UNIVERSIDADE DA CORUÑA

Grado en Inteligencia Artificial

Fundamentos de Procesamiento de Lenguaje Natural – Curso 2024/25

Guía de la práctica

Modelos neuronales para representación vectorial de palabras

Introducción

El éxito de los modelos de lenguaje actuales como GPT, Gemini, LLaMa, Mistral, DeepSeek, etc. se sustenta en varios pilares:

- Disponibilidad de gran cantidad de datos.
- Arquitectura Transformers.
- Representaciones de palabras en forma de vectores.

2000-2013

Representaciones discretas de palabras y modelos SVM

2013-2018

Representaciones vectoriales estáticas y entrenamiento en grandes cantidades de datos

2018-2023

Representaciones distribuidas contex tualizadas: Representaciones vectorial es, grandes cantidades de datos, y arquitectura Transformers.

2023-Presente

Grandes modelos de lenguaje de propósito general: Representaciones vectoriales, grandes cantidades de datos, arquitectura Transformers,y Reinforcement Learning with Human Feedback

Representaciones de palabras en PLN

Representaciones *discretas* de palabras:

1. Incapaces de manejar relaciones semánticas entre palabras.
2. Palabras cercanas semánticamente no se representan próximas entre ellas.
3. Muy populares hasta la explosión del deep learning (~2013).

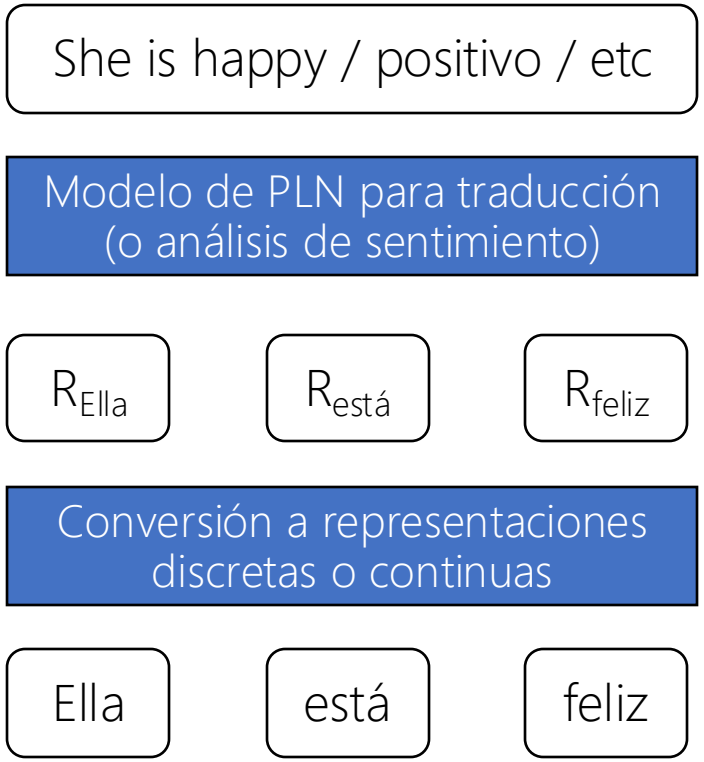
Representaciones *continuas* de palabras:

1. Permiten representar relaciones semánticas entre palabras de manera eficaz.
2. Palabras semánticamente cercanas se representan cercanas en el espacio de salida vectorial.
3. Muy populares después del año 2013, y una de las dos principales razones (junto con las capacidades de entrenamiento a gran escala) que nos han permitido llegar hasta modelos como GPT.

Representaciones de palabras en PLN

Estas representaciones, discretas o continuas, se utilizan como entrada para resolver distintas tareas de procesamiento de lenguaje natural:

- Traducción automática
- Análisis del sentimiento (p.ej. texto positivo o negativo)
- Búsqueda de respuestas
- Modelado de lenguaje
- ...



