Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Inteligencia Artificial
Prof. Ing. Luis Carlos Hernández Vega
Eros Hernández Romero - 2014081083
Tutorial #5 Representación de Palabras por Vectores

Conclusiones

El objetivo principal de este tutorial es **representar palabras como si fueran vectores**, lo cual llaman "**Palabras embebidas**". Cuando se habla de tareas de reconocimiento de imágenes o voz, se toman grandes conjuntos de datos (Como pixeles, coeficientes) para obtener información necesaria y poder realizar con éxito dichas tareas.

Respecto al reconocimiento de palabras o procesamiento de un lenguaje natural, usualmente se trata como **símbolos discretos** en donde, por ejemplo, la palabra 'gato' podría estar representada como 'Id537' o la palabra 'perro' representada por 'Id143'. Esto provoca que el modelo en realidad **aproveche muy poco lo que ha aprendido** sobre un objeto, cuando está procesando los datos de otros. Esto provoca que los datos se **dispersen** y a la vez provoca que se requieran aún más datos para poder generar modelos estadísticos. Se justifica la implementación de este tutorial pues la **representación de vectores** puede solucionar este problema.

El punto es que los modelos de espacio vectorial, puedes representar las palabras analizadas en un espacio vectorial continuo, es decir que las **palabras semánticamente parecidas**, se pueden asignar a **puntos cercanos**. El proceso se basa en dos enfoques:

- Métodos basados en Recuento: Análisis Semántico. Calculan las estadísticas de la frecuencia con la que una palabra coincide con sus palabras vecinas y luego mapean estos conteos.
- Métodos Predictivos: Modelos de Lenguaje Probabilístico. Intentan predecir directamente una palabra de sus vecinos en términos de vectores.

En el caso de este tutorial, el modelo que se utiliza es el **Predictivo**, pues lo que hace el programa es intentar predecir una palabra en específico de sus vecinos.

Se nos introduce el modelo '**Skip-gram**', el cual es capaz de predecir palabras tomando como punto de referencia una palabra en específico. Basándome en la explicación del tutorial:

Consideramos: "el rápido zorro marrón saltó sobre el perro perezoso"

1. Se forma un conjunto de pares de datos de palabras y los contextos en los que aparecen. Algo así como:

([el, marrón], rápido), ([rápido, zorro], marrón), ([marrón, saltado], zorro)

2. Lo que hace el modelo, es invertir los contextos y los objetivos, e intenta predecir cada palabra de contexto con su palabra destino. Es decir, que si una entrada a la red neuronal fuera 'rápido', la salida sea 'el' o 'marrón'.

Cómo podría aplicarse a un problema real

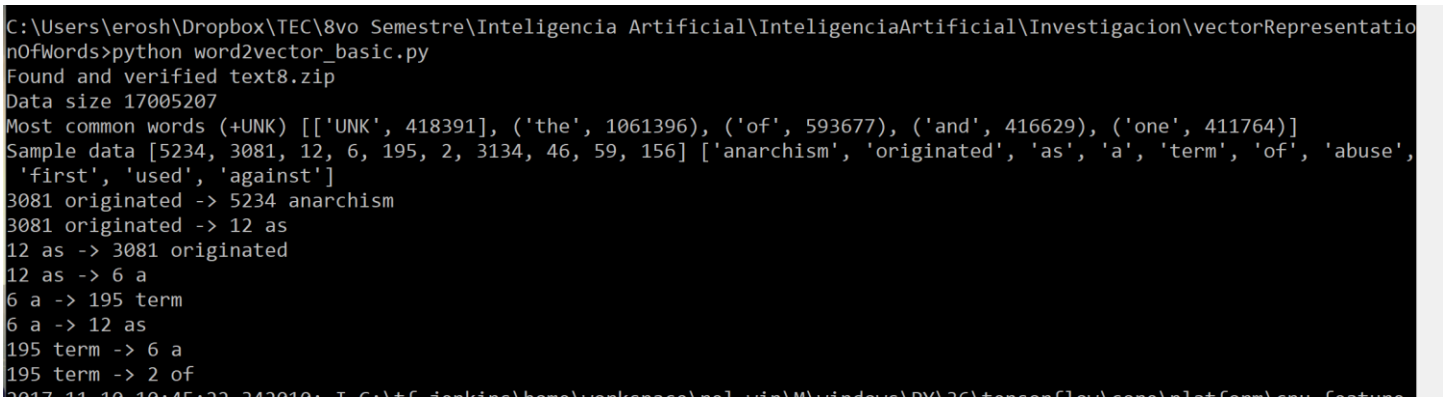
Se puede aplicar para un proceso llamado '**Sentient Analysis**', en donde por ejemplo, se puede crear un 'bot' con una red neuronal, que reciba muchos 'Tweets' de Twitter como datos de entrada y que comience a detectar palabras clave, esta las pondría dentro de un clasificador que determina si los resultados de dichas palabras son positivos o negativos. Entonces, luego una empresa como Starbucks puede venir y verificar 'Tweets' que contengan su hashtag y determinar si los resultados han sido positivos o negativos.

