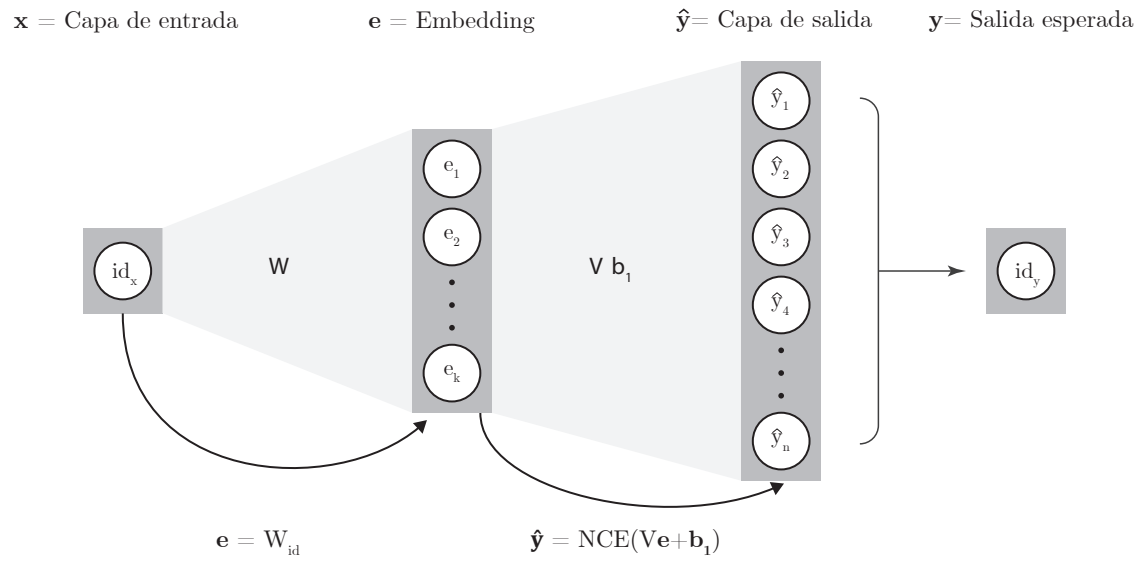

Figura 19 – Modelo W2V final

6.2.3.3. División en batches

Dadas las restricciones de memoria de GPU, las cuales no permiten introducir en el modelo todo el conjunto de datos al completo, se ha creado un método capaz de dividir el dataset en batches de menor dimensión.

Este procedimiento se llevará a cabo solamente durante la fase de entrenamiento de la red neuronal, dado que las dimensiones de los conjuntos de DEV y TEST sí permiten ser cargados en memoria.

Realizar esta división del conjunto de datos de entrenamiento soluciona de una forma sencilla el problema pero incrementa el número de hiperparámetros a optimizar en el futuro. El tamaño de cada uno de los batches es una variable con la que se puede especular y será necesario probar con varias configuraciones a fin de encontrar la óptima en nuestro caso.

6.2.4. Ajuste de hiperparámetros

El modelo con el que se va a trabajar posee un elevado número de hiperparámetros que han de ser fijados previamente para llevar a cabo el entrenamiento. Para obtener los valores que mejor se adaptan a nuestro caso se utilizará el conjunto de datos DEV (ver Sección 6.2.2.1) y sobre él se probarán distintas configuraciones siguiendo el método conocido como Grid-Search.

Grid-Search es una metodología que se utiliza para probar todas las configuraciones posibles dentro de unos valores prefijados. Para llevarlo a cabo es necesario indicar un conjunto de valores posibles para cada uno de los hiperparámetros y posteriormente se probarían todos los casos.

En nuestro caso concreto poseemos los hiperparámetros indicados en la Tabla 11 , donde se indica además los posibles valores que estos pueden tomar.

Parámetro	Valores	Descripción
emb_size	64, 128, 256, 512	Tamaño de la capa de embeddings
batch_size	256, 512, 1024, 2048	Número de ítems en cada batch
lr	0.1, 1, 10	Learning rate del modelo

Tabla 11 – Hiperparámetros a optimizar y posibles valores

Tras probar la diferentes combinaciones de hiperparámetros, e incluyendo otros prefijados en secciones anteriores (como el tamaño de la ventana), se utilizarán los siguientes valores para entrenar el modelo ya que han sido los que mejor resultado obtuvieron sobre el conjunto de DEV:

Parámetro	Valor	Descripción
emb_size	64	Tamaño de la capa de embeddings
batch_size	512	Número de ítems en cada batch
lr	0.1	Learning rate del modelo
ventana	2	Tamaño de la ventana (Skip-Gram)

Tabla 12 – Hiperparámetros y valores de entrenamiento

6.2.5. Entrenamiento

Con los valores obtenidos previamente y una vez definidas las operaciones, variables y conjuntos de datos necesarios, se pude dar paso a la fase de entrenamiento. En esta fase se utilizará el conjunto de datos de TRAIN y DEV para alimentar al modelo y obtener los embeddings asociados a cada una de las palabras.

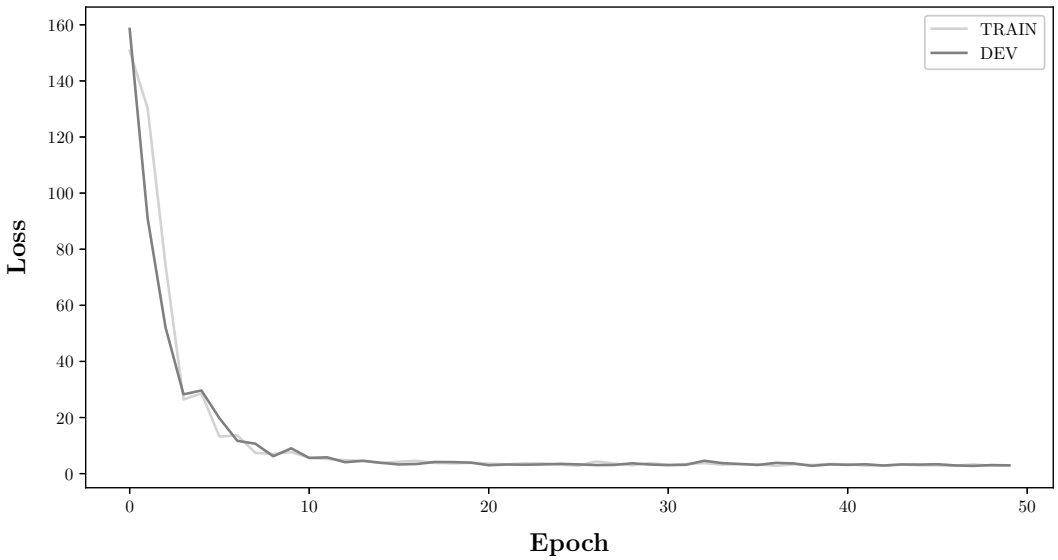

Figura 20 – Evolución de la loss de TRAIN y DEV durante el entrenamiento del W2V.

El número de epochs[7] a realizar se establece en 1500 tras detener en entrenamiento de forma manual al no producirse una mejora relevante en la loss de TEST. En la Figura 20 se puede ver una gráfica con la evolución del entrenamiento durante las 50 primeras epochs. Destacar que la escala del eje Y no permite visualmente apreciar mejoras a partir de la epoch 10 aunque estas existen realmente.

Una vez finalizado el entrenamiento del modelo, se han obtenido los siguientes resultados:

Loss en TRAIN + DEV	**Loss en TEST**
1.608	1.631

Tabla 13 – Resultados entrenamiento W2V

6.2.6. Pruebas

Con el fin de realizar una evaluación de los embeddings aprendidos por el modelo se ha buscado, para una canción dada, la canción más cercana. Este proceso se realiza simplemente calculando la distancia entre el vector que representa a la canción indicada y los vectores restantes. A continuación se muestran algunos resultados:

Canción de entrada	**Más cercana**	**Distancia**
Dire Straits - Sultans Of Swing	Dire Straits - Money For Nothing	3.8660
Ludwig Van Beethoven - Für Elise	Gustav Mahler - Symphony No.5	5.2357
Eminem - Lose Yourself	Eminem - Mockingbird	3.6801
Daft Punk - Technologic	Daft Punk - Television Rules The Nation	3.4544
R.E.M. - Losing My Religion	R.E.M. - Shiny Happy People	3.4317
Shakira - Hips Don'T Lie	Shakira - How Do You Do	6.0337
Flogging Molly - For A Dying Song	Flogging Molly - Paddy'S Lament	5.2626

Tabla 14 – Test realizados

Con el fin de poder representar los embeddings en 2 dimensiones y ver si existen agrupaciones de ítems, se ha aplicado el método t-SNE [22] para reducir las dimensiones de los embeddings (64) a 2 componentes. El resultado de representar los 2000 primeros embeddings se muestra en el siguiente gráfico:

[7] Una epoch es una vuelta completa al conjunto de datos de TRAIN a la hora de entrenar el modelo.

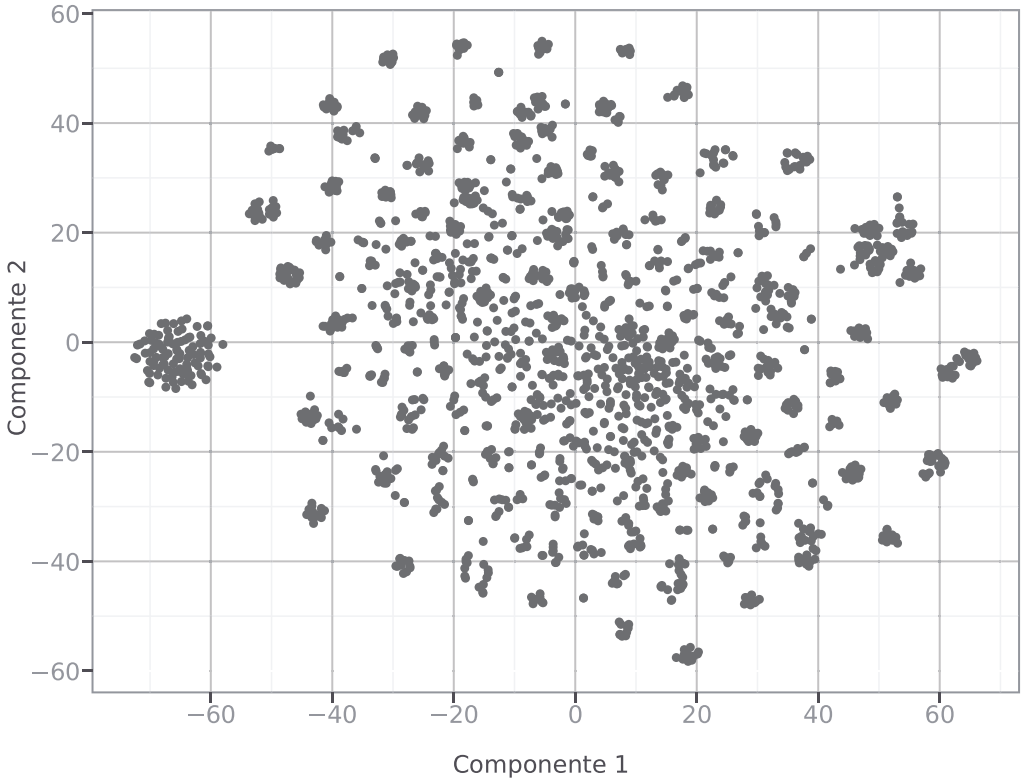

Figura 21 – Componentes 1 y 2 obtenidas del t-SNE a partir de los 2000 primeros embeddings

Los resultados se asemejan a lo esperado dado que, los ítems que se encuentran agrupados en pequeños clusters, representan a aquellas canciones que se suelen escuchar juntas o muy cercanas.

6.3. Aprendizaje de perfiles de usuario

En esta parte de la experimentación correspondiente con la fase número 3 de la Figura 12, se pretende obtener los perfiles de cada uno de los usuarios utilizando los métodos explicados previamente (Sección 4.3).

Dado que la idea principal de esta investigación es la de generar, para cada usuario, dos perfiles en vez de uno, será necesario una división previa de los datos.