Representaciones *discretas* de palabras

Ejemplo de Representación Discreta (One-Hot Encoding)

Supongamos un vocabulario simple: {Rey, Reina, Hombre, Mujer, Perro, Gato}

Rey: [1, 0, 0, 0, 0, 0]

Reina: [0, 1, 0, 0, 0, 0]

Hombre: [0, 0, 1, 0, 0, 0]

Mujer: [0, 0, 0, 1, 0, 0]

Perro: [0, 0, 0, 0, 1, 0]

Gato: [0, 0, 0, 0, 0, 1]

Representaciones *discretas* de palabras

Cada palabra del vocabulario se asigna a un vector único en el que sólo una posición tiene el valor 1, y todas las demás posiciones son 0.

Directo y fácil de entender.

Carece de la capacidad para representar relaciones entre palabras: 'hombre' no se parece más a 'mujer' o 'rey' que 'perro' o 'gato'.

Esto fue un gran problema de cara a desarrollar modelos de PLN que generalicen bien.

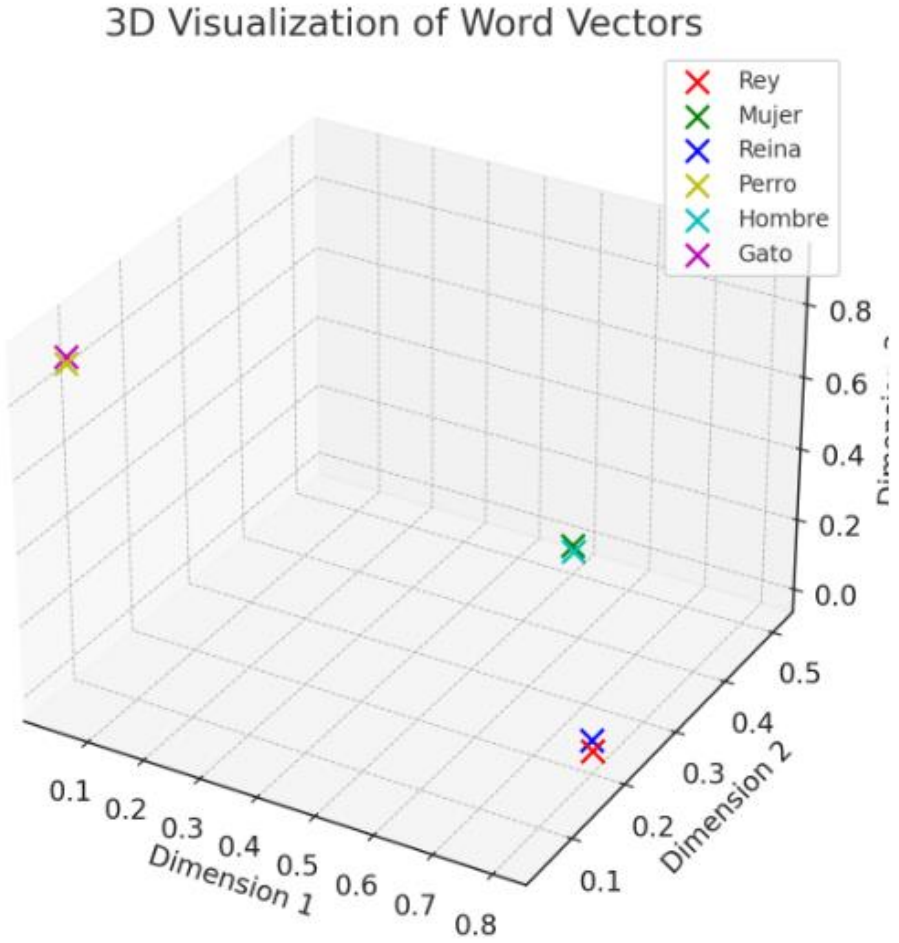
Representaciones *continuas* de palabras

Las palabras se representan en espacios vectoriales continuos de baja dimensionalidad.