Dejo una página de referencia para algunos trabajos prácticos:

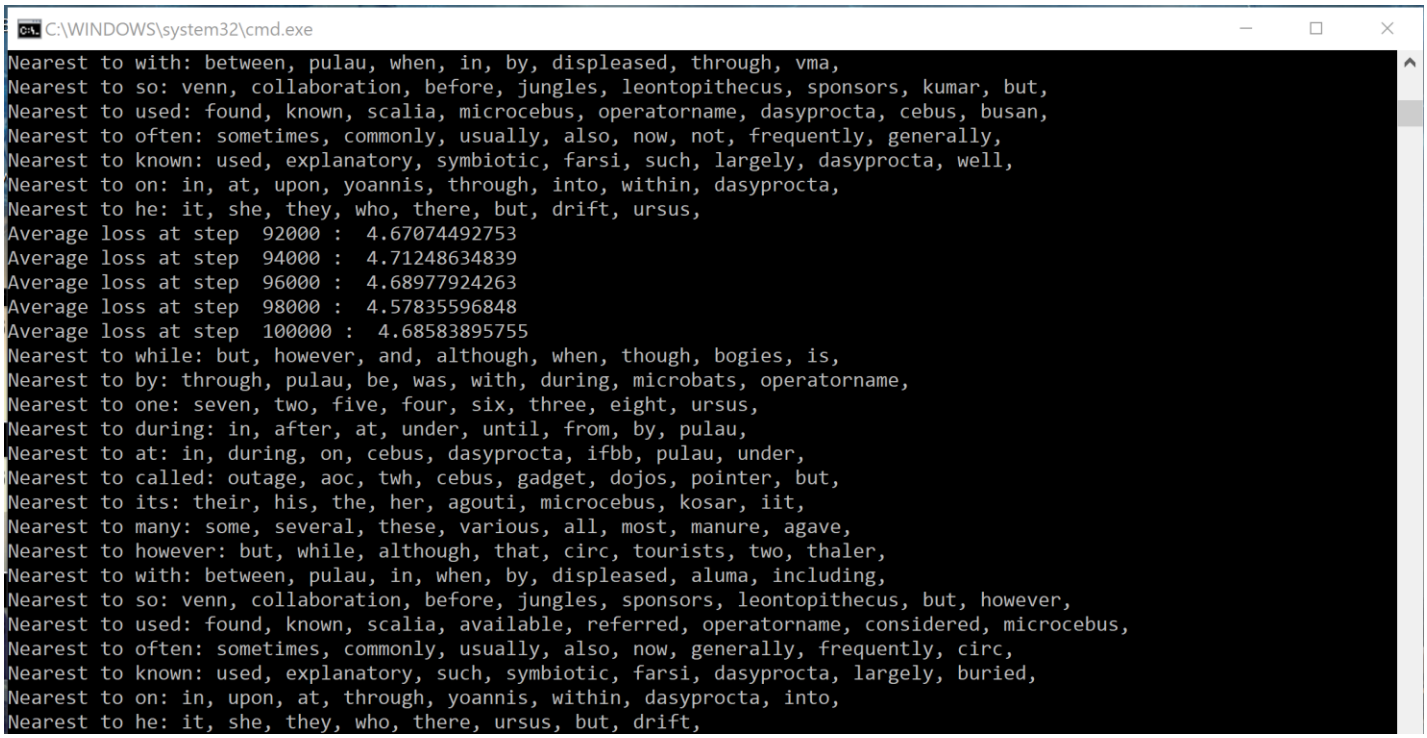
https://www.quora.com/What-are-some-practical-use-of-word2vec