6.3.1. División de datos

Para cada uno de los usuarios disponibles, se planea tener un perfil que represente su actividad reciente (denominado a partir de ahora perfil reciente) y otro perfil en representación de su actividad pasada (se denominará perfil consolidado).

La creación de estos perfiles, requiere de una separación previa de los datos de los que se dispone para esta tarea (ver Figura 22).

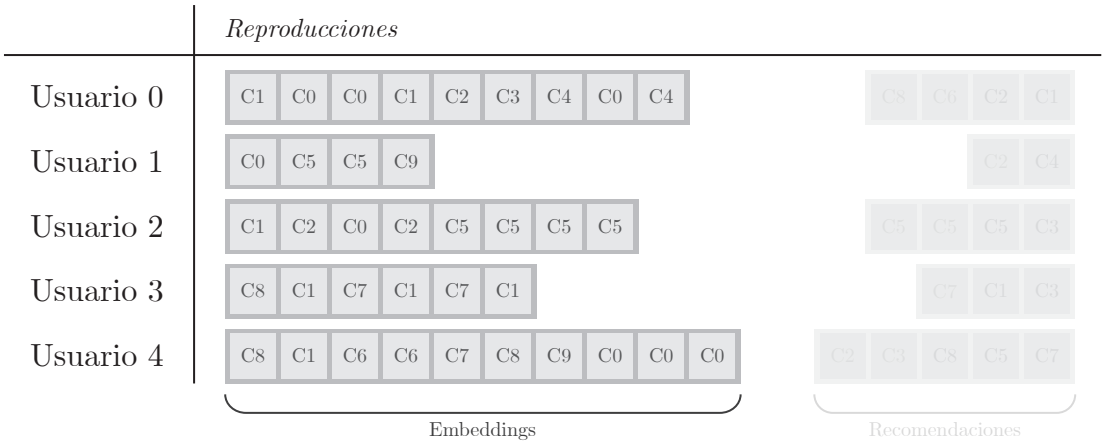

Figura 22 – Datos disponibles para el aprendizaje de embeddings

Para ello se ha decidido utilizar, para cada usuario, los datos de los últimos 30 días para el perfil reciente y el resto para el perfil consolidado (ver Figura 23).

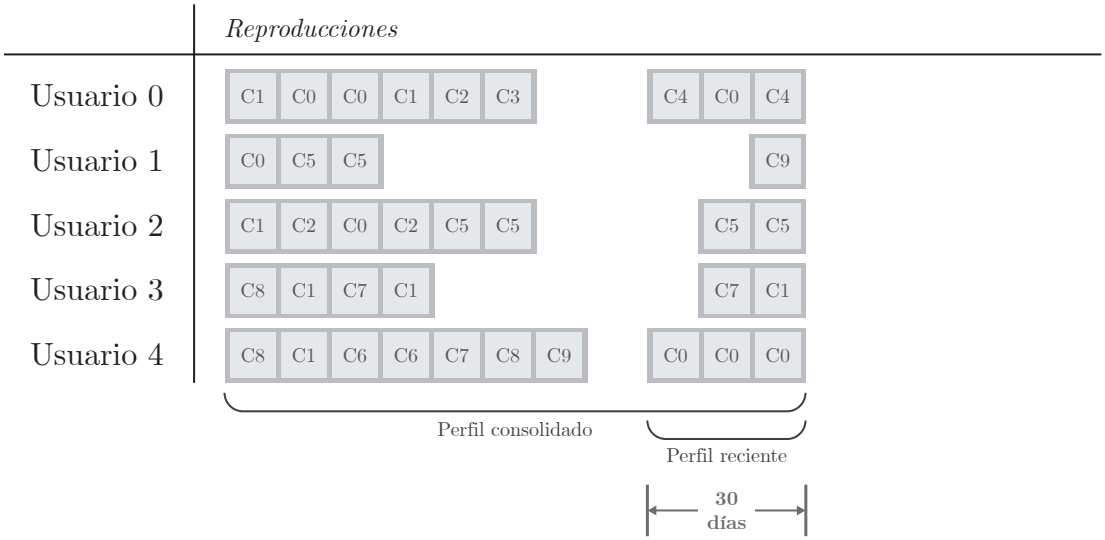

Figura 23 – Separación del conjunto en función de la fecha de las reproducciones

Esta división se ha realizado teniendo en cuenta solo aquellos usuarios que posean en estos 30 días más de 5 reproducciones. Existen 2 usuarios que no cumplen con estas restricciones, por lo que se han eliminado dada la falta de información que aportan en el problema a resolver.

6.3.2. Doc2Vec

Partiendo de lo descrito en el Apartado 4.3, cada uno de los perfiles de usuario se generará combinando el resultado de generar un perfil con el D2V-DM con el resultado del D2V-DBOW. Teniendo esto en cuenta, cada uno de los perfiles de usuario se compondrá de:

- **Perfil consolidado** Concatenación del D2V-DM con el D2V-DBOW con los datos para perfil consolidado.
- **Perfil reciente** Concatenación del D2V-DM con el D2V-DBOW con los datos para perfil reciente.

Esto implica que para generar todos los perfiles requeridos son necesarias 4 ejecuciones del método Doc2Vec.

6.3.2.1. Codificación de ejemplos (D2V-DM)

Dada la arquitectura de este método en concreto (Figura 7), se han de generar ejemplos de entrenamiento y test del tipo $(u\ , c_1\ , c_2) \Rightarrow c_3$ como se indicó en la Ecuación (13).

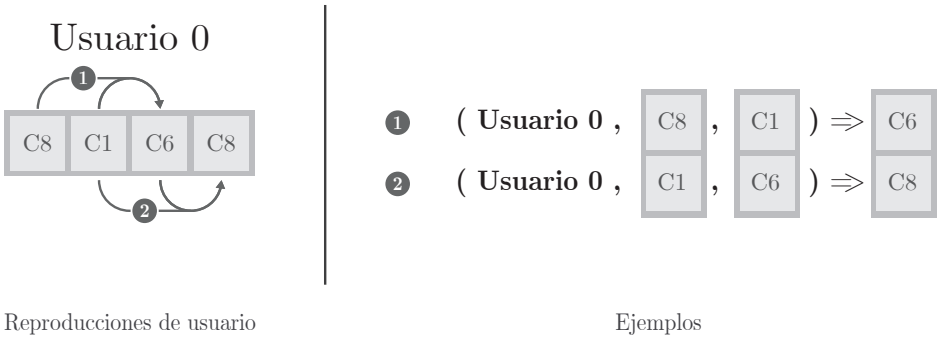

Figura 24 – Creación de ejemplos en el método D2V-DM.

En este caso existe también un tamaño de ventana, es decir, un número de canciones previas a utilizar. Este valor se ha fijado en 2 por las razones ya descritas en la Sección 6.2.2 del modelo W2V. El número total de ejemplos, utilizando este procedimiento, viene dado por la expresión,

$$n - w \qquad \forall\ n\ > w \tag{25}$$

donde n sería el número de reproducciones y w el tamaño de la ventana.

6.3.2.2. Codificación de ejemplos (D2V-DBOW)

En este otro caso, la estructura de los ejemplos es muy diferente. Según lo indicado en la Ecuación (17) los ejemplos han de ser del tipo $u\ \Rightarrow\ c_n$.

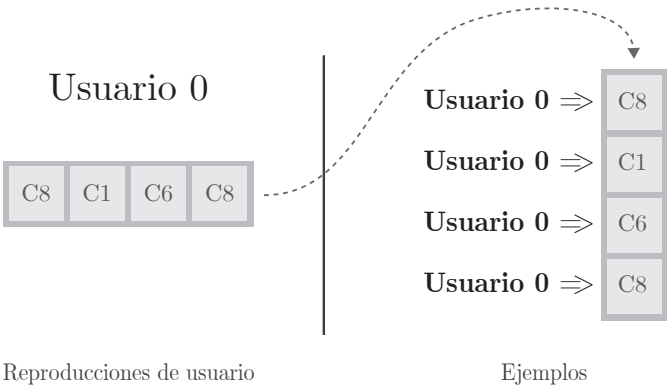

Figura 25 – Creación de ejemplos en el método D2V-DBOW.

Siguiendo este procedimiento, el número de ejemplos de cada usuario coincide su número de reproducciones.

6.3.2.3. Partición de ejemplos

Una vez generados todos los ejemplos, ya sean mediante el método 1 (D2V-DM) o utilizando el método 2 (D2V-DBOW), es necesario dividirlos en 3 subconjuntos al igual que sucedía en el caso del W2V. Estos subconjuntos serán:

- **Conjunto de TRAIN:** Utilizado para entrenar al método D2V con un tamaño que represente el 98 % del conjunto completo.
- **Conjunto de DEV:** Subconjunto utilizado para ajustar los Hiperparámetros del modelo con el fin de minimizar el valor de la función de pérdida. Su tamaño corresponde con el 1 % de los ejemplos restantes.
- **Conjunto de TEST:** Se utilizará únicamente para evaluar el resultado final del modelo. Está formado por los ejemplos no utilizados en ninguno de los datasets anteriores.

6.3.2.4. Ajuste de hiperparámetros

Partiendo de los ejemplos anteriores, se han realizado diversas pruebas utilizando el conjunto de TRAIN para entrenar al modelo y el conjunto de DEV para realizar la validación.

Tras la ejecución de estas pruebas de forma manual, se ha determinado que los mejores hiperparámetros para entrenar al modelo D2V-DM son:

Parámetro	Valor	Descripción
emb_size	64	Tamaño de la capa de embeddings
batch_size	512	Número de ítems en cada batch
lr	0.1	Learning rate del modelo
ventana	2	Tamaño de la ventana (Skip-Gram)

Tabla 15 – Hiperparámetros y valores de entrenamiento

En cuanto al D2V-DBOW, los hiperparámetros a utilizar serán:

Parámetro	Valor	Descripción
emb_size	64	Tamaño de la capa de embeddings
batch_size	1024	Número de ítems en cada batch
lr	0.1	Learning rate del modelo

Tabla 16 – Hiperparámetros y valores de entrenamiento

6.3.2.5. Entrenamiento

Puesto que existen 2 métodos diferentes para aprender perfiles (D2V-DM y D2V-CBOW) y además se quieren utilizar 2 conjuntos de datos (consolidado y reciente), se entrenarán 4 modelos con el fin de generar todos los embeddings necesarios.

A continuación se indica la evolución del entrenamiento para cada uno de los modelos junto con el valor de la loss en TRAIN y TEST de la última epoch. Inicialmente se indicarán los resultados obtenidos al utilizar los datos recientes.

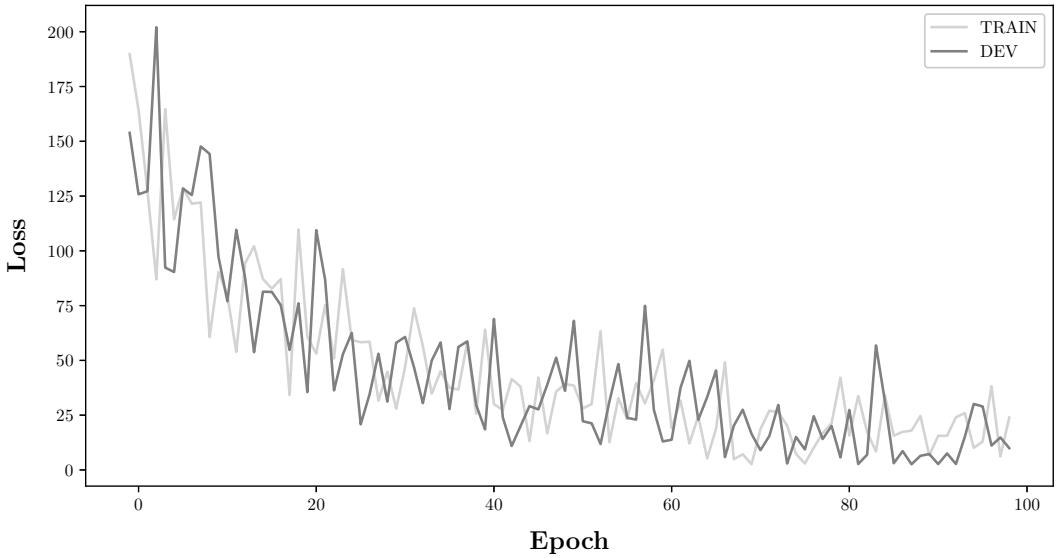

Figura 26 – Evolución del entrenamiento con el método D2V-DM para los datos recientes (100 primeras epochs).