Baja dimensionalidad: normalmente unos pocos cientos de dimensiones. Un ejemplo (artificialmente bajo) con solo 3 dimensiones:

Rey: [0.8, 0.2, 0.0]
Reina: [0.79, 0.21, 0.01]
Hombre: [0.5, 0.5, 0.0]
Mujer: [0.49, 0.51, 0.0]
Perro: [0.05, 0.05, 0.9]
Gato: [0.04, 0.06, 0.9]

Representaciones *continuas* de palabras



Representaciones *continuas* de palabras

"Rey" y "reina" están muy cerca uno del otro en el espacio vectorial, reflejando su relación semántica en el ámbito de la monarquía.

De manera similar, "hombre" y "mujer" están cercanos entre sí.

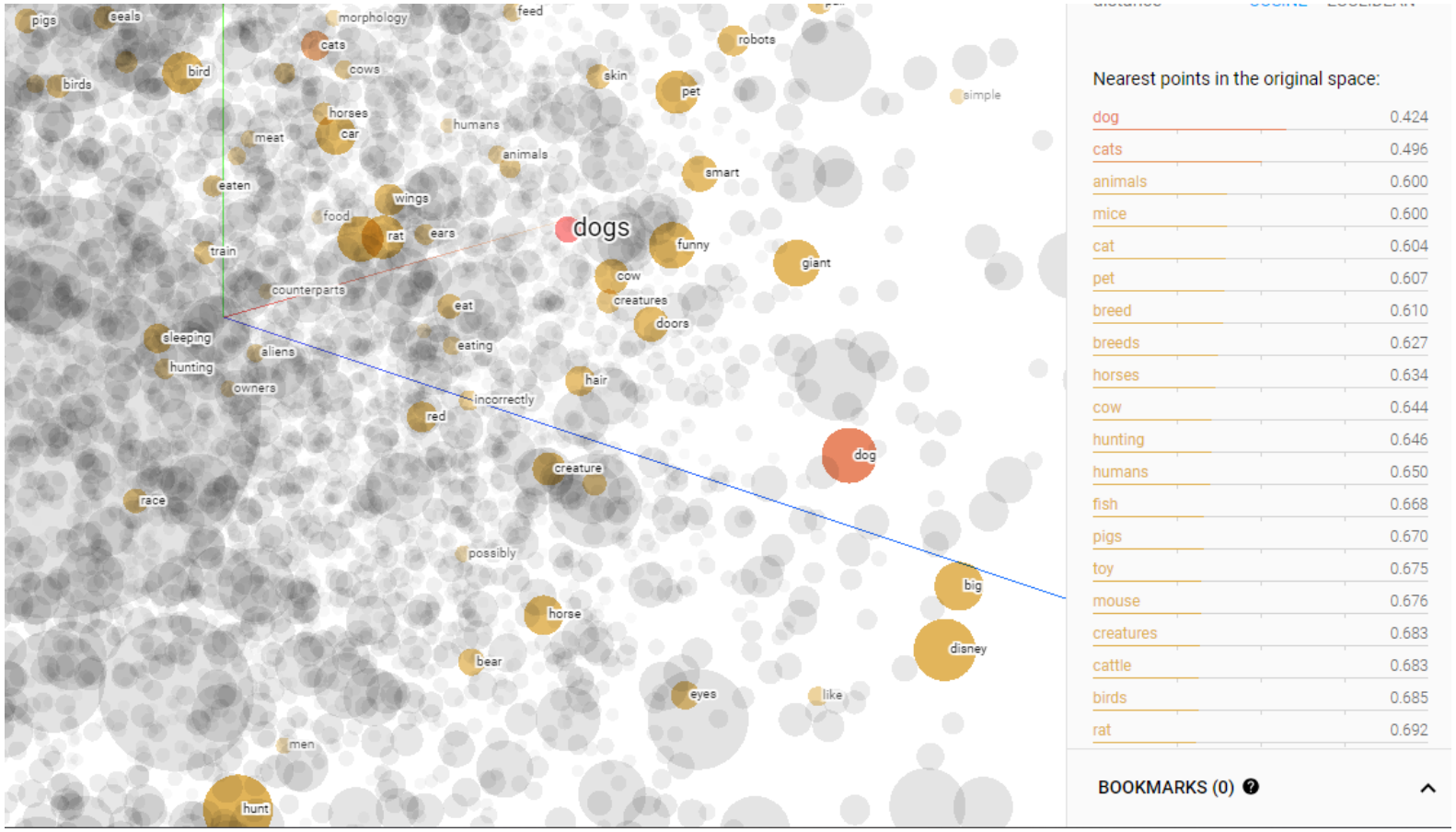
Por otro lado, "perro" y "gato", siendo ambos animales domésticos, también se representan con vectores cercanos en otra región del espacio vectorial.