Screenshots de Resultados

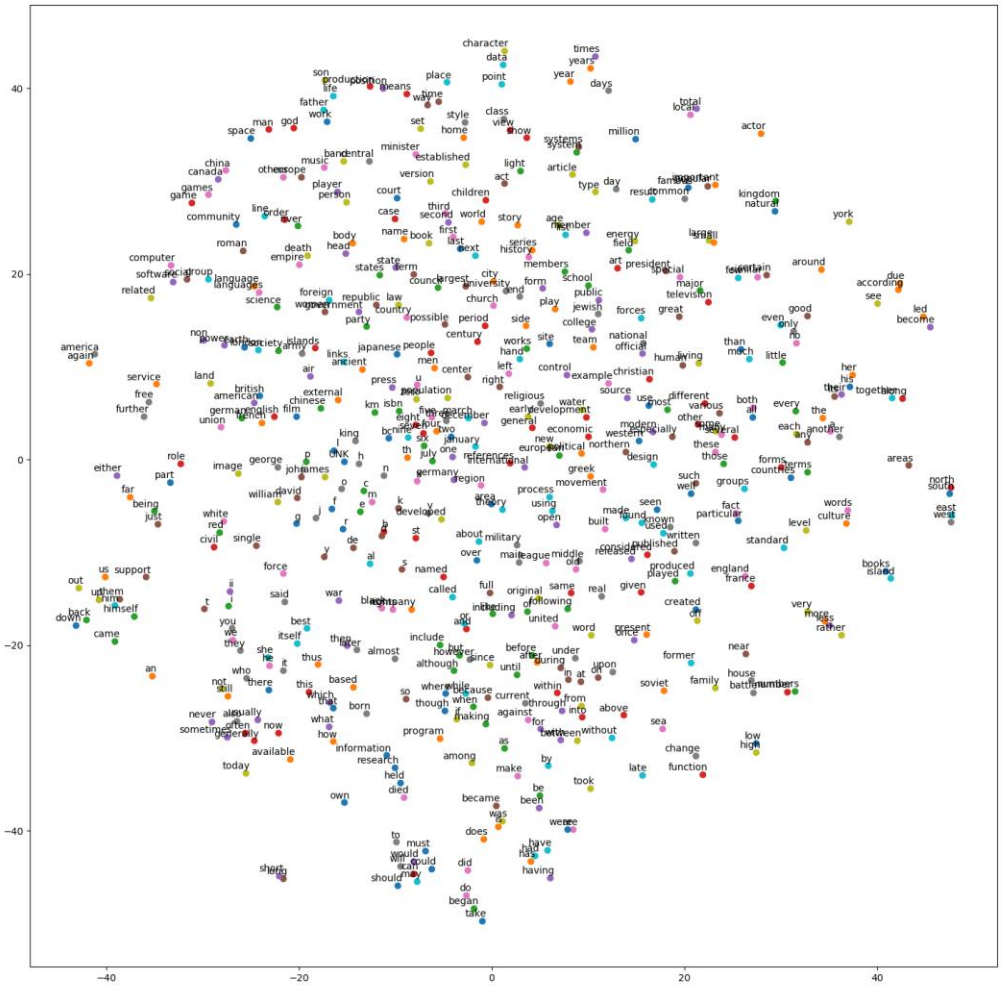
Inicialmente, se debe ejecutar el programa 'word2vector_basic.py', y se comienza con el entrenamiento de la red neuronal.