Loss en TRAIN + DEV	**Loss en TEST**
2.081	1.993

Tabla 17 – Valor de la loss en la última epoch del método D2V-DM para los datos recientes.

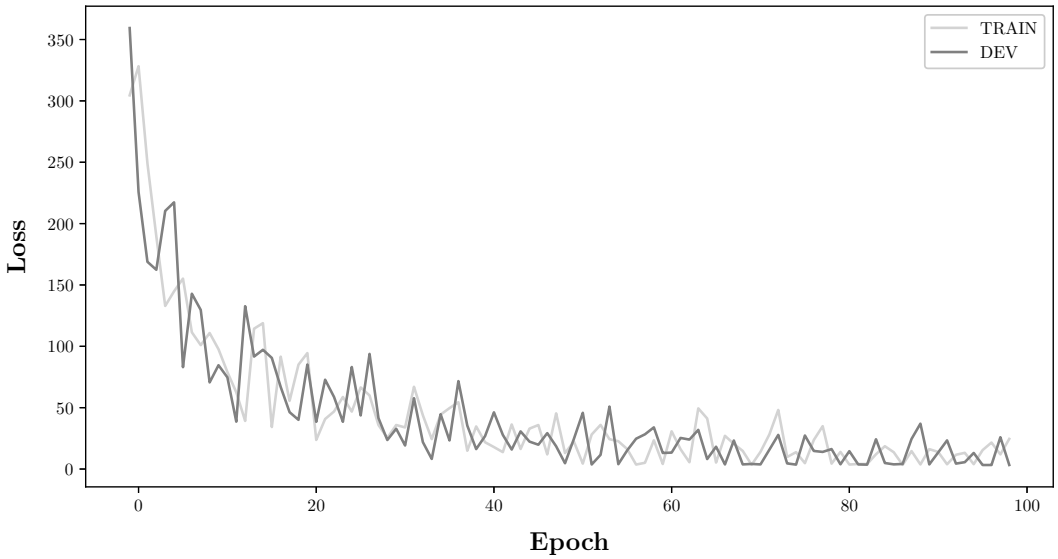

Figura 27 – Evolución del entrenamiento con el método D2V-DBOW para los datos recientes (100 primeras epochs).

Loss en TRAIN + DEV	Loss en TEST
2.547	2.697

Tabla 18 – Valor de la loss en la última epoch del método D2V-DBOW para los datos recientes.

En cuanto a los resultados de entrenamiento para los datos consolidados, la evolución parece muy estable en comparación con los resultados anteriores.

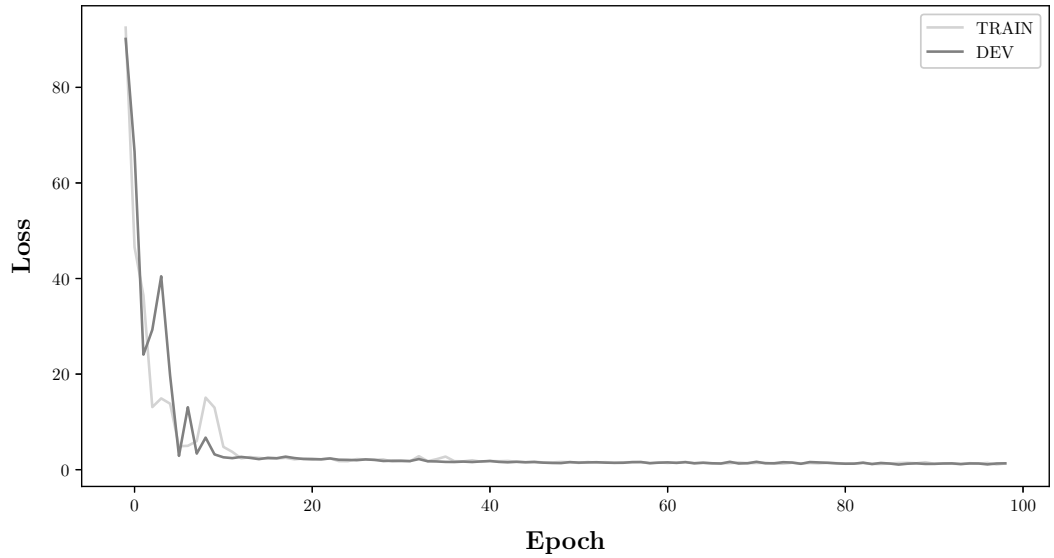

Figura 28 – Evolución del entrenamiento con el método D2V-DM para los datos consolidados (100 primeras epochs).

Loss en TRAIN + DEV	Loss en TEST
1.148	1.109

Tabla 19 – Valor de la loss en la última epoch del método D2V-DM para los datos consolidados.

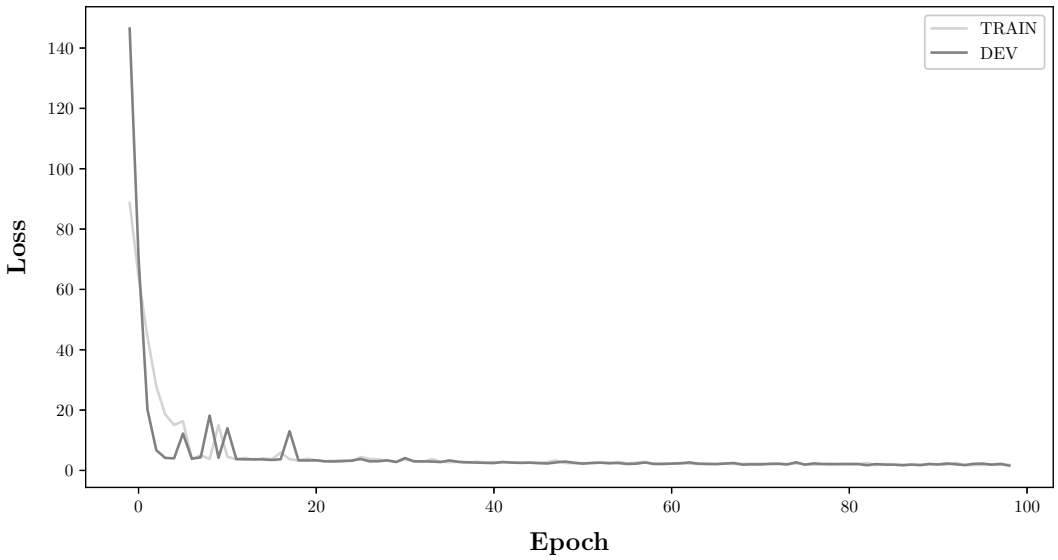

Figura 29 – Evolución del entrenamiento con el método D2V-DBOW para los datos consolidados (100 primeras epochs).

Loss en TRAIN + DEV	Loss en TEST
1.583	1.441

Tabla 20 – Valor de la loss en la última epoch del método D2V-DBOW para los datos consolidados.

Finalmente indicar que se han entrenado los modelos durante aproximadamente 1500 epochs, cuando se ha detenido de forma manual por falta de mejora en la función de pérdida del mismo.

6.4. Recomendaciones

La última sección de la experimentación (correspondiente con el apartado número 4 de la Figura 12), tiene como objetivo realizar y evaluar recomendaciones a partir de todos los embeddings aprendidos. Para ello se presentarán varias tareas de recomendación diferentes en las que se utilizarán tanto los embeddings aprendidos como las representaciones en versión *One-Hot*.

A partir de este punto se utilizarán los datos separados inicialmente para llevar a cabo las recomendaciones (Figura 30), separando este conjunto en TRAIN, DEV y TEST con unas proporciones de 70 %, 15 % y 15 % respectivamente.

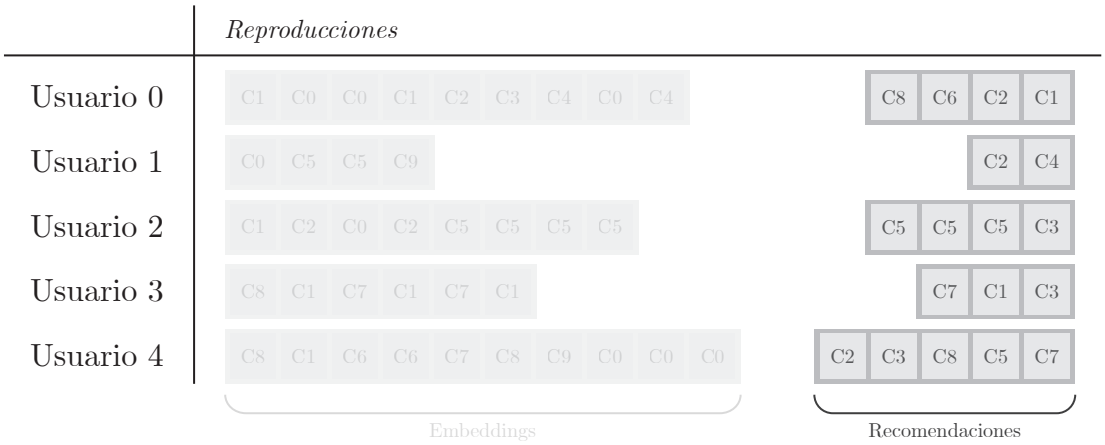

Figura 30 – Datos disponibles para realizar recomendaciones

A continuación se detallarán los propósitos de cada una de las tareas de recomendación así como las pruebas que se realizarán.

6.4.1. Tarea 1

La primera tarea de recomendación trata de responder a la pregunta *¿Escuchará el usuario* $\boldsymbol{u}$ *la canción* $\boldsymbol{c}$ *en el futuro?*. Para ello, se realizarán varias pruebas variando la representación utilizada para cada uno de los usuarios y para cada una de las canciones (ver Tabla 21).

Para cada una de las pruebas se tiene $\mathbf{e}_u$, vector representando a un usuario, y $\mathbf{e}_c$ representando una canción. El objetivo es aprender un modelo capaz de realizar una clasificación binaria. Este modelo retornará un valor muy próximo a 1 en el caso de que el usuario haya escuchado dicha canción y un valor cercano a cero en el caso contrario.

Con cada uno de los vectores anteriores, se realiza un producto escalar y se obtiene un valor sobre el que se aplica la función *sigmoide*() que permite obtener la probabilidad entre 0 y 1 de que dicho usuario haya escuchado la canción indicada.

Un punto a tener en cuenta es la necesidad de convertir los embeddings de los usuarios y de las canciones a un mismo espacio dado que, de no se así, sería imposible realizar el producto escalar entre ellos. En esta tarea, este nuevo espacio será de dimensión 32.

A partir de este punto, para representar a un usuario se poseen las opciones *One-Hot*, *D2V Consolidado*, *D2V Reciente* y la concatenación de *D2V Consolidado + D2V Reciente*. En el

caso de las canciones solo existen dos opciones, siendo estas *One-Hot* y *W2V*.

Destacar que en ambos casos se hace uso de la representación *One-Hot* con el fin de poseer un caso que sirva de referencia y permita realizar comparaciones. Combinando todas las opciones anteriores, surgen los siguientes 8 tests:

Test	Representación usuarios	Representación canciones
01	One-Hot	W2V
02	D2V Consolidado	W2V
03	D2V Reciente	W2V
04	D2V Consolidado + D2V Reciente	W2V
05	One-Hot	One-Hot
06	D2V Consolidado	One-Hot
07	D2V Reciente	One-Hot
08	D2V Consolidado + D2V Reciente	One-Hot

Tabla 21 – Representaciones de usuarios y canciones a utilizar en cada uno de los test de la tarea 1.

