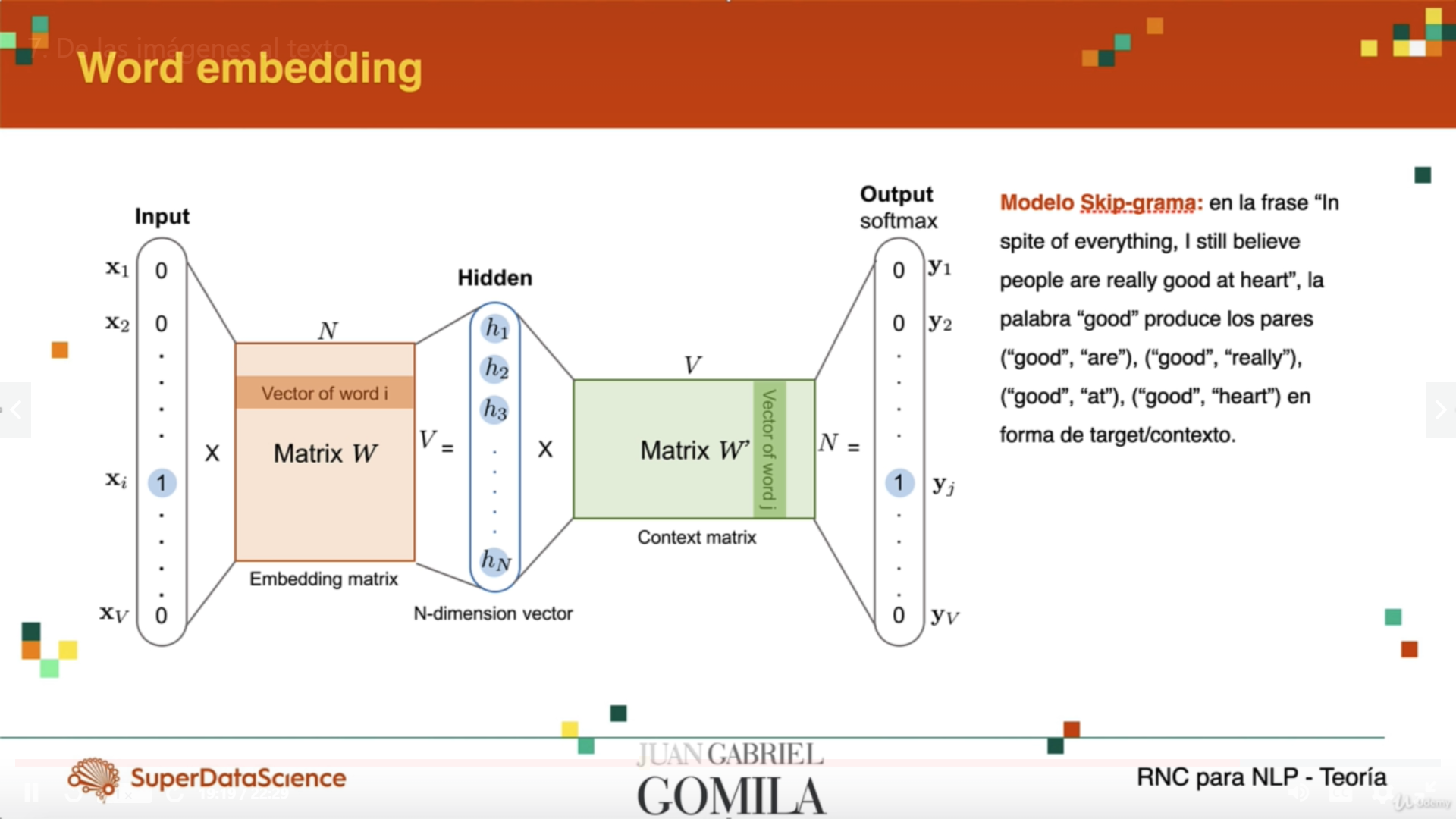
Idea detrás de una RNC para NLP



La idea es que vamos a tener un vector de entrada, como el vector **one hote encoding.** Entonces, todo lo que necesitamos es que este vector, indicando que aparece la palabra i-ésima en la frase que estamos analizando sea multiplicado matricialmente por lo que llamamos la **"matriz de embeddings"** y que el producto de este vector columna por la matriz de embeddings nos de el vector codificado y que pertenecerá a la **capa oculta**. Como estamos multiplicando un vector de puros 0´s con un solo valor igual a 1 por la matriz de embeddings lo que obtendremos será la fila i-ésima de la matriz de embeddings. Entonces, esta matriz no es ni más ni menos que una lista de los vectores originales de las palabras embebidas(transformadas) en un espacio dimensional de menor dimensión. Entonces, cuando hacemos el producto de cualquier vector hardcodeado por esta matriz, lo único que hacemos es obtener el vector embebido que es la fila i-ésima en cuestión, donde esta la palabra que estamos analizando.

Ahora bien, la pregunta sería ¿cómo entrenamos esta matriz? ¿cómo encontramos las mejores coordenadas para esta tarea? ¿cómo aprendemos cuáles son los vectores embebidos para cada palabra que hemos transformado en ceros y unos? La idea va a ser que en un momento dado, éste vector embebido en un espacio de dimensión inferior, mucho más pequeño el tamaño original de nuestro vocabulario, tendrá que ser reconstruido. Queremos volver a partir del vector embebido a reconstruir el vector original de modo que hagamos una operación contraria. Esto es lo que llamaremos la **"matriz de contexto" (W)** la cual no tiene ninguna relación con la **matriz W**, solamente en dimensión para poder multiplicar, pero básicamente lo que vamos hacer es multiplicarla por otra matriz para intentará reconstruir la entrada original. Entonces, la idea es que vamos a querer multiplicar el vector embebido por otra matriz, la matriz de contexto (ya se verá por que más adelante) y obtener un vector del mismo tamaño que tenía el vocabulario originalmente, pero con algo más. Es decir, voy a querer añadirle un significado dependiendo del contexto de las palabras que rodearán a la palabra original que codificamos, entonces vamos a querer este tipo de salidas como en la imagen. El vector va a necesitar tener una correlación semántica en la salida con la que tenía originalmente de modo que en la salida vamos a buscar qué palabras son las que salían o son las que salen cerca de la palabra de las palabras que habías codificado en un inicio. De hecho, en el lenguaje humano, las palabras que rodean a una palabra dada es lo que llamamos el contexto y los humanos pues damos un contexto u otro dependiendo de lo que estamos hablando. Aquí es donde entra el juego el primer modelo que es el "Skip-grama" el cual analiza por ventanas por ejemplo de 2 las dos palabras antes y después que van de la palabra a analizar. En el ejemplo, las palabras "are,"really","at" y "heart" en el output deberían de tener un valor muy cercano o igual a 1. La parte internaste de todo está sobre todo en la matriz de embeddings nos devolverá en el espacio vectorial que hayamos decidido, palabras cercanas en coordenadas similares. En particular, es muy importante darnos cuenta de que, si dos palabras tienen un significado parecido, es bastante probable que, a partir de los datos analizados en el corpus, tengan contextos similares.

En conclusión, este Word embedding, utilizando el modelo "Skip-grama" trata de reducir la dimensión en un espacio vectorial que sea más pequeño que el número de palabras, forzando a que esa reducción de información traiga cerca palabras con relaciones semánticas entre ellas. Lo hace de modo que esas relaciones conserven el álgebra matemática de los vectores de modo que se puedan sumar restar o llevar a cabo las operaciones típicas de inteligencia artificial.