También se pueden capturar relaciones más complejas y sutiles, como analogías (por ejemplo, la famosa relación "rey" es a "hombre" como "reina" es a "mujer") mediante operaciones vectoriales.

Esto es algo que simplemente no es posible con las representaciones discretas de one-hot encoding, donde cada palabra es igualmente distante de todas las demás.

Representaciones *continuas* de palabras

Demo de visualización: https://projector.tensorflow.org/



Representaciones *continuas* de palabras

¿Cómo podemos aprender dichas representaciones?

¿Cómo hacer que el vector aprendido para 'perro' se parezca más al de 'gato' que al de las palabras 'mujer', 'cuchara' o 'volar' ?

El problema que aprenderemos a resolver en esta práctica: cómo aprender representaciones de palabras que puedan servir como entrada a multitud de tareas de PLN.

“ You Shall Know a Word by the Company It Keeps ”

John Rupert Firth (1957, Lingüista)

Modelos neuronales para representación vectorial de palabras

1. Modelo basado en la predicción de palabras dada su contexto:
    - Entrada: una ventana con varias palabras del contexto anterior y posterior a la palabra objetivo.
    - Salida: La palabra objetivo.

2. Modelo basado en la predicción de contexto dada una palabra:
    - Entradas: Pares (palabra objetivo, una palabra del contexto)
    - Salida: Una etiqueta diciendo si ese par de palabras ocurren en un mismo contexto.

Modelo 1, basado en predicción de palabras dado su contexto

Creación de las muestras de entrenamiento (ejemplo para contexto de tamaño n=5, cogiendo las dos palabras anteriores y las dos siguientes).

Ejemplo de algunas muestras de entrenamiento para la oración: "*Frodo y Sam miraron al horizonte con determinación*"

- Entrada: ["y", "Sam", "al", "horizonte"] Salida: "miraron"
- Entrada: ["Sam", "miraron", "horizonte", "con"] Salida: "al"
- Entrada: ["miraron", "al" "con", "determinación] Salida: "horizonte"

Modelo 1, basado en predicción de palabras dado su contexto

Conversión de strings (palabras) a identificadores enteros numéricos, para que puedan ser entendidos por el modelo de Keras.

- Entrada: ["y", "Sam", "al", "horizonte"] Salida: "miraron"
- Entrada: ["Sam", "miraron", "horizonte", "con"] Salida: "al"
- Entrada: ["miraron", "al" "con", "determinación] Salida: "horizonte"

- Entrada: [101, 257, 58, 432] Salida: 389
- Entrada: [257, 389, 432, 165] Salida: 58
- Entrada: [389, 58, 165, 590] Salida: 432

Modelo 1, basado en predicción de palabras dado su contexto

Enviaremos las muestras en batches para el entrenamiento en Keras. Podemos empezar con un tamaño de 128, pero tenéis libertad para adaptarlo según vuestras preferencias.

Batches grandes: más eficientes, pero potencialmente peor resultados.

