Práctica de laboratorio 7: Embeddings.

Castro Elvira D. [2022710039], Pineda Rugerio N. [20222710240], Castañón Hernández V. J. [2022710020], Galicia Cocoletzi N. [2022710234], Sánchez Zanjuampa M. A. [2022710029], y Nava Mendez E. U. [2021710144].

Profesor: Lauro Reyes Cocoletzi

***Resumen*—Para el entrenamiento de un modelo Word2vec, se trabajó con un dataset extraído de documentos de Wikipedia y se realizó un procesamiento de subsampling para reducir la cantidad de palabras repetidas. Luego, se calcularon los embeddings de las palabras y se entrenó un modelo utilizando una capa de embedding con 20 épocas y un learning rate de 0.02. El objetivo era utilizar este modelo para calcular la similitud entre palabras mediante la distancia coseno.**

Sin embargo, los resultados obtenidos fueron pobres debido a que el dataset era demasiado pequeño y las relaciones entre las palabras no tenían sentido. Esto puede deberse a la falta de diversidad temática en el dataset, lo que limita la capacidad del modelo para capturar patrones semánticos significativos.

***Palabras clave*—Embeddigns, Word2vec, Subsampling, Muestreo negativo, Distancia coseno**

I. Introducción

Los embeddings son representaciones vectoriales que facilitan el análisis de las relaciones semánticas y sintácticas entre palabras en un corpus dado. Son herramientas fundamentales en el procesamiento del lenguaje natural, permitiendo mejorar la comprensión, la generación de contenido y la extracción de información relevante.

Esta práctica tiene como objetivo analizar el proceso necesario para implementar adecuadamente los embeddings en un contexto específico, como el análisis de un corpus sobre un tema determinado. Esto implica enfrentar desafíos particulares según la naturaleza de los datos proporcionados y el enfoque de análisis requerido.

I-A. Marco teórico

I-A1. Embeddings.:

Los embeddings son representaciones numéricas de objetos del mundo real que permiten a los modelos de aprendizaje automático comprender relaciones y similitudes complejas entre objetos. Estos convierten los datos de entrada en valores continuos que los modelos pueden interpretar, vectorizando los objetos en un espacio de baja dimensión, capturando similitudes y diferencias entre ellos.

Los vectores son valores numéricos que representan información en un espacio multidimensional y ayudan a los modelos de ML a encontrar similitudes entre elementos dispersos.

Cada objeto del que aprende un modelo de ML tiene varias características o atributos. Por ejemplo, las películas y programas de televisión pueden tener características como género, tipo y año de lanzamiento. Los modelos de ML pueden interpretar variables numéricas como años, pero no pueden comparar características no numéricas como género, tipo, episodios y temporadas totales. De manera que los vectores de embedding codifican datos no numéricos en una serie de valores que los modelos de ML pueden entender y relacionar. A medida que se introducen más variables, se puede refinar el modelo para condensar más información en un espacio vectorial más pequeño.

Tradicionalmente, los modelos de ML utilizan one-hot encodding para mapear variables categóricas en formas que puedan aprender. Este método de codificación divide cada categoría en filas y columnas y les asigna valores binarios.

La codificación one-hot expande los valores dimensionales de 0 y 1 sin proporcionar información que ayude a los modelos a relacionar los diferentes objetos. Por ejemplo, el modelo no puede encontrar similitudes entre la palabra "manzanaz "naranja.ª pesar de ser frutas, ni puede diferenciar entre "naranjaz "zanahoriaçomo frutas y verduras. A medida que se agregan más categorías a la lista, la codificación resulta en variables dispersas con muchos valores vacíos que consumen enormes espacios de memoria.

Por otro lado, los embeddings vectorizan objetos en un espacio de baja dimensión al representar similitudes entre objetos con valores numéricos. Los embeddings de redes neuronales aseguran que el número de dimensiones permanezca manejable con la expansión de las características de entrada. La reducción de dimensionalidad permite que los embeddings retengan información que los modelos de ML utilizan para encontrar similitudes y diferencias en los datos de entrada.

Como ya se mencionó, los embeddings se crean utilizando redes neuronales, una de las capas ocultas aprende cómo factorizar características de entrada en vectores, esto ocurre antes de las capas de procesamiento de características.