6.4.1.1. Creación de ejemplos

El aprendizaje de esta tarea, requiere de ejemplos positivos de TRAIN, DEV y TEST del tipo

$$(usuario,\ canción\ escuchada) \Rightarrow 1$$

así como casos negativos del tipo

$$(usuario,\ canción\ \boldsymbol{no}\ escuchada) \Rightarrow 0$$

Si no se incluyesen estos casos negativos, el modelo, probablemente retornaría ante cualquier entrada una salida positiva, es decir, para cualquier combinación de usuario y canción, nos diría que la va a escuchar en un futuro.

La generación de los casos de entrenamiento (Figura 31) no es una tarea trivial, dado que para cada uno de los usuarios se poseen datos de reproducciones que ha realizado (es decir, casos positivos) pero no negativos. Estos ejemplos negativos se han de generar de forma minuciosa, buscando para cada usuario aquellas canciones que nunca ha escuchado para evitar incurrir en contradicciones.

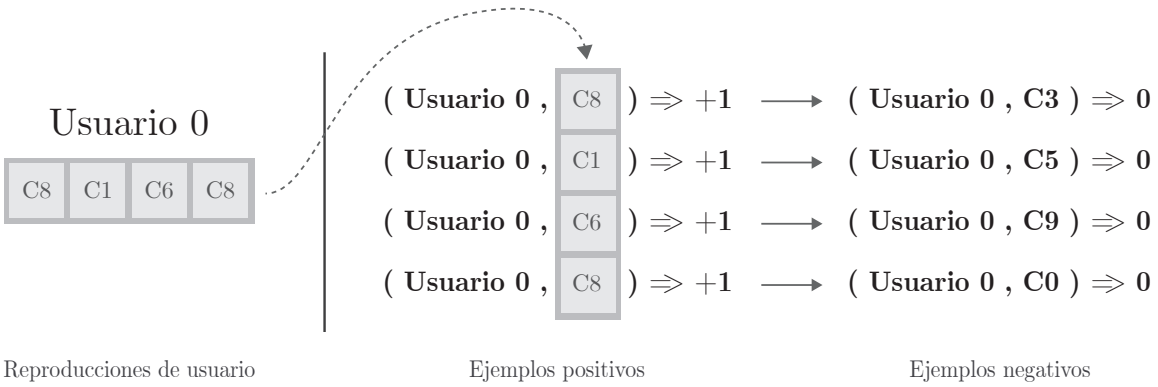

Figura 31 – Creación de ejemplos en la tarea 1.

Otro punto a tener en cuenta a la hora de generar dichos casos, es la existencia de canciones desconocidas. En el momento en que se generaron los vectores de las canciones con el W2V, se utilizaron los datos separados para aprender embeddings (Figura 14), pero en esta tarea de recomendación se utiliza el otro conjunto de datos. En este conjunto la mayoría de las veces hay canciones conocidas, pero en algún caso existen ítems cuyo embedding no existe (el W2V no lo aprendió) y por tanto hay que eliminar. Este problemática se ilustra en la Figura 32.

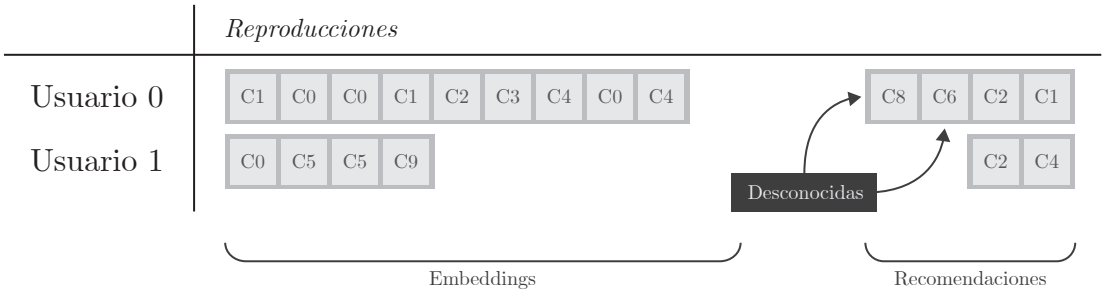

Figura 32 – Canciones desconocidas en subconjunto de embeddings

6.4.1.2. Modelo

Con el fin de poder comparar los resultados de los diferentes test de la tarea 1, se utilizará el mismo modelo (Figura 33) para todos ellos, alterando solamente la capa de entrada.
El modelo utilizado posee los siguientes componentes:

- **Capa de entrada:** En ella se introducen los IDs de usuario y canción.
- **Capa de embeddings:** En esta capa se obtienen los embeddings de usuario y canción de tamaños 128 y 64 respectivamente.
- **Capa de transformación:** En este punto se produce el paso de los embeddings de usuario y canción a la misma dimensión (32) con el fin de permitir el producto escalar.
- **Loss:** Utilizando la salida real (y) y el producto escalar de los vectores transformados, se calcula la pérdida del modelo utilizando la función $SoftPlus$.
- **Probabilidad:** Una vez entrenado el modelo, se utiliza la función *Sigmoid* para conocer la probabilidad de un caso dado.

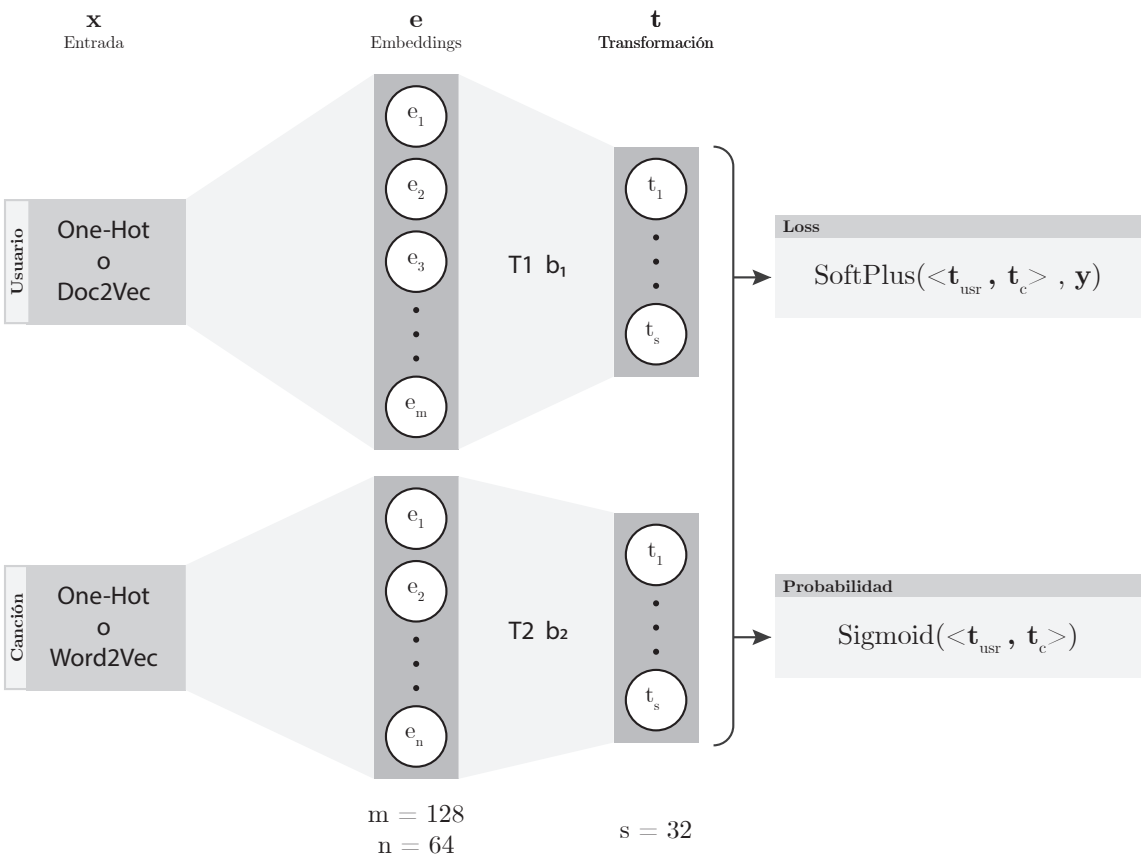

Figura 33 – Arquitectura de la tarea 1.

Para cada uno de los tests de la Tabla 21, los cambios a realizar en la capa de entrada del modelo serán los siguientes:

- **TEST 01:** El usuario se pasará de *One-Hot* a dimensión 128. La canción no se ha de transformar dado que se utiliza el embedding del W2V aprendido previamente (ya en dimiension 64).
- **TEST 02:** No es necesario ninguna transformación previa, el perfil consolidado del usuario D2V ya posee dimensión 128 y las canciones W2V están en espacio 64.
- **TEST 03:** Como en el caso anterior, ningún cambio es necesario.
- **TEST 04:** Este test requiere transformar la concatenación del perfil consolidado y reciente (128+128=256) a tamaño 128.
- **TEST 05:** Se transformará el vector *One-Hot* del usuario a dimensión 128. El vector *One-Hot* de la canción se pasará a dimensión 64, todo ello para cumplir con las dimensiones de la capa de embeddings.
- **TEST 06:** El perfil del usuario (D2V-Consolidado) ya posee la dimensión adecuada. La canción se transformará como en el caso anterior.
- **TEST 07:** Mismo procedimiento que en el test anterior utilizando el perfil reciente.
- **TEST 08:** El test final requiere transformar la concatenación del perfil consolidado y reciente a tamaño 128 y las canciones a tamaño 64.

Siendo $\mathcal{U}$ y $\mathcal{C}$ los conjuntos que contienen los identificadores de los usuarios y canciones codificados en formato One-Hot, se tiene un conjunto de datos formado por tripletas del tipo,

$$\mathcal{D}_1 = \{(\boldsymbol{u}, \boldsymbol{c}, y) : \boldsymbol{u} \in \mathcal{U}, \boldsymbol{c} \in \mathcal{C}, y \in \{+1, -1\}\} \ . \tag{26}$$

Se obtendrán inicialmente los embeddings asociados a cada uno de los vectores $\boldsymbol{u}, \boldsymbol{c}$ utilizando las matrices $\boldsymbol{D}$ y $\boldsymbol{W}$ respectivamente como refleja la Ecuación (27). Estas matrices variarán en función del test que se esté resolviendo.

$$\boldsymbol{e_{usr}} = \boldsymbol{uD} \quad \boldsymbol{e_c} = \boldsymbol{cW} \tag{27}$$

A partir de los embeddings $\boldsymbol{e_{usr}}, \boldsymbol{e_c}$ de tamaños 128 y 64 respectivamente, es necesaria una nueva transformación de ambos a un mismo espacio que posibilite el producto escalar. Esta transformación viene dada por las matrices $\boldsymbol{T_1}$ y $\boldsymbol{T_2}$ y los términos independientes $\boldsymbol{b1}$ y $\boldsymbol{b2}$ que retornan,

$$\boldsymbol{t_{usr}} = \boldsymbol{e_{usr}}\ \boldsymbol{T1} + \boldsymbol{b_1} \quad \boldsymbol{t_c} = \boldsymbol{e_c}\ \boldsymbol{T2} + \boldsymbol{b_2} \tag{28}$$

en un espacio de dimensión 32. Obteniendo el producto escalar

$$dotprod = \langle \boldsymbol{t_{usr}}, \boldsymbol{t_c} \rangle \tag{29}$$

de estos vectores, se puede calcular la pérdida con la función *SoftPlus* de la siguiente forma,

$$loss = -\log\left(1 + e^{-y \cdot dotprod}\right), \tag{30}$$

la cual se minimizará empleando para ello el Descenso del Gradiente.

6.4.1.3. Entrenamiento

Conocidos e implementados los modelos se realizará previamente, para cada uno de ellos, un grid-search entrenando con el conjunto de TRAIN y evaluando con DEV. Se llevarán a cabo 4 pruebas alternando el learning rate con el fin de obtener la mejor combinación de hiperparámetros para el modelo. Algunos hiperparámetros (*batch size*, *número ejemplos negativos*...) no se han variado y se han mantenido fijos dada la elevada explosión combinatoria y el tiempo de ejecución de la tarea.