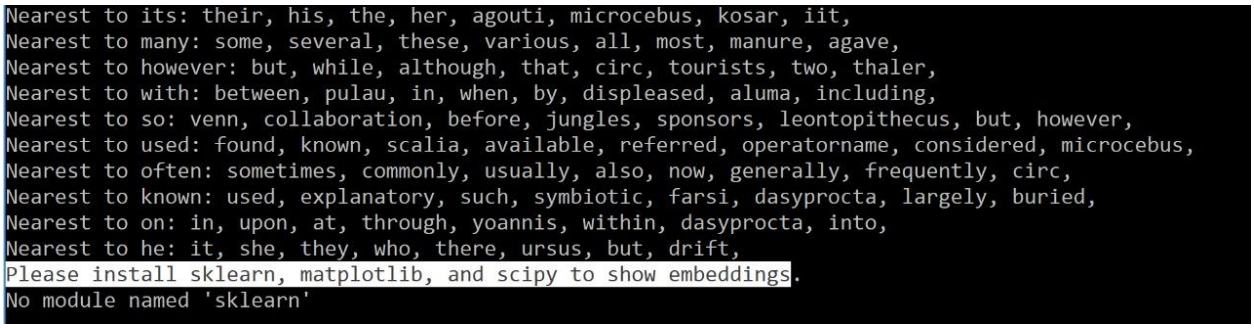
Después de **100 000 steps**, finaliza el entrenamiento. El proceso tomó aproximadamente 15 minutos.



Luego de que el entrenamiento finaliza, se genera una **imagen PNG** en la siguiente ruta: C:\Users\erosh\AppData\Local\Temp. Esta imagen es un **gráfico** de puntos en donde se pueden apreciar como imágenes que presentan **características semánticas similares**, se encuentran **cercanos** entre ellos.



Cabe mencionar que es necesario tener las librerías de '**sklearn**', '**matplotlib**' y '**scipy**' instaladas para poder generar dicho gráfico, pues si falta alguna de ellas, el programa no podrá generarlo.