I-A2. Representación semántica:

La semántica de una palabra se refiere al significado que tiene en el lenguaje natural. Este significado puede ser tanto el sentido literal como el sentido contextual o connotativo en el que se utiliza una palabra en diferentes contextos. Cuando hablamos de embeddings y su relación con la semántica de las palabras, estamos considerando cómo estos modelos capturan y representan el significado de las palabras en un espacio vectorial.

Los embeddings capturan la semántica de las palabras al considerar el contexto en el que aparecen. Dos palabras que aparecen en contextos similares tendrán embeddings más cercanos entre sí en el espacio vectorial. Por ejemplo, en el caso de la palabra en inglés "bank", el contexto puede determinar si se refiere a un banco financiero o a la orilla de un río.

Los embeddings también permiten realizar operaciones vectoriales que reflejan relaciones semánticas entre palabras. Por ejemplo, la operación rey "hombre- "mujer"puede resultar en un vector que esté cerca del embedding de reina", lo que refleja la relación de género en el lenguaje.

La semántica distribucional es otro enfoque que se centra en el significado de las palabras basado en su distribución en contextos lingüísticos, sostiene que el significado de una palabra puede ser inferido de manera efectiva a partir de cómo se utiliza en el contexto de una oración o texto más amplio.

I-A3. Capa de embedding.:

Esta capa se encarga de asignar cada token, como una palabra o un carácter, a un vector de características en un espacio vectorial de alta dimensión. Estos vectores de características representan las palabras de manera semántica, lo que permite que la red neuronal pueda entender y procesar el texto.

Cada palabra se representa por un vector con muchos elementos, lo que permite capturar una amplia gama de relaciones semánticas entre las palabras.

I-A4. Subsampling y muestreo negativo.:

El subsampling o submuestreo de palabras es una técnica que se utiliza para reducir la frecuencia de palabras comunes en un corpus de texto durante el entrenamiento de modelos de embeddings de palabras. La idea detrás del subsampling es eliminar palabras muy frecuentes que pueden no proporcionar mucha información útil para el modelo y, al mismo tiempo, mantener palabras menos comunes que pueden ser más informativas. Esto permite mejorar la calidad de los embeddings de palabras al reducir el ruido causado por palabras extremadamente frecuentes que no aportan mucha información.

El muestreo negativo es una tecnica que funciona de tal forma para entrenar al modelo para distinguir entre palabras reales y palabras aleatorias generadas artificialmente.

- Para cada palabra de entrada en el corpus de entrenamiento, se eligen varias palabras negativas al azar del vocabulario. Estas palabras negativas son palabras que no aparecen en el contexto de la palabra de entrada.
- El modelo se entrena para predecir la palabra de entrada a partir del contexto real y para predecir palabras negativas a partir del mismo contexto. Esto ayuda al modelo a aprender a distinguir entre palabras reales y ruido aleatorio.

Esto permite que el modelo aprenda ejemplos que no estan relacionados con la palabra objetivo, mejorando la relación semántica entre palabras que si están contenidas en el documento y reduciendo la probabilidad de sobreajuste.

I-A5. Modelo Skip-gram.:

Este modelo es una técnica de embeddings que hace representaciones continuas, densas y de baja dimensión de las palabras de un vocabulario. Toma una palabra como entrada e intenta predecir las palabras del contexto (palabras circundantes) a partir de una palabra objetivo. Su arquitectura consiste en una capa de entrada, una capa oculta y una capa de salida.

- La capa de entrada es la palabra que se quiere predecir.
- La capa de salida son las palabras del contexto.
- La capa oculta representa los embeddings de la palabra de entrada.

El modelo utiliza una función objetivo de muestreo negativo para optimizar los pesos y aprender los embeddings. Cabe mencionar que el modelo se entrena durante un número determinado de épocas.

Se ha demostrado que el modelo de skip-gram produce embeddings de palabras de alta calidad que capturan el contexto y la similitud semántica entre palabras.