En cada ejecución, se realizarán un número de epochs determinado por una metodología denominada *early stopping* que detiene la ejecución cuando la pérdida no se reduce significativamente. En nuestro caso, se tomarán los resultados de la loss en DEV en las últimas 100 epochs (por tanto, como mínimo se ha de iterar sobre el conjunto este número de veces) y con ellos se calculará la pendiente de la recta de tendencia que siguen. Mientras esta pendiente posea un valor negativo mayor que $-1e^{-5}$ se continuará la ejecución hasta un máximo de 1000 epochs.

Como resultado de este proceso se han obtenido los siguientes resultados:

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	100	1	78681.091	6448.044
2	100	0.01	0.451	0.377
3	102	0.0001	**0.395**	0.324
4	455	0.000001	0.398	0.334

Tabla 22 – Resultados del *grid-search* para el test número 1

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	100	1	0.960	484.581
2	100	0.01	0.516	0.491
3	199	0.0001	**0.421**	0.378
4	780	0.000001	0.430	0.394

Tabla 23 – Resultados del *grid-search* para el test número 2

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	100	1	0.812	1044.909
2	104	0.01	0.575	0.524
3	202	0.0001	**0.478**	0.438
4	928	0.000001	0.486	0.453

Tabla 24 – Resultados del *grid-search* para el test número 3

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	147	1	4760.145	243297.929
2	100	0.01	0.464	0.396
3	194	0.0001	**0.411**	0.359
4	757	0.000001	0.418	0.375

Tabla 25 – Resultados del *grid-search* para el test número 4

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	100	1	1.330e10	9.439e7
2	100	0.01	666.826	1.219
3	100	0.0001	5.325	1.32e−9
4	410	0.000001	**0.540**	0.381

Tabla 26 – Resultados del *grid-search* para el test número 5

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	100	1	7.720e8	2448158.530
2	100	0.01	977.177	2.283
3	100	0.0001	4.438	1.480e−9
4	464	0.000001	**0.526**	0.262

Tabla 27 – Resultados del *grid-search* para el test número 6

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	100	1	6.330e8	1826939.486
2	100	0.01	599.637	1.776
3	100	0.0001	4.765	1.980e−7
4	401	0.000001	**0.542**	0.318

Tabla 28 – Resultados del *grid-search* para el test número 7

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	100	1	2.800e11	1.580e9
2	100	0.01	3837.685	19.596
3	100	0.0001	4.520	1.840e−5
4	435	0.000001	**0.539**	0.276

Tabla 29 – Resultados del *grid-search* para el test número 8

A la vista de los resultados anteriores, se entrenará cada uno de los modelos con los datos de TRAIN y DEV utilizando los siguientes hiperparámetros:

Test	Usuarios	Canciones	Epochs	Learning rate
01	One-Hot	W2V	102	0,0001
02	D2V Consolidado	W2V	199	0,0001
03	D2V Reciente	W2V	202	0,0001
04	D2V Consolidado + D2V Reciente	W2V	194	0,0001
05	One-Hot	One-Hot	410	0,000001
06	D2V Consolidado	One-Hot	464	0,000001
07	D2V Reciente	One-Hot	401	0,000001
08	D2V Consolidado + D2V Reciente	One-Hot	435	0,000001

Tabla 30 – Hiperparámetros de la tarea 1

6.4.1.4. Evaluación y resultados

Para realizar la fase de evaluación es necesario generar, con los datos previamente separados para ello, casos positivos y negativos como sucedía en la fase de entrenamiento. En un caso como este no suele ser necesario generar casos negativos pero, para poder comparar los modelos se ha de rellenar una matriz de confusión como la siguiente:

	Clase positiva	**Clase negativa**
Predicción positiva	True Positives	False Positives
Predicción negativa	False Negatives	True Negatives

Tabla 31 – Matriz de confusión

Para cada uno de los ejemplos de test, se ha de incrementar el valor adecuado dentro de la tabla. Por ejemplo, en un caso donde un usuario u haya escuchado una canción c (clase positiva), y el modelo retorne 0.75 (predicción positiva) se ha de incrementar el contador de casos *True Positives*. Se ha de tener en cuenta que una predicción positiva es aquella cuya probabilidad supere un valor de 0.5.

A partir de los tabla anterior se calcularán las siguientes medidas:

- **Precision:** Representa, en este caso, la fiabilidad del sistema para predecir canciones escuchadas en el futuro. Una precision alta indica que, si el sistema dijo que una canción se escuchará en el futuro, hay una alta probabilidad de que esto sea cierto. Se calcula mediante la siguiente ecuación:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{31}$$

- **Recall:** Esta medida indica cuántas de las canciones escuchadas en el futuro retorna el sistema. Su valor se obtiene mediante la ecuación:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{32}$$

- **F1:** Combina las medidas anteriores realizando una media armónica con el fin de obtener la exactitud en evaluación. Su ecuación se indica a continuación:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{33}$$

Con el objetivo de aclarar el significado de estas medidas, en la Figura 34 se puede ver de forma gráfica cómo se comportan y qué tratan de representar.

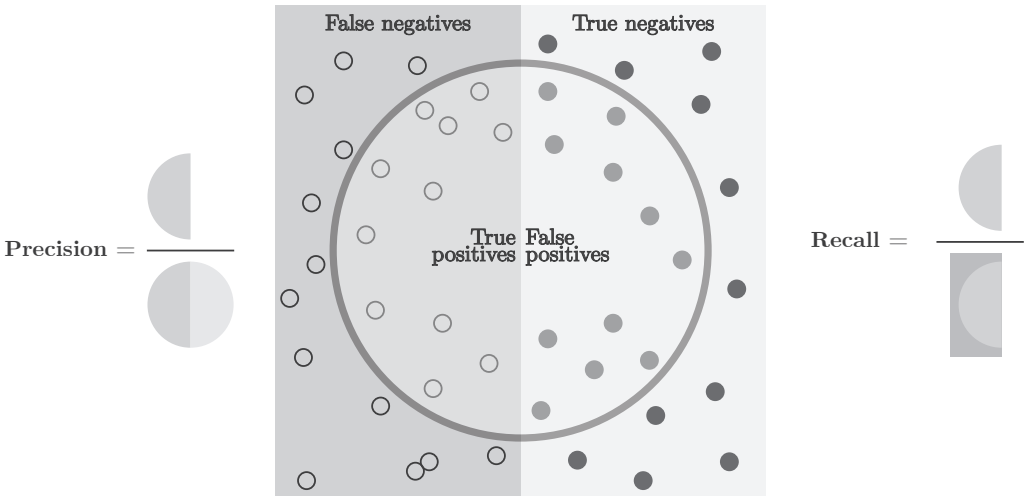

Figura 34 – Representación gráfica de las medidas *Precision* y *Recall*.

Tras realizar cada una de las pruebas definidas obteniendo las métricas correspondientes, se han obtenido los siguientes resultados:

Test	Usuarios	Canciones	Loss	Precision	Recall	F1
01	One-Hot	W2V	0.410	83.053 %	80.170 %	**81.586** %
02	D2V Consolidado	W2V	0.435	80.476 %	79.133 %	79.799 %
03	D2V Reciente	W2V	0.490	77.536 %	74.463 %	75.969 %
04	D2V Cons. + D2V Rec.	W2V	0.426	81.690 %	79.128 %	80.389 %
05	One-Hot	One-Hot	0.533	77.537 %	73.801 %	75.623 %
06	D2V Consolidado	One-Hot	0.515	81.479 %	76.138 %	78.718 %
07	D2V Reciente	One-Hot	0.531	79.424 %	73.329 %	76.255 %
08	D2V Cons. + D2V Rec.	One-Hot	0.525	80.927 %	75.584 %	78.164 %

Tabla 32 – Resultados de la tarea 1 para cada una de las pruebas realizadas.

Se puede ver a simple vista que, el test con los mejores resultados de Precision, Recall y por tanto F1, es el número 01. Este test utiliza una representación de usuarios en formato *One-Hot* y una codificación de canciones mediante W2V, lo que hace pensar que, para esta tarea, no es muy relevante la codificación del usuario.

Utilizando solamente las versiones *One-Hot* de usuarios y canciones (test 05), se observa que el resultado en F1 es el peor del conjunto de tests para esta tarea. Este comportamiento parece reflejar que, si bien no es importante obtener un embedding previo de los usuarios, si es relevante obtener un embedding de las canciones para esta tarea de recomendación concreta.

En cuanto a los tests que hacen uso del D2V para representar a los usuarios, los peores resultados los reporta el número 03, reafirmando que utilizar el perfil reciente no es lo ideal en esta tarea.

6.4.2. Tarea 2

En este caso, el tipo de tarea de recomendación a resolver trata de predecir, para un usuario y una canción dada, la siguiente canción a escuchar. Al igual que en la primera tarea, se variarán las representaciones de los usuarios y las canciones del mismo modo, resultando 8 pruebas diferentes (Tabla 33).

Test	Representación usuarios	Representación canciones
09	One-Hot	W2V
10	D2V Consolidado	W2V
11	D2V Reciente	W2V
12	D2V Consolidado + D2V Reciente	W2V
13	One-Hot	One-Hot
14	D2V Consolidado	One-Hot
15	D2V Reciente	One-Hot
16	D2V Consolidado + D2V Reciente	One-Hot

Tabla 33 – Representaciones de usuarios y canciones a utilizar en cada uno de los test de la tarea 2.

6.4.2.1. Creación de ejemplos

Al igual que en la tarea anterior, para generar los ejemplos de entrenamiento es necesario eliminar previamente todas aquellas canciones que no poseen embedding (Figura 32). Una vez eliminadas, simplemente se ha de generar para cada usuario ejemplos del tipo:

$$(usuario,\ canción\ escuchada) \Rightarrow\ siguiente\ canción\ escuchada$$

A modo aclaratorio, en la Figura 35 se detalla el procedimiento de creación de los ejemplos.

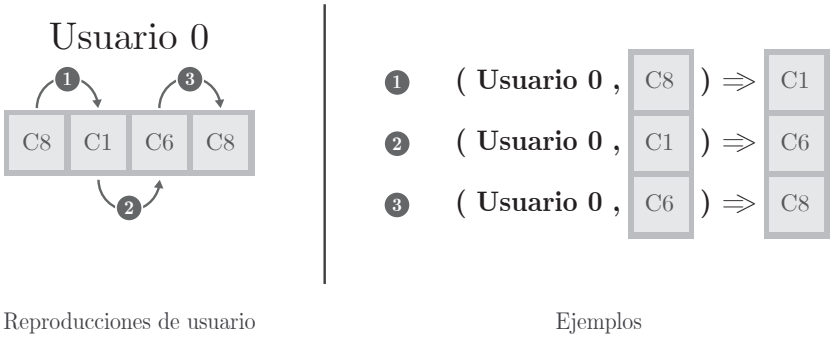

Figura 35 – Creación de ejemplos en la tarea 2.

6.4.2.2. Modelo

En este caso, se seguirá el mismo principio que en la tarea 1, es decir, utilizar el mismo modelo para cada uno de los test alterando unicamente la capa de entrada. El modelo indicado en la Figura 36 posee los siguientes elementos:

- **Capa de entrada:** En ella se introducen los IDs de usuario y canción.
- **Capa de embeddings y concatenación:** En esta capa se obtienen los embeddings de usuario y canción de tamaños 128 y 64 respectivamente y se concatenan creando un vector de dimensión 192.
- **Capa de salida:** La capa final contiene un vector del tamaño del número de canciones existentes. En cada ítem del vector se encuentra una valoración de que la canción que ocupa dicha posición sea la siguiente a escuchar.
- **Loss:** Utilizando los logits ($\hat{y}$) junto con la salida real (y), se calcula la pérdida del modelo utilizando la función $NCE\ Loss$.
- **Probabilidad:** Una vez entrenado el modelo, se utiliza la función $SoftMax$ para transformar en probabilidades el vector de la capa de salida.