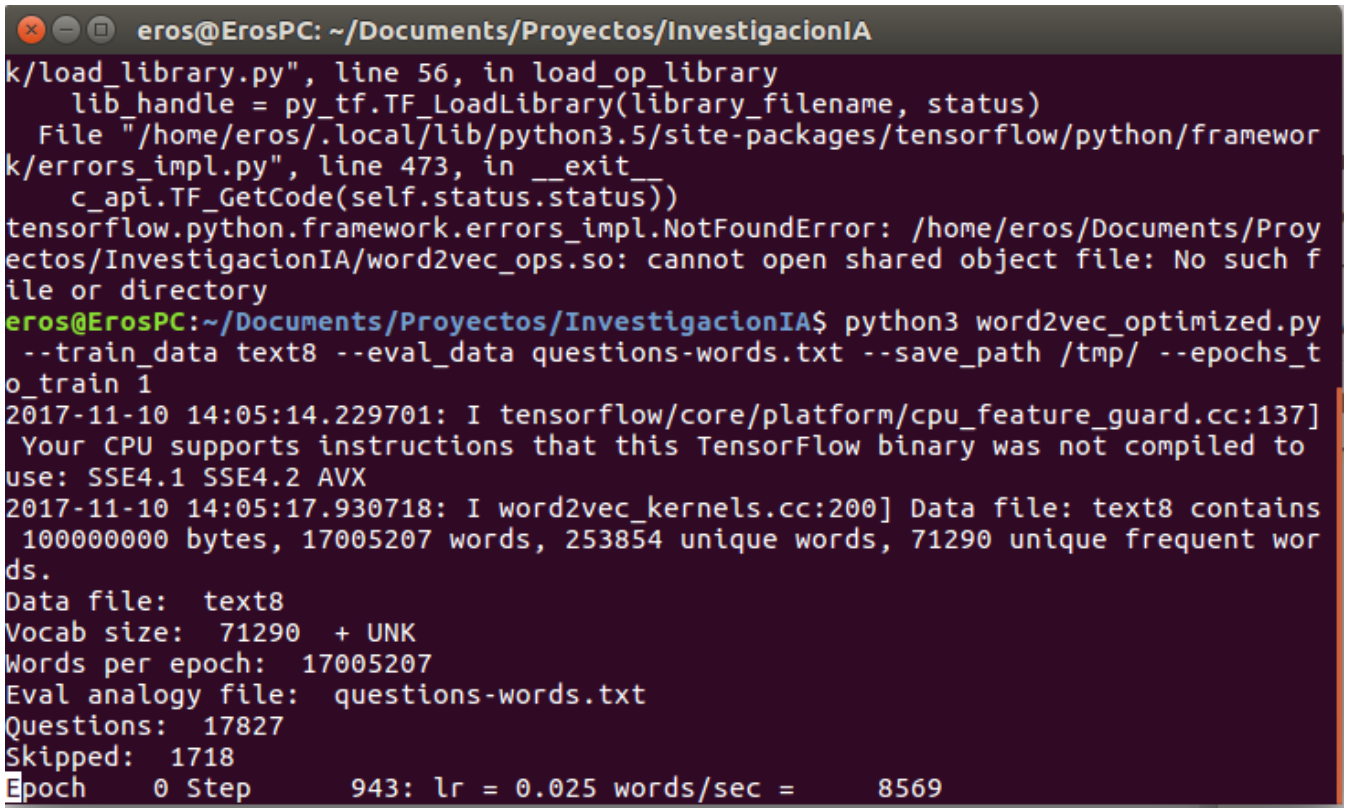


Luego de este procedimiento, se debe ejecutar el programa '**words2vec_optimized.py**' y después de un poco de investigación me di cuenta de que no puede ser ejecutado en Windows, puesto que el programa debe generar un archivo llamado '**word2vec_ops.so**' para ser utilizado más adelante, pero para generar este archivo, se deben compilar uno programas con extensión .cc, por lo que procedí a iniciar una máquina virtual con Linux, instalar Python 3, TensorFlow y sus dependencias.

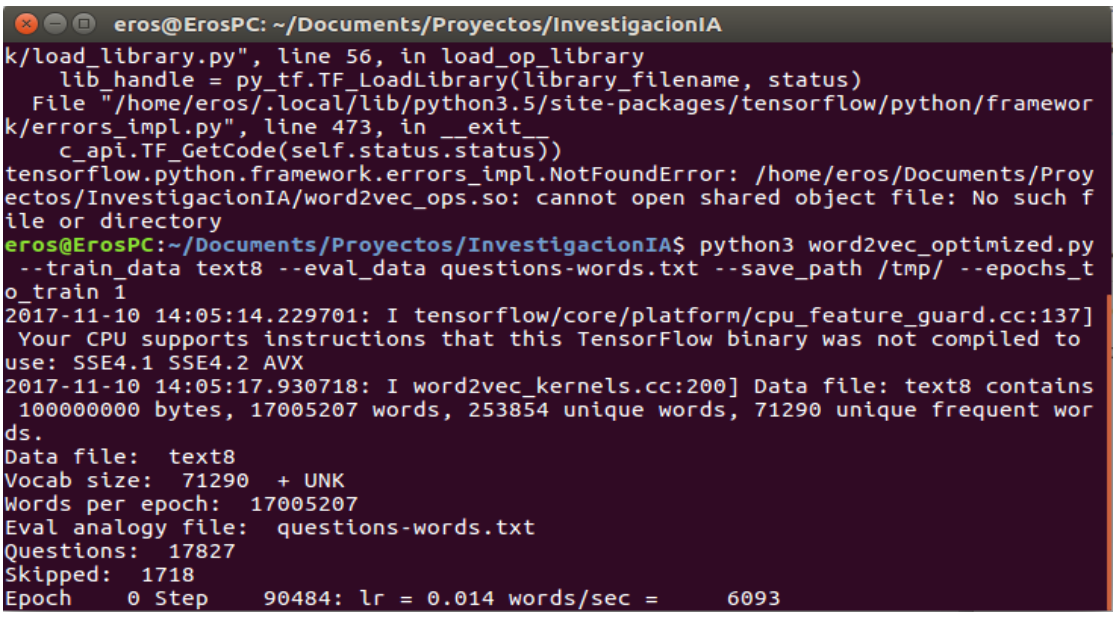
Una vez listo el ambiente de Linux, se deben obtener otros archivos necesarios para el entrenamiento de la red neuronal optimizada.