Batches pequeños: entrenamiento más lento, pero posible mejor generalización.

Ejemplo de batch (3) de entradas entradas con n=5

101 257 58 432
257 389 432 165
389 58 165 590

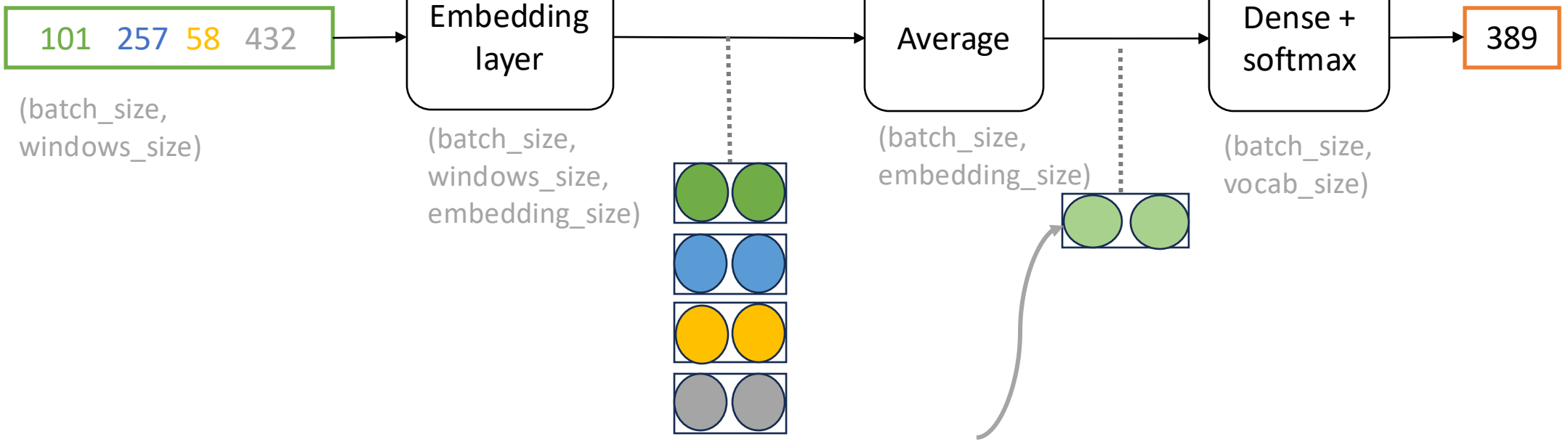
Salidas asociadas

389
58
432

Modelo 1, basado en predicción de palabras dado su contexto

La capa embedding obtiene un vector para cada identificador de palabra. Podemos ver esta capa como un diccionario que mapea identificadores números a vectores.

Usamos una capa de salida para predecir la palabra objetivo



Hacemos la media de las embeddings del batch para cada dimensión del vector

Modelo 1, basado en predicción de palabras dado su contexto

Capas de Keras de interés para el desarrollo del primer modelo:
https://www.tensorflow.org/api_docs/python/tf/keras/Input

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Lambda
(para el cálculo del vector medio, usando lambda x: K.mean(x, axis=1) como valor para el parámetro function, con import tensorflow.keras.backend as K)

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

Modelo 1, extracción de los vectores de embeddings

Podéis extraer el diccionario que mapea IDs de palabras a embeddings con el siguiente comando:

embeddings = model.get_layer(nombre_capa_embedding).get_weights()[0]

Donde model es el modelo de Keras y nombre_capa_embeddings es el identificador que le habéis dado a la capa Embedding a través de su atributo name.

Para acceder a la embedding de una palabra podemos utilizar directamente embeddings[word_id]

Disponéis de un ejemplo dentro de materiales/utils.ipynb

Modelo 2, basado en predicción de contexto en base a una palabra objetivo

Creación de las muestras de entrenamiento (ejemplo para contexto de tamaño n=5, cogiendo las dos palabras anteriores y las dos posteriores).