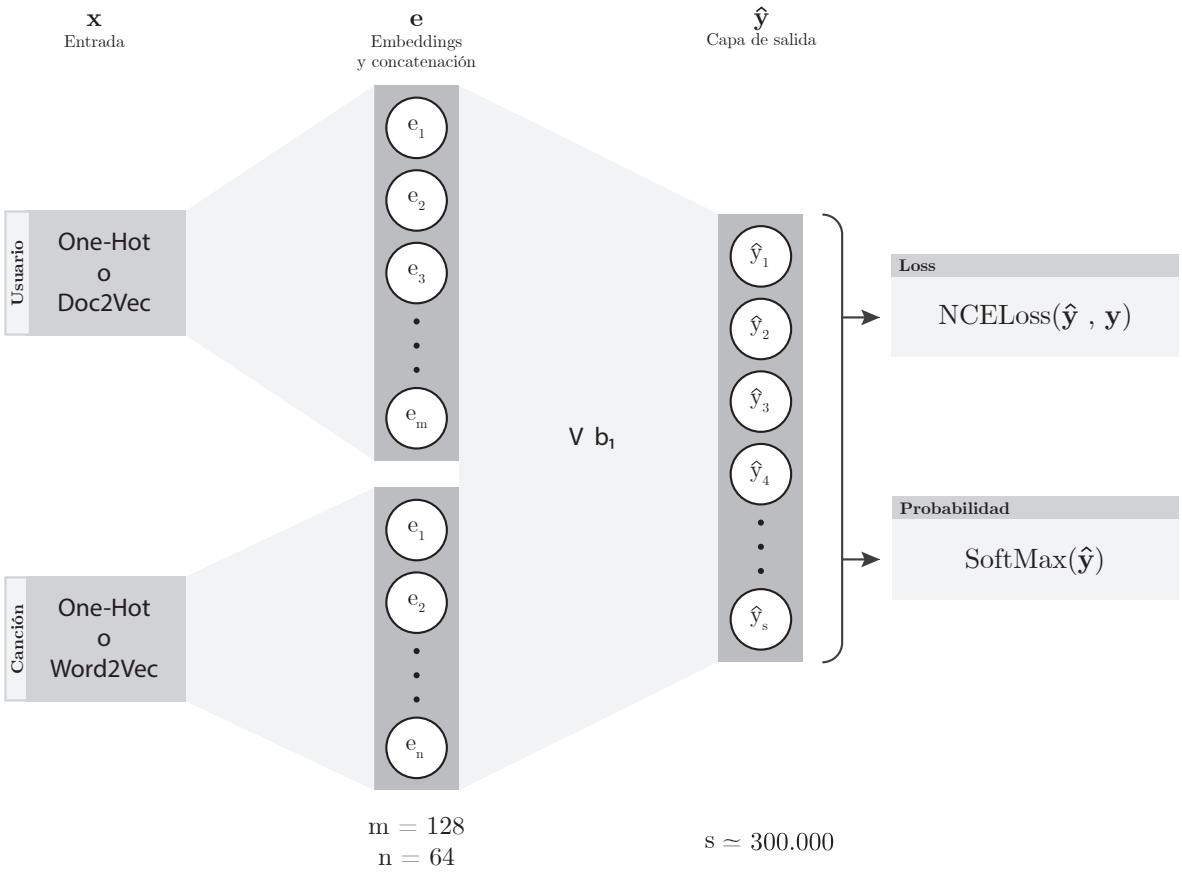

Figura 36 – Arquitectura de la tarea 2.

Siendo $\mathcal{U}$ y $\mathcal{C}$ los conjuntos que contienen los identificadores de los usuarios y canciones codificados en formato One-Hot, se tiene un conjunto de datos formado por tripletas del tipo,

$$\mathcal{D}_2 = \{(\boldsymbol{u}, \boldsymbol{c_i}, \boldsymbol{y}) : \boldsymbol{u} \in \mathcal{U},\ \boldsymbol{c_i}, \boldsymbol{y} \in \mathcal{C}\}\ . \tag{34}$$

Se obtendrán inicialmente los embeddings asociados a cada uno de los vectores $\boldsymbol{u}, \boldsymbol{c}$ utilizando las matrices $\boldsymbol{D}$ y $\boldsymbol{W}$ respectivamente como indica la Ecuación (35). Estas matrices variarán en función del test que se esté resolviendo.

$$\boldsymbol{e_{usr}} = \boldsymbol{uD} \quad \boldsymbol{e_c} = \boldsymbol{c_i W} \tag{35}$$

Los embeddings $\boldsymbol{e_u}, \boldsymbol{e_c}$ de tamaños 128 y 64 respectivamente se concatenarán obteniendo un vector $\boldsymbol{e_{conc}}$ de tamaño 192. Esta concatenación se transformará finalmente a tamaño s mediante la matriz $\boldsymbol{V}$ y el término independiente $\boldsymbol{b1}$ retornando

$$\hat{\boldsymbol{y}} = \boldsymbol{e_{conc}V} + \boldsymbol{b1} \ . \tag{36}$$

En este caso, dado el elevado número de clases diferentes en la capa de salida, se ha optado por usar nuevamente el método NCE explicado en la Sección 6.2.3.2 como función de perdida,

$$loss = NCELoss(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{37}$$

la cual se minimizará empleando para ello el Descenso del Gradiente.

En cuanto a los cambios a realizar en la capa de entrada del modelo para cada uno de los test, se aplicarán las mismas modificaciones que en la tarea de recomendación número 1 (Apartado 6.4.1.2).

6.4.2.3. Entrenamiento

El procedimiento en este caso será el mismo que en la tarea de recomendación anterior, es decir, se realizará un *grid-search* variando el learning rate y se detendrá la ejecución según lo estimado por el *early-stopping*. Como resultado de este procedimiento se obtiene:

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	172	1	**2.543**	0.490
2	1000	0.01	2.983	2.425
3	1000	0.0001	4.044	3.773
4	1000	0.000001	310.949	311.005

Tabla 34 – Resultados del *grid-search* para el test número 9

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	458	1	31.818	0.623
2	1000	0.01	**2.428**	0.886
3	1000	0.0001	72.714	74.324
4	1000	0.000001	246.963	248.682

Tabla 35 – Resultados del *grid-search* para el test número 10

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	746	1	22.523	0.599
2	1000	0.01	**2.635**	1.271
3	1000	0.0001	73.58	75.215
4	1000	0.000001	247.257	249.164

Tabla 36 – Resultados del *grid-search* para el test número 11

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	1000	1	∞	∞
2	164	0.01	**3.160**	0.825
3	1000	0.0001	3.196	2.356
4	1000	0.000001	31.598	31.684

Tabla 37 – Resultados del *grid-search* para el test número 12

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	1000	1	∞	∞
2	1000	0.01	**3.741**	3.256
3	1000	0.0001	4.091	3.787
4	551	0.000001	342.248	342.545

Tabla 38 – Resultados del *grid-search* para el test número 13

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	379	1	29.603	0.568
2	624	0.01	**2.634**	1.158
3	1000	0.0001	4.532	4.205
4	1000	0.000001	250.945	251.250

Tabla 39 – Resultados del *grid-search* para el test número 14

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	411	1	21.314	0.511
2	1000	0.01	**2.827**	1.472
3	1000	0.0001	4.616	4.342
4	1000	0.000001	251.462	251.771

Tabla 40 – Resultados del *grid-search* para el test número 15

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	1000	1	∞	∞
2	164	0.01	3.299	0.906
3	1000	0.0001	**3.192**	2.370
4	1000	0.000001	32.611	31.287

Tabla 41 – Resultados del *grid-search* para el test número 16

A la vista de los resultados anteriores, se entrenará cada modelo con los datos de TRAIN y DEV utilizando los siguientes hiperparámetros:

Test	Usuarios	Canciones	Epochs	Learning rate
09	One-Hot	W2V	172	1
10	D2V Consolidado	W2V	1000	0.01
11	D2V Reciente	W2V	1000	0.01
12	D2V Consolidado + Reciente	W2V	164	0.01
13	One-Hot	One-Hot	1000	0.01
14	D2V Consolidado	One-Hot	624	0.01
15	D2V Reciente	One-Hot	1000	0.01
16	D2V Consolidado + Reciente	One-Hot	1000	0.0001

Tabla 42 – Hiperparámetros de la tarea 2

6.4.2.4. Evaluación y resultados

Analizando la tarea de recomendación que se pretende resolver, se puede ver que, para cada par usuario canción que se introduce en la entrada, pueden existir diferentes salidas. Todas estas salidas son válidas en algún momento, pero solo una nos interesa para el usuario y canción que acabamos de introducir.

Un usuario puede haber escuchado una canción $c1$ y pasar a escuchar $c2$, pero es muy probable que en el futuro el usuario vuelva a escuchar $c1$ pasando a continuación a $c9$. Estos casos provocan las salidas múltiples del modelo, siendo todas ellas válidas en algún momento del tiempo.

Dada la complejidad de la tarea de recomendación, se ordenará la salida del modelo de mayor a menor probabilidad para obtener una lista de canciones que pueden preceder a la canción de la entrada y se buscará si la canción deseada se encuentra en un top x de canciones, siendo $x = \{5, 50, 100, 1000\}$. Esta métrica también se conoce como *precision at*.

Con la métrica anterior conoceremos si la siguiente canción a escuchar se encuentra en el top 5,50,100 y 1000 del vector ordenado de salida, pero con el fin de complementar esta información, se calculará también la mediana de la posición real de la canción en la recomendación. Los resultados de estas métricas se encuentran en la Tabla 43.

Test	**Usuarios**	**Canciones**	**Top 5**	**Top 50**	**Top 100**	**Top 1000**	**Mediana**
09	One-Hot	W2V	7.750 %	**25.161** %	**31.807** %	**58.575** %	511
10	D2V Cons.	W2V	7.310 %	18.158 %	23.766 %	52.122 %	857
11	D2V Rec.	W2V	**8.333** %	19.419 %	24.709 %	50.168 %	986
12	Cons. + Rec.	W2V	1.711 %	8.952 %	14.093 %	45.710 %	1299
13	One-Hot	One-Hot	1.372 %	4.142 %	6.160 %	18.304 %	17476
14	D2V Cons.	One-Hot	1.523 %	7.741 %	12.299 %	43.114 %	1550
15	D2V Rec.	One-Hot	1.612 %	7.703 %	12.022 %	39.262 %	2202
16	Cons. + Rec.	One-Hot	1.216 %	6.151 %	9.621 %	31.071 %	4527

Tabla 43 – Resultados de la tarea 2

Inicialmente se puede ver que los test donde se usa la representación W2V para las canciones poseen un mejor rendimiento que los casos que hacen uso del One-Hot.

El rendimiento de los 3 primeros perfiles que utilizan el W2V (tests 09, 10 y 11) es muy superior al resto de pruebas, reportando los mejores resultados de la tarea. Destaca en el top 5 el uso del perfil reciente junto con el W2V, lo que implica que, el 8.333 % de las veces, la siguiente canción a escuchar se encuentra en el top 5 de las predichas.

En el resto de métricas (Top 50, 100 y 1000), el uso del One-Hot como perfil de usuario junto con el W2V supera notablemente a sus competidores, lo cual también se ve reflejado en la mediana de la posición real. Destacar que, la concatenación de perfiles D2V, posee unos resultados muy bajos en comparación con sus versiones individuales.

Para finalizar, el uso de la representación One-Hot en las canciones (tests 13, 14, 15 y 16) posee unos resultados muy pobres en comparación con los analizados previamente. Se puede ver una ligera mejora en los test 14 y 15 donde se utilizan los perfiles D2V de manera individual, pero muy baja en el caso del top 5.

6.4.3. Tarea 3

La última tarea de recomendación que se abordará tiene como fin ampliar la tarea 2. En el caso previo se pretendía que, dada una canción, se recomendara la siguiente a escuchar; en este caso se intentará realizar la misma labor utilizando las dos canciones anteriores. El objetivo de añadir este extra de información es conocer si los resultados de las predicciones varían en gran medida además de ver si existe algún método que se sobresalga en dicho ámbito.

Las combinaciones y pruebas a realizar en este caso, no difieren de las tareas anteriores:

Test	Representación usuarios	Representación canciones
17	One-Hot	W2V
18	D2V Consolidado	W2V
19	D2V Reciente	W2V
20	D2V Consolidado + D2V Reciente	W2V
21	One-Hot	One-Hot
22	D2V Consolidado	One-Hot
23	D2V Reciente	One-Hot
24	D2V Consolidado + D2V Reciente	One-Hot

Tabla 44 – Representaciones de usuarios y canciones a utilizar en cada uno de los test de la tarea 3.