- Text8: Datos de entrenamiento para la red, se obtienen del tutorial y al haber ejecutado el primer entrenamiento.
- questions-words.txt: Datos para evaluación, se encuentran en la siguiente página: http://download.tensorflow.org/data/questions-words.txt

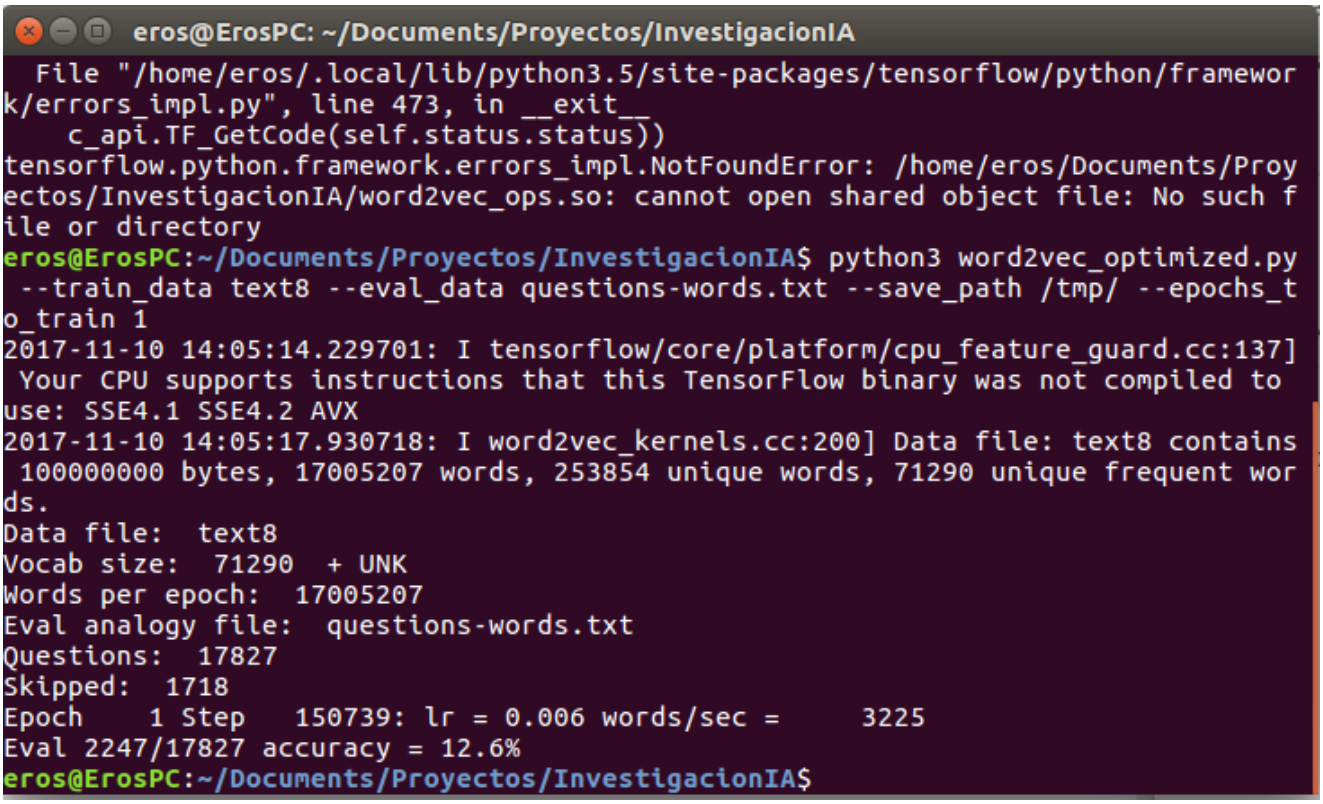
Con estos archivos, se procede a hacer el **entrenamiento** de la **red neuronal optimizada**.



El proceso toma aproximadamente **20 minutos**.



Una vez finalizado el entrenamiento con aproximadamente **150 000 steps**, se obtiene una **precisión** aproximada de **12.6%**.



-Fin de Documentación para Tutorial #5-