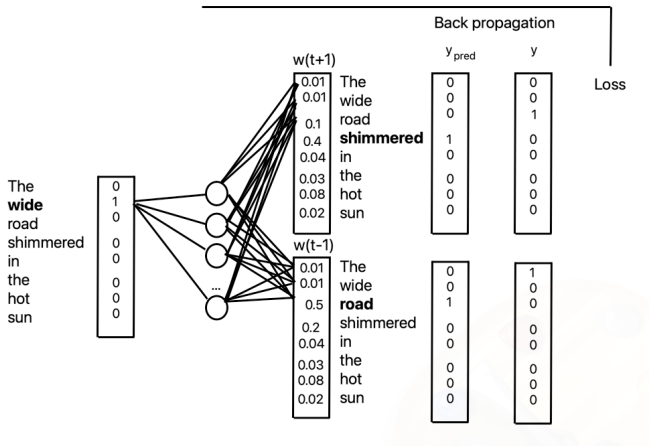

Figura 1: Fuente: medium.com. Arquitectura del modelo Skip-gram.

I-A6. Función de pérdida Sigmoide.:

La función de pérdida Sigmoid Binary Cross-Entropy (BCE) se utiliza en tareas de clasificación binaria para predecir la probabilidad de que una instancia pertenezca a una de dos clases. Esta función de pérdida se emplea para calcular la pérdida entre las predicciones del modelo y las etiquetas reales.

Se suele utilizar en escenarios en los que el objetivo es determinar la probabilidad de que una entrada pertenezca a una clase concreta de entre dos posibilidades.

I-B. Aporte

- Extraer embeddings de un corpus mediante el modelo Skip-gram
- Analizar la implementación de emebeddings en diferentes contextos, así como el preprocesamiento implicado dentro del proceso.
- Comprobar la importancia de las relaciones semánticas dentro del procesamiento de lenguaje natural

II. Desarrollo

II-A. Carga del Dataset

El documento ocupado pertenece a un fragmento de Wikipedia, el dataset es un diccionario con la siguiente forma:

```
{'id': '1',
 'url': 'https://simple.wikipedia.org/wiki/April',
 'title': 'April',
 'text': 'April is the fourth month...'
}
```

Y contiene, 205328 filas, las cuales solo se utilizaron 12,833 debido a los largos tiempos de ejecución y al tamaño del contenido. El dataset se encuentra disponible en: https://huggingface.co/datasets/wikipedia, en particular se usó la versión $20220301.simple$. De cada fila únicamente se extrajo en 'text' y fueron colocados en un vector y posteriormente divididos en palabras.

II-B. Subsampling

Debido a que en los textos es común encontrar palabras que se repiten constantemente y no cuentan con una aportación significativa dentro del contexto del documento, se decidió eliminarlas. Palabras como 'the', 'of', 'and', entre otras, se repiten con mucha frecuencia y al entrenar modelos pueden implicar un mayor costo computacional para procesarlas. Para realizar la eliminación de estas palabras se emplea la siguiente fórmula:

$$P(w_i) = \max\left(1 - \sqrt{\frac{t}{f(w_i)}}, 0\right),$$

donde $f(w_i)$ es la relación entre la cantidad de palabras w_i y la cantidad total de palabras en el dataset, y la constante t es un hiperparámetro (10^{-4} en el experimento). Cabe destacar que solo cuando la frecuencia relativa $f(w_i) > t$ se puede descartar la palabra (de alta frecuencia) w_i, y cuanto mayor sea la frecuencia relativa de la palabra, mayor será la probabilidad de que se descarte. Al trazar un histograma podremos ver que después de hacer este recorte se redujo el tamaño de las oraciones iniciales. Es importante aclarar que este proceso es diferente de la eliminación de las Stopwords, ya que la eliminación de las stopwords sugiere una eliminación definitiva a diferencia del Subsampling donde se eliminan las palabras con base en probabilidad. Y esto se hace notorio al momento de realizar un nuevo conteo de las palabras restante, donde si bien no se eliminaron de forma total este tipo de palabras, su número de repeticiones se vio mermado. Finalmente, se asigna un índice a cada uno de los tokens.

II-C. Extracción de palabras centrales y de contexto

El objetivo de este procedimiento es la búsqueda de pares de palabras respecto a otra que será designada como palabra central. De esta manera, se establece una relación entre palabras para que más tarde, dentro de los algoritmos de aprendizaje máquina, se puedan obtener los contextos. En este caso, la estructura de estos pares es: 'palabra contexto - palabra central - palabra contexto'. Esto indica que la primera palabra será considerada una palabra de contexto, la segunda una palabra central, mientras que la otra también indicara contexto. Por supuesto, la cantidad de palabras a considerar para el contexto también puede variar, por lo que se debe considerar una cantidad de palabras (tamaño de ventaneo) adecuado.