6.4.3.1. Creación de ejemplos

La creación de ejemplos en este caso requiere, al igual que en las tareas previas, de eliminar previamente todas aquellas canciones que no poseen embedding. Eliminadas estas canciones se puede proceder a la generación de ejemplos con la siguiente estructura:

$$(usuario,\ canción\ escuchada\ 1, canción\ escuchada\ 2) \Rightarrow\ siguiente\ canción\ escuchada$$

Este proceso se puede ver representado gráficamente en la Figura 37 donde se detalla el procedimiento de creación de los ejemplos.

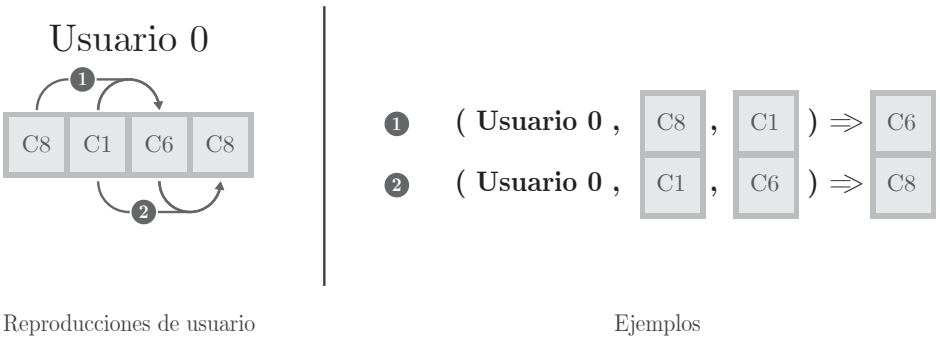

Figura 37 – Creación de ejemplos en la tarea 3.

6.4.3.2. Modelo

En esta tarea, el modelo será muy similar al utilizado en la tarea 2, variando simplemente la entrada y la concatenación de embeddings. Para cada uno de los tests a realizar se utilizará el mismo modelo alterando unicamente la capa de entrada. El modelo indicado en la Figura 38 posee los siguientes elementos:

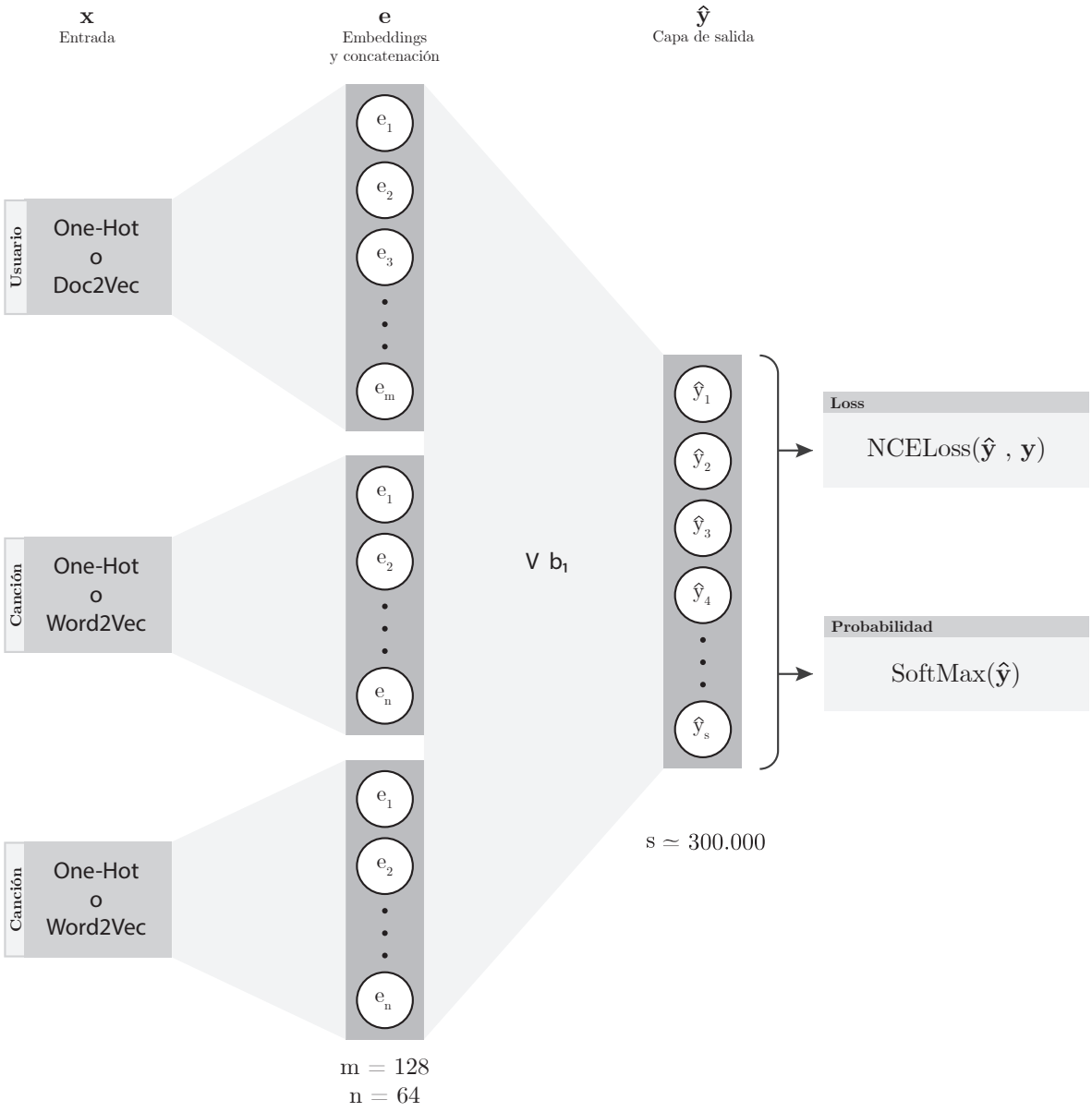

Figura 38 – Arquitectura de la tarea 3.

- **Capa de entrada:** En ella se introducen el ID de usuario y los de las canciones.
- **Capa de embeddings y concatenación:** En esta capa se obtiene el embedding del usuario, la canción 1 y la 2 de tamaños 128, 64 y 64 respectivamente para concatenarlos finalmente creando un vector de dimensión 256.
- **Capa de salida:** La capa final contiene un vector del tamaño del número de canciones existentes. En cada ítem del vector se encuentra una valoración de que la canción que ocupa dicha posición sea la siguiente a escuchar.
- **Loss:** Utilizando los logits ($\hat{y}$) junto con la salida real (y), se calcula la pérdida del modelo utilizando la función *NCE Loss*.

- **Probabilidad:** Una vez entrenado el modelo, se utiliza la función $SoftMax$ para transformar en probabilidades el vector de la capa de salida.

Partiendo de $\mathcal{U}$ y $\mathcal{C}$, conjuntos que contienen los identificadores de usuarios y canciones codificados en formato One-Hot, se tiene un conjunto de datos formado por cuádruplas del tipo,

$$\mathcal{D}_3 = \{(\boldsymbol{u}, \boldsymbol{c_i}, \boldsymbol{c_j}, \boldsymbol{y}) : \boldsymbol{u} \in \mathcal{U},\ \boldsymbol{c_i}, \boldsymbol{c_j}, \boldsymbol{y} \in \mathcal{C}\}\ . \tag{38}$$

Se obtendrán inicialmente los embeddings asociados a cada uno de los vectores $\boldsymbol{u}, \boldsymbol{c}$ utilizando las matrices $\boldsymbol{D}$ y $\boldsymbol{W}$ respectivamente como refleja la Ecuación (39). Estas matrices variarán en función del test que se esté resolviendo.

$$\boldsymbol{e_{usr}} = \boldsymbol{uD} \quad \boldsymbol{e_{ci}} = \boldsymbol{c_iW} \quad \boldsymbol{e_{cj}} = \boldsymbol{c_jW} \tag{39}$$

Los embeddings $\boldsymbol{e_u}, \boldsymbol{e_{ci}}, \boldsymbol{e_{cj}}$ de tamaños 128, 64 y 64 respectivamente se concatenarán obteniendo un vector $\boldsymbol{e_{conc}}$ de tamaño 256. Esta concatenación se transformará finalmente a tamaño s mediante la matriz $\boldsymbol{V}$ y el término independiente $\boldsymbol{b1}$ retornando:

$$\hat{\boldsymbol{y}} = \boldsymbol{e_{conc}V} + \boldsymbol{b1}\ . \tag{40}$$

En este caso, dado el elevado número de clases diferentes en la capa de salida, se ha optado por usar nuevamente el método NCE explicado en la Sección 6.2.3.2 como función de perdida,

$$loss = NCELoss(\boldsymbol{y}, \hat{\boldsymbol{y}}) \tag{41}$$

la cual se minimizará empleando para ello el Descenso del Gradiente.

Destacar que en cada una de las canciones de la capa de entrada se utilizará siempre la misma representación, es decir, no se codificará una canción con el *One-Hot* y la siguiente con el *Word2Vec* dado que este comportamiento carece de sentido. En los test donde las canciones se representen vía *One-Hot*, se utilizará la misma matriz para ambas canciones de entrada, es decir, no se aprenderán dos matrices diferentes.

En cuanto a los cambios a realizar en la capa de entrada del modelo para cada uno de los test, se efectuarán las mismas modificaciones ya aplicadas en las tareas de recomendación 1 y 2 (Apartado 6.4.1.2).

6.4.3.3. Entrenamiento

Al igual que en las tareas anteriores se realizará un *grid-search* variando el learning rate y se detendrá la ejecución según lo estimado por el *early-stopping*. Como resultado de este procedimiento se obtiene:

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	188	1	3.903	0.320
2	1000	0.01	**2.784**	2.076
3	1000	0.0001	4.028	3.754
4	1000	0.000001	299.705	298.645

Tabla 45 – Resultados del *grid-search* para el test número 17

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	443	1	33.238	0.431
2	1000	0.01	**2.306**	0.797
3	1000	0.0001	71.872	71.477
4	1000	0.000001	247.804	247.396

Tabla 46 – Resultados del *grid-search* para el test número 18

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	699	1	24.094	0.417
2	1000	0.01	**2.468**	1.120
3	1000	0.0001	72.685	72.313
4	1000	0.000001	248.257	247.836

Tabla 47 – Resultados del *grid-search* para el test número 19

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	1000	1	∞	∞
2	167	0.01	**3.135**	0.782
3	1000	0.0001	3.160	2.342
4	1000	0.000001	31.361	31.423

Tabla 48 – Resultados del *grid-search* para el test número 20

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	1000	1	∞	∞
2	1000	0.01	**3.677**	3.246
3	1000	0.0001	4.019	3.740
4	465	0.000001	344.371	343.545

Tabla 49 – Resultados del *grid-search* para el test número 21

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	343	1	30.699	0.415
2	597	0.01	**2.610**	1.179
3	1000	0.0001	3.819	3.432
4	1000	0.000001	248.336	250.100

Tabla 50 – Resultados del *grid-search* para el test número 22

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	1000	1	∞	∞
2	1000	0.01	**2.813**	1.454
3	1000	0.0001	3.982	3.659
4	1000	0.000001	248.860	250.607

Tabla 51 – Resultados del *grid-search* para el test número 23

Ejecución	Epochs	Learning rate	DEV loss	TRAIN loss
1	1000	1	∞	∞
2	165	0.01	3.322	0.907
3	1000	0.0001	**3.199**	2.365
4	1000	0.000001	31.521	30.990

Tabla 52 – Resultados del *grid-search* para el test número 24

A la vista de los resultados anteriores, se entrenará cada modelo con los datos de TRAIN y DEV utilizando los siguientes hiperparámetros:

Test	Usuarios	Canciones	Epochs	Learning rate
17	One-Hot	W2V	1000	0,01
18	D2V Consolidado	W2V	1000	0,01
19	D2V Reciente	W2V	1000	0,01
20	D2V Consolidado + Reciente	W2V	167	0,01
21	One-Hot	One-Hot	1000	0,01
22	D2V Consolidado	One-Hot	597	0,01
23	D2V Reciente	One-Hot	1000	0,01
24	D2V Consolidado + Reciente	One-Hot	1000	0,0001

Tabla 53 – Hiperparámetros de la tarea 3

6.4.3.4. Evaluación y resultados

El método y métricas de evaluación utilizados en este caso son exactamente los mismos empleados en la tarea 2. En la Tabla 54 representada a continuación se pueden ver los resultados.