Ejemplo de algunas muestras de entrenamiento para la oración: "*Frodo y Sam miraron al horizonte con determinación*"

Entrada: "miraron"	Salida: ["y", "Sam", "al", "horizonte "]
Entrada: "al"	Salida: ["Sam", "miraron", "horizonte", "con"]
Entrada: "horizonte"	Salida: ["miraron", "al" "con", "determinación"]

Modelo 2, basado en predicción de contexto en base a una palabra objetivo

Entrada: "miraron”

Salida: ["y", "Sam", "al", "horizonte"]

Por simplicidad, puede transformarse esta muestra de entrenamiento en cuatro muestras equivalentes, que trataremos de manera separada:

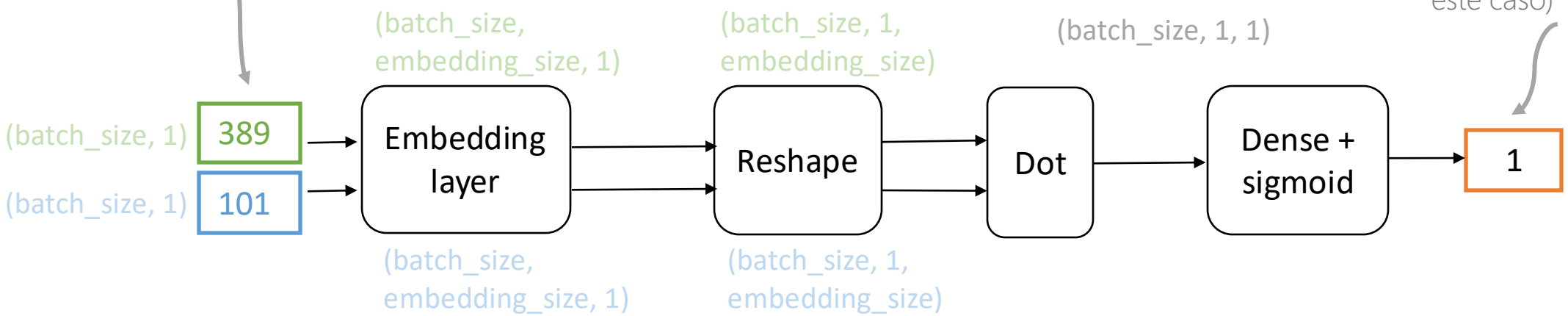
Entrada: ["miraron", "y"]	Salida: 1
Entrada: ["miraron", "Sam"]	Salida: 1
Entrada: ["miraron", "al"]	Salida: 1
Entrada: ["miraron", "horizonte"]	Salida: 1

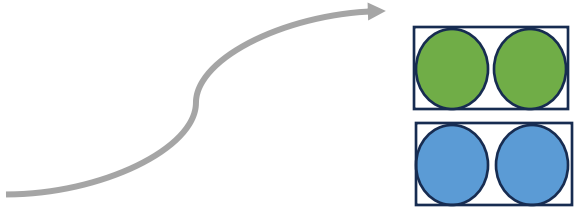
Ejemplo de entradas	Salidas asociadas
389 101	1
389 257	1
389 58	1
389 432	1

Modelo 2, basado en predicción de contexto en base a una palabra objetivo

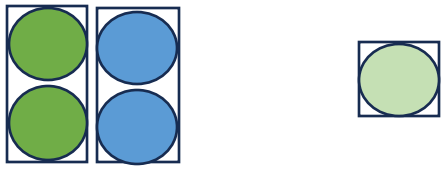
En el caso del modelo basado en contexto, la entrada al modelo son dos entradas separadas (la palabra central y una palabra de su contexto)

Generamos una etiqueta de salida (que siempre serán unos, en este caso)





La capa embedding obtiene una representación separada para cada palabra



Aplicamos un Reshape para poder multiplicar los vectores mediante la operación Dot

Aplicamos un Reshape para poder multiplicar los vectores mediante la operación Dot. Valores altos se asocian con palabras similares (que deberían ocurrir en el mismo contexto) y valores bajos con palabras que ocurren en distintos contextos

Modelo 2, basado en predicción de contex to en base a una palabra objetivo

Motivación para entradas separadas y la operación dot en modelos basados en contexto:

Ejemplo con entradas separadas:

Rey: [2, 3]
Reina: [2, 2.5]
Dot product: (2*2) + (3*2.5) = 4 + 7.5 = 11.5.