II-D. Muestreo negativo

Un muestreo negativo se sirve de las palabras centrales y de contexto obtenidas anteriormente. Se llama negativo, ya que en lugar de intentar predecir la probabilidad de todas las palabras en el vocabulario en cada entrenamiento, este busca redecir un conjunto pequeño de palabras que no aparecen en el contexto junto con la palabra central. Primero se hace el cálculo de los pesos de muestreo para cada palabra en el vocabulario. Para ello se eleva la frecuencia de cada palabra con el peso en un valor de 0.75. Luego aleatoriamente se seleccionan palabras al azar del vocabulario conforme a la probabilidad de selección de los pesos calculados anteriormente. De esta manera se generan las palabras negativas, es en esta sección donde se excluyen las palabras que ya se encuentren dentro de las palabras contextos. Por último repetimos todos pasos anteriores para cada palabra de contexto. Al final se genera una lista con todas las palabras negativas generadas.

II-E. Minilotes

Una vez extraídas todas las palabras centrales junto con sus palabras de contexto y las palabras negativas muestreadas, transformamos los datos en minilotes de ejemplos que se pueden cargar de forma iterativa durante el entrenamiento. En un minilote, el ejemplo i^{th} incluye una palabra central, sus palabras de contexto n_i y palabras negativas m_i. Debido a los diferentes tamaños de ventana de contexto, $n_i + m_i$ varía para diferentes i. Por lo tanto, para cada ejemplo, se concatenan sus palabras de contexto y palabras negativas y rellenamos con ceros hasta que la longitud de la concatenación alcance $\max_i n_i + m_i$ (‘max_len‘). Para excluir los rellenos en el cálculo de la pérdida, definimos una variable de máscara ‘mask‘. Hay una correspondencia de uno a uno entre los elementos en ‘mask‘ y los elementos en ‘context_negative‘, donde los ceros (de lo contrario unos) en ‘mask‘ corresponden a rellenos en ‘context_negative‘. Para distinguir entre ejemplos positivos y negativos, separamos las palabras de contexto de las palabras negativas en contexts_negatives a través de una variable labels. De forma similar a las máscaras, también existe una correspondencia uno a uno entre los elementos de las etiquetas y los elementos de los contextos_negativos, donde los unos (de lo contrario, los ceros) de las etiquetas corresponden

a las palabras de contexto en contexts_negatives. El resultado es un lote con un formato adecuado para un entrenamiento con Pytorch.

II-F. Implementación completa

Los códigos descritos anteriormente se probaron por separado con el fin de comprender e ilustrar mejor el preprocesamiento de datos, por lo que podemos simplificar todo lo anterior en una sola función para facilitar la entrada a los algoritmos de entrenamiento.

II-G. Embeddings y capas de embeddings

Para el entrenamiento del modelo se necesitan obtener los embeddings de las palabras, para esto se utilizó la librería de PyTorch, esta contiene una función que nos permite crear capas de embeddings y recibe como parámetro el tamaño y la dimensionalidad de la capa. Esto quiere decir que realiza una matriz de tamaño número de palabras x cantidad de características de cada palabra.

II-H. Parámetros del modelo

El modelo recibe una serie de parámetros que le ayudan a obtener las relaciones semánticas y sintácticas entre las palabras del texto. Se crean dos capas de embeddings: una para las palabas centrales y otra para las palabras de contexto.

Las palabras centrales en Skip gram se refieren a las palabras a las que se les va a poner atención a la hora de realizar una predicción del contexto.

Las palabras de contexto por el contrario a las palabras centrales son las que se intentan predecir a través de una palabra central dada.

En el programa, se utilizó la misma función de PyTorch para obtener estas capas de embeddings, dándole una dimensión a cada embedding de 100 características. Una vez hecho esto se creó un modelo de redes neuronales secuenciales al cual se le pasaron las capas de embeddings de las palabras centrales y de contexto.