Test	Usuarios	Canciones	Top 5	Top 50	Top 100	Top 1000	Mediana
17	One-Hot	W2V	**11.316 %**	22.414 %	26.033 %	41.169 %	2912
18	D2V Cons.	W2V	9.162 %	21.302 %	27.031 %	**54.496 %**	712
19	D2V Rec.	W2V	10.500 %	**22.863 %**	**28.398 %**	53.395 %	752
20	Cons. + Rec.	W2V	1.366 %	9.786 %	15.172 %	46.712 %	1226
21	One-Hot	One-Hot	1.844 %	4.980 %	6.904 %	18.989 %	17252
22	D2V Cons.	One-Hot	1.582 %	7.808 %	12.549 %	43.091 %	1556
23	D2V Rec.	One-Hot	1.858 %	8.031 %	12.458 %	39.566 %	2157
24	Cons. + Rec.	One-Hot	1.245 %	6.110 %	9.607 %	31.021 %	4560

Tabla 54 – Resultados de la tarea 3

Representando cada una de las canciones mediante su embedding W2V (4 primeros tests) se puede ver un comportamiento similar al visto en la tarea anterior. En este caso, el mejor resultado en el top 5 lo posee el perfil de usuario One-Hot junto con el W2V, seguido muy de cerca por el perfil D2V reciente + W2V.

El resto de métricas (top 50, 100 y 1000) poseen sus mejores resultados en los test 19 y 18 respectivamente, siendo muy parejos sus valores. Como ha sucedido en la tarea anterior, la concatenación de perfiles no mejora en absoluto los resultados de los perfiles individuales, siendo en esta tarea los que peores resultados poseen.

Los tests que hacen uso de la codificación One-Hot para las canciones poseen, una vez más, los peores resultados en todas las métricas disponibles. Cabe destacar, finalmente, la reducida diferencia entre los resultados en top 5 de esta tarea y la anterior a pesar de ser esta última más "fácil" de aprender.

7. Material utilizado

7.1. Hardware

Para la programación, pruebas, limpieza de datos y escritura de la memoria se ha utilizado un PC con la siguiente configuración:

- **CPU:** Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz (4 Cores / 8 Threads)
- **GPU:** Nvidia GeForce GTX 1050 Ti (4 GB)
- **RAM:** 16 GB
- **Sistema Operativo:** Windows 10 x64

La ejecución de experimentos (aprendizaje de embeddings, tareas de recomendación...) se ha realizado en un ordenador con las siguientes características:

- **CPU:** Intel(R) Xeon(TM) i9-7900X CPU @ 3.30GHz (10 Cores / 20 Threads)
- **GPU:** 2x Nvidia TITAN Xp (12 GB) SLI
- **RAM:** 64 GB
- **Sistema Operativo:** Ubuntu 16.04.4 LTS x64

Con el fin de paralelizar experimentos, se ha utilizado además un servicio de supercomputación online que consta de 4 nodos donde cada uno posee:

- **CPU:** 2x Intel(R) Xeon(TM) Processor E5-2680 v3 @ 2.50GHz (12 Cores / 24 Threads)
- **GPU:** 2x Nvidia Tesla K80 (24 GB)
- **RAM:** 128 GB
- **Sistema Operativo:** Red Hat Enterprise Linux Server 6.7 x64

7.2. Software

Todo el código necesario se ha realizado utilizando el lenguaje interpretado *Python* dado el alto número de librarías existentes para este tipo de problemas. Dentro de las librerías utilizadas destacan:

- **TensorFlow:** Libería open-source proporcionada por Google que permite la resolución de numerosas tareas de machine-learning utilizando paralelización de GPU y CPU.
- **Scikit-learn:** Libería open-source con herramientas para la resolución de problemas de clustering, machine-learning, análisis de datos...
- **NumPy:** Libería científica (open-source) a de python que permite realizar numerosas operaciones matemáticas de forma sencilla y rápida gracias a su implementación en C y C++.
- **Pandas:** Libería open-source de análisis de datos que permite realizar operaciones de manera sencilla en estructuras con datos de diversa naturaleza.

7.3. Otros

Las figuras que aparecen a lo largo de la memoria han sido generadas utilizando el software *Adobe Ilustrator* y todo el documento ha sido escrito en código LaTeX utilizando el editor *TeXstudio*.

8. Conclusiones

Tras analizar todas las pruebas y valores obtenidos en cada una de las 3 tareas de recomendación, se pueden extraer varias conclusiones.

Comenzando por la parte de los embeddings para las canciones, se puede ver de forma clara que, la diferencia entre un caso donde se utiliza el One-Hot (tests 05-08, 13-16 y 21-24) frente a otro igual donde se hace uso del W2V (tests 01-04, 09-12 y 17-20) es muy notable, siendo siempre mucho mejor este último.

Este comportamiento refleja de forma clara la utilidad de una representación mediante embeddings W2V en tareas de recomendación en el lado de las canciones, superando de forma concisa la tradicional representación One-Hot.

En cuanto a los perfiles de usuario y, respondiendo a uno de los objetivos principales de este trabajo, se puede observar que el uso de la representación One-Hot de los mismos supera a los vectores aprendidos mediante una tarea previa (D2V) en la mayoría de los casos. Los test que utilizan el One-Hot como representación de los usuarios, aprenden un perfil específico para la tarea de recomendación concreta, lo que parece muchos más efectivo que aprender previamente un perfil genérico para cada uno de los usuarios y tareas.

Otro de los objetivos principales de la investigación era conocer si es útil mantener, para cada uno de los usuarios, un perfil reciente y otro consolidado. A la vista de los resultados, parece que esta separación sólo es efectiva en la segunda tarea (test 11) donde el top 5 del perfil consolidado supera levemente los resultados del One-Hot. Sin embargo, estos resultados no parecen suficientes para asegurar que el uso de este perfil sea mejor que una versión One-Hot del mismo.

Cabe destacar que el uso del perfil consolidado posee mejores resultados en la tarea de recomendación 1 que el uso del perfil reciente. Este comportamiento se invierte en las tareas número 2 y 3 donde parece más efectivo hacer uso de un perfil reciente.

Otro aspecto a tener en cuenta es el mal funcionamiento de la concatenación de perfiles consolidado y reciente a lo largo de todas las tareas, siendo prácticamente siempre el que peor resultados reporta.

Por tanto, a la vista de los resultados y análisis previos, se puede concluir que la mejor opción para este conjunto de datos y tareas de recomendación es la utilización de un perfil One-Hot para los usuarios combinada con la representación W2V de las canciones.

9. Mejoras y trabajo futuro

A continuación se indican una serie de posibles mejoras aplicables a algunas secciones de la experimentación con el fin de evaluar nuevos métodos u obtener resultados más fiables.

9.1. Creación de ejemplos

En la Sección 6.2.2 se detalla el funcionamiento de la codificación de ejemplos para el método Word2Vec y se indica que es realizado a partir de la lista de reproducciones de todos los usuarios concatenadas.

Realizar esta concatenación implica que se generen ejemplos "falsos" en las uniones entre varios usuarios como refleja la Figura 39.

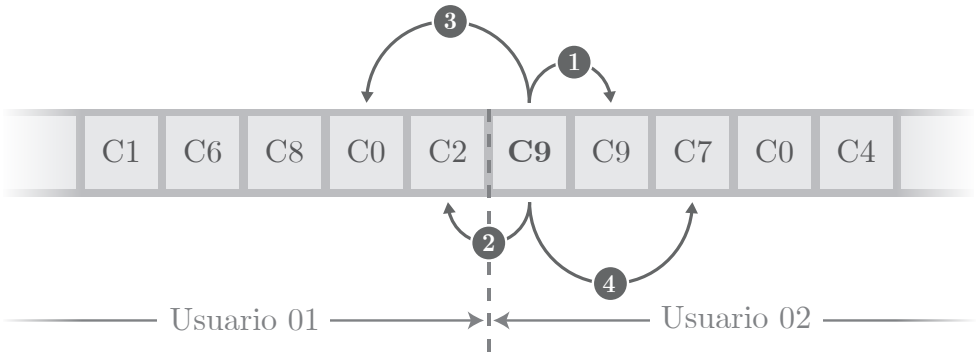

Figura 39 – Ejemplos "falsos" del método Word2Vec.

En ese caso concreto tanto el ejemplo número 2 como el número 3 serían ejemplos "falsos" dado que se encuentran justamente en la unión entre ambos usuarios. Esta casuística no se produce de forma habitual, por esto, se ha decidido asumir el error durante la experimentación.

9.2. Embeddings de usuarios

Además del método Doc2Vec empleado a lo largo del documento, existen otros métodos que podrían ser empleados para la codificación de usuarios pero, por falta de tiempo, no se han podido abordar. Estos métodos son, principalmente, los denominados LSTM [23] (Long Short-Term Memory) y GRU [24] (Gated Recurrent Units) basados en redes neuronales recurrentes (RNN).

Tanto el método LSTM como el GRU (versión simplificada del anterior) son muy utilizados para resolver problemas en los que la secuencia y el orden poseen una alta relevancia. Algunos ejemplos serían el lenguaje hablado, la evolución del mercado financiero, datos de sensores...

Estas propiedades hacen factible su uso en los problemas que se plantean en este documento y por ello se proponen en forma de trabajo futuro.

9.3. Early-stopping en tareas de recomendación

A lo largo de la experimentación se han realizado numerosos *grid-search* en los cuales se ha empleado el método *early-stopping* con el fin de obtener el número de epochs adecuado para cada uno de los 24 test realizados.

Durante este proceso, se ha fijado en 1000 el número máximo de epochs, lo que ha provocado en numerosas ocasiones la parada prematura de los test por este motivo y no por la ausencia de mejora en la función de pérdida. Aumentando este número, se podría mejorar la precisión de cada uno de los métodos reportando mejores resultados en algunos casos.

Aún conociendo esto, se ha fijado en 1000 este valor dado el gran coste temporal que implica realizar cada uno de los 24 *grid-seach*.

10. Agradecimientos

Finalmente agradecer a “NVIDIA Corporation” por la donación de dos tarjetas *Titan Xp* usadas a lo largo de la investigación sin las cuales no se podría haber realizado todo el conjunto de experimentos en un periodo de tiempo tan reducido.

Referencias

[1] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[2] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer, 2011.

[3] Amazon, `https://www.amazon.com`.

[4] Netflix, `https://www.netflix.com`.

[5] Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender systems handbook*, pages 77–118. Springer, 2015.

[6] Spotify, `https://www.spotify.com`.

[7] Google, `https://www.google.com`.

[8] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[9] Jason Weston, Samy Bengio, and Philippe Hamel. Multi-tasking with joint semantic spaces for large-scale music annotation and retrieval. *Journal of New Music Research*, 40(4):337–348, 2011.

[10] Shuo Chen, Josh L Moore, Douglas Turnbull, and Thorsten Joachims. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 714–722. ACM, 2012.

[11] Joshua L Moore, Shuo Chen, Thorsten Joachims, and Douglas Turnbull. Learning to embed songs and tags for playlist prediction. In *ISMIR*, volume 12, pages 349–354, 2012.

[12] Oscar Luaces, Jorge Díez, Thorsten Joachims, and Antonio Bahamonde. Mapping preferences into euclidean space. *Expert Systems with Applications*, 42(22):8588–8596, 2015.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[14] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[15] Geoffrey E Hinton, James L Mcclelland, and David E Rumelhart. Distributed representations, parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations, 1986.

[16] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[17] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[18] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[19] Last.fm, `https://www.last.fm`.

[20] Sequencematcher, difflib, `https://docs.python.org/2/library/difflib.html`.

[21] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[22] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.