Rey: [2, 3]
Manzana: [-2, -1]
Dot product (2*-2) + (3*-1) = -4 - 3 = -7

Modelo 2, basado en predicción de palabras dado su contexto

Capas de Keras de interés para el desarrollo del segundo modelo:
https://www.tensorflow.org/api_docs/python/tf/keras/Input

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Reshape

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dot

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

Evaluación cualitativa de embeddings

Objetivo:

1. Comprender la estructura y calidad de las embeddings de palabras generadas por nuestro modelo.
2. Identificar relaciones semánticas capturadas por dichas embeddings.

Técnicas Utilizadas

1. Visualización con t-SNE: Técnica de reducción de dimensionalidad no lineal adecuada para incrustar datos de alta dimensión en un espacio de dos o tres dimensiones, facilitando su visualización.
2. Similitud del Coseno: Medida que calcula la similitud entre dos vectores de un espacio con producto interno, valorada en función del coseno del ángulo entre ellos.

Visualización con t-sne

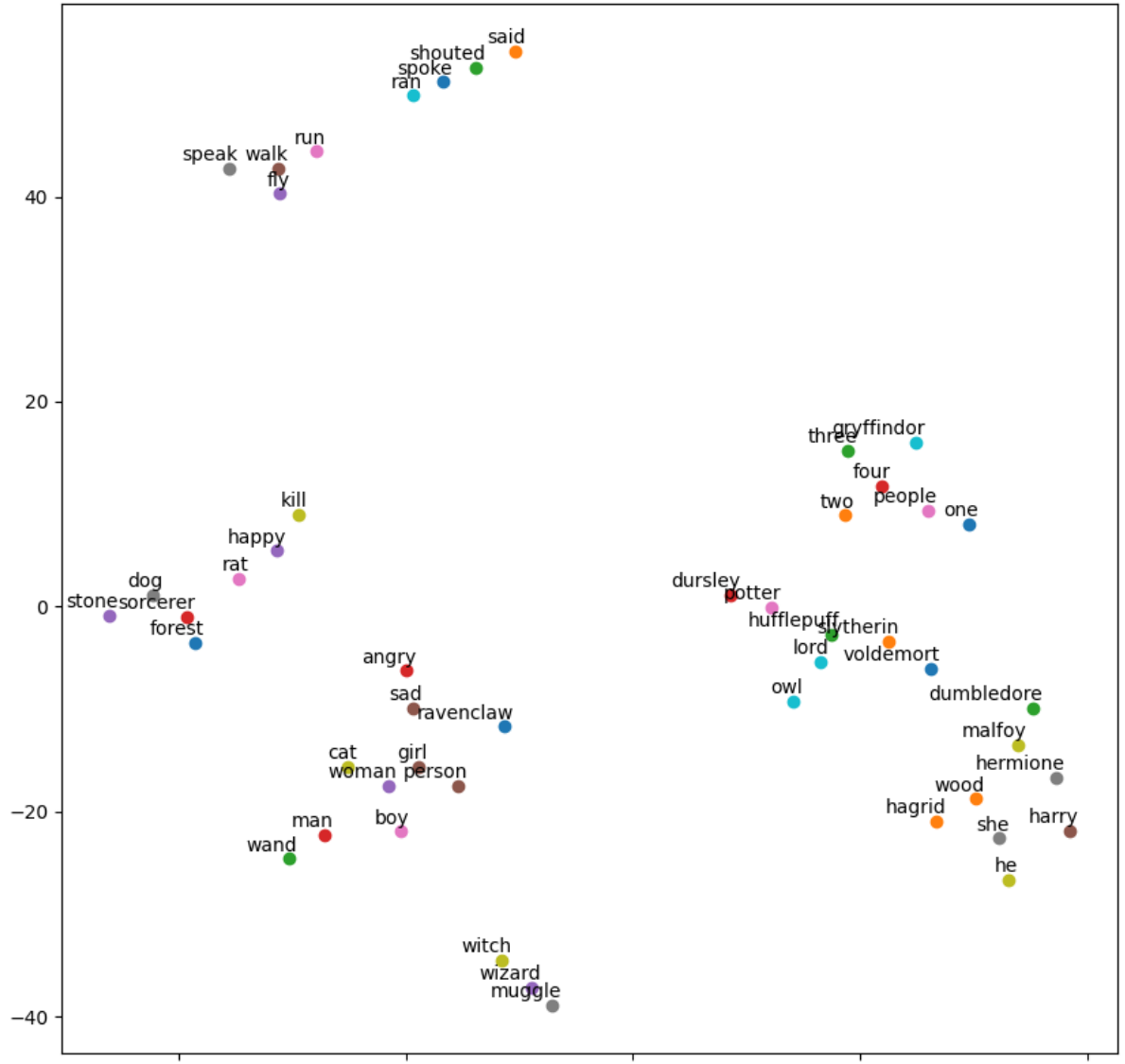
Descripción: Transforma embeddings de alta dimensión a un espacio de baja dimensión (2D o 3D) preservando la similitud entre puntos.