II-I. Definición de la función Forward

Lo que hace la función Forward es definir la "pasada hacia adelante"del modelo, recibe como parámetros las palabras de contexto y centrales y las capas de embeddings definidas anteriormente para posteriormente vectorizarlas y asignar cada una de las palabras (centrales y de contexto) a los embeddings a los que se relacionan.

Posteriormente se hace una multiplicación entre estas nuevas matrices, que da como resultado una estimación de la similitud entre las palabras de contexto y centrales; esto nos va a servir posteriormente en el entrenamiento para calcular la función de pérdida y ajustar los valores de los embeddings durante el entrenamiento.

II-J. Entrenamiento

Una vez que se tiene la función forward, el entrenamiento se lleva a cabo con la ayuda de la función de perdida, ya que es mediante esta que se realiza la actualización de los parámetros. La salida del modelo son los mismos embeddings empleados para él entrenamiento. La función de perdida empleada es la entropíaropía cruzada binaria para el muestreo negativo con el cual se ha estado trabajando, durante el proceso se pudo observar que los resultados obtenidos de esta función de perdida se pueden tener mediante ka función sigmoidea, aunque se vería mermada la eficiencia del modelo.

II-K. Definición del ciclo de entrenamiento

El ciclo de entrenamiento inicia asignando pesos a cada uno de las capas de embeddings utilizando la técnica Xavier Uniforme, posteriormente se les aplica el optimizador Adam para que calcule los nuevos pesos durante el entrenamiento.

De la misma manera, se definen los parámetros del learning rate y se define el bucle que iterará por el número de épocas elegidas, en cada época se recorren los batchs definidos anteriormente, para calcular las predicciones se utilizan los datos de las palabras centrales y el c

II-L. Aplicación de Embeddings de palabras

Una vez que el modelo word2vec ha sido entrenado, se empleó la similitud del coseno de los vectores de palabras del modelo entrenado para encontrar palabras del diccionario que sean más similares semánticamente a una palabra de entrada. Algo a recordar, es la importancia de tener un dataset de un tamaño considerable para obtener n

III. RESULTADOS

A partir del dataset de 12,833 textos, se obtuvieron:

- El tamaño del vocabulario sin frecuencia mínima es 109,065
- El tamaño del vocabulario con frecuencia mínimo de 10 es 9,739

Al realizar el proceso de entrenamiento del modelo, puede ser muy tardado y podemos encontrar palabras con mucha frecuencia en el texto que regularmente no nos representan relaciones semánticas significativas en un contexto dado, en nuestro caso podemos encontrar:

Tabla I: Plabras frecuenes

Palabra	Frecuencia
the	51728
of	32483
and	24746
in	19820
a	18490
is	18372
to	16771
The	10811
are	8759

Al aplicar el submuestro obtenemos una reduccion significativa de las palabras:

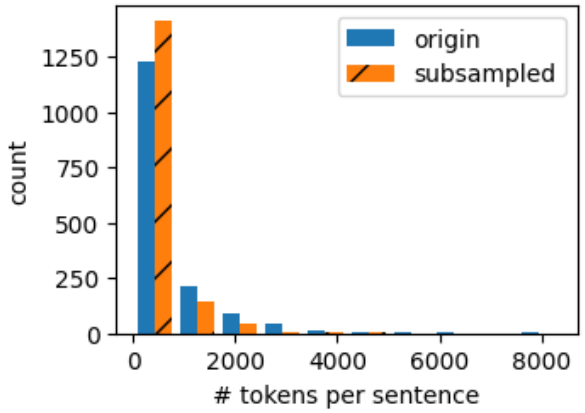

Figura 2: histograma del antes y después del submuestreo

Tabla II: Reducción de palabras frecuentes

Palabra	Antes	Después
the	51728	2280
of	32483	1865
and	24746	1543
in	19820	1469
a	18490	1434
is	18372	1344
to	16771	1358
The	10811	1005
are	8759	988

En el proceso de entrenamiento se realizaron 20 épocas y con learning rate de 0.002, obteniendo:

```
loss 0.399, 22386.8 tokens/sec on cuda:0
loss 0.370, 44812.9 tokens/sec on cuda:0
...
loss 0.300, 430202.7 tokens/sec on cuda:0
loss 0.299, 291823.5 tokens/sec on cuda:0
```