Pasos:

1. Selección de embeddings para visualizar.
2. Aplicación de t-SNE para reducir la dimensionalidad.
3. Visualización en 2D o 3D para interpretar la proximidad y agrupaciones de palabras.

Se proporciona código de ejemplo en materiales/embeddings_visualization.ipynb.

Ejemplo de visualización t-sne para algunas palabras del primer libro de Harry Potter



Similitud del coseno

Intuición: palabras con mayor similitud reflejan palabras que ocurren en contextos similares, y que por lo tanto están relacionadas semánticamente.

Pasos:

1. Selección de palabras objetivo sobre las que vamos a calcular la similitud.

2. Aplicación de la función de similitud del coseno con el resto de las palabras de vocabulario.

3. Para cada palabra objetivo, seleccionar las X palabras con vectores más similares (las 10 palabras más similares)

Keras – keras.preprocessing.text

Módulo de ayuda para el tratamiento de texto en lenguaje natural, antes de ser enviado a la red neuronal.

Se usará en la práctica para:

1. Crear y entrenar el Tokenizador (para saber cómo separar los tokens del dataset de entrada).
2. Transformar palabras en IDs (para que sirvan de entrada al modelo de Keras).
3. Conocer el tamaño del vocabulario de entrada, para saber el número de dimensiones de la clase de salida.

Se muestra un ejemplo sobre como entrenar y usar el Tokenizador en materiales/tokenizer.ipynb

Keras – tensorflow.gpu

Para el modelo neuronal, es posible entrenar con GPU para poder entrenar modelos en un tiempo razonable. De todos modos, debería ser posible ejecutar los modelos en CPU también sin problemas.

Para ello, es posible usar los recursos que libremente da https://colab.research.google.com/

En materiales/gpu.ipynb se explica como seleccionar que los modelos se ejecuten en GPU y comprobar que el código Python está detectando la GPU.

Fases principales (para cada módelo)

1. Leer el dataset y crear las muestras de entrenamiento: uso del Tokenizer y definición de la función que desplace la ventana para crear las muestras de entrenamiento.

2. Definir el modelo y entrenarlo.

3. Evaluación de las embeddings para las palabras propuestas, mediante similitud del coseno y visualización t-sne.