Una vez entrenado el modelo Word2vec, usando la similitud coseno de los vectores de palabras del modelo entrenado para encontrar palabras del diccionario que sean más similares semánticamente a una palabra de entrada, obtuvimos:

Tabla III: Palabras similares a: **communities**

Palabra	Similitud
Erik	0,400
Zombies	0,390
conflicts	0,367
Gino	0,367

Tabla IV: Palabras similares a: **computer**

Palabra	Similitud
Ethernet	0,389
Noel	0,370
Effendi	0,358
Winner	0,350

Tabla V: Palabras similares a: **Valley**

Palabra	Similitud
known	0,427
bonsai	0,407
archosaur	0,386
life-style	0,377

Como observamos los resultados son muy inexactos, los embeddings obtenidos no nos dan una representación adecuada para calcular la similitud coseno y conocer las palabras más similares unas de otras, esto se debe a la pequeña cantidad de datos utilizados para el entrenamiento, para obtener una representación más exacta se necesita una cantidad enorme de datos, considerando que los costos computacionales serán altos.

IV. Conclusiones

La implementación del modelo WOrd2Vec para la obtención de embeddings fue relativamente rápida al utilizar el texto de Don Quijote. Sin embargo, los resultados en los valores de los embeddings fueron muy bajos. Esto se puede deber a que el corpus es relativamente pequeño y no contiene la suficiente información para obtener valores de similitud entre palabras mas altas.

A través de diversas etapas de preprocesamiento como el subsampling de palabras, y mediante la generación de embeddings con un modelo de skip-gram se construye un sistema adaptable a diferentes conjuntos de datos y temas específicos, para poder extraer información relevante y relaciones semánticas en una gran cantidad de contextos, en este caso específicamente para el corpus de texto extraído de un conjunto de datos de Wikipedia. Lo que permitió la búsqueda de información, como palabras similares a otras según su contexto, concretamente, según su relación semántica.

Se probaron también corpus mas grandes (de mas de 2 millones de palabras), pero en este caso el rendimiento del algoritmo se empezó a ver mas reducido, aunque para la cantidad de datos procesados, el tiempo de ejecución fue aceptable. Esto aun se puede mejorar tras aplicar una mejor selección de hiperparámetros en la parte de entrenamiento. También se propone para mejora una limpieza de texto pues en algunos textos, las palabras de los títulos que tienen mayúsculas no se consideran en el conteo con las que son todas minúsculas, además de que las palabras seguidas de un signo de puntuación se consideraban como tokens junto con los signos, dando lugar a un conteo de palabras erróneo.

Referencias

[1] Amazon.com. https://aws.amazon.com/what-is/embeddings-in-machine-learning/. Recuperado el 11 de mayo de 2024.
[2] Cloudflare.com. https://www.cloudflare.com/learning/ai/what-are-embeddings/ Recuperado el 11 de mayo de 2024.
[3] Jiang, J. (2021, mayo 5). What is embedding and what can you do with it. Towards Data Science. https://towardsdatascience.com/what-is-embedding-and-what-can-you-do-with-it-61ba7c05efd8 Recuperado el 11 de mayo de 2024.
[4] Malingan, N. (2023, febrero 22). Skip-Gram Model in NLP. Scaler Topics. https://www.scaler.com/topics/nlp/skip-gram-model/ Recuperado el 11 de mayo de 2024.

[5] Sarkar, D. (s/f). Implementing deep learning methods and feature engineering for text data: The Skip-gram model. KDnuggets. https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-skip-gram.html Recuperado el 11 de mayo de 2024.
[6] Huang, D. J. (2021, junio 11). Learning Day 57/Practical 5: Loss function — CrossEntropyLoss vs BCELoss in Pytorch; Softmax vs sigmoid; Loss calculation. Dejunhuang. https://medium.com/dejunhuang/learning-day-57-practical-5-loss-function-crossentropyloss-vs-bceloss-in-pytorch-softmax-vs-bd866c8a0d23 Recuperado el 11 de mayo de 2024.
[7] Unirioja.es. https://dialnet.unirioja.es/servlet/articulo?codigo=6506616 Recuperado el 11 de mayo de 2024.