Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Integración de información cognitiva en modelos de lenguaje

Tesis de Licenciatura en Ciencias de la Computación

Gabriel Aimé Leclercq

Director: Bianchi, Bruno
Codirector: Travi, Fermín
Buenos Aires, 2024

INTEGRACIÓN DE INFORMACIÓN COGNITIVA EN MODELOS DE LENGUAJE

Los ojos han probado ser una ventana a una gran variedad de procesos cognitivos, por ejemplo, relacionados con la atención y la memoria. Ambas son funciones fundamentales del proceso de lectura y comprensión lectora. Por ello, el estudio de los movimientos de los ojos durante la lectura ha captado la atención de los neurolingüistas por más de un siglo, estableciendo al seguimiento ocular como una herramienta fundamental para entender el procesamiento del lenguaje en el cerebro.

En paralelo, en el campo del procesamiento del lenguaje natural, se han desarrollado herramientas para análisis del texto, principalmente a través de la tarea de predicción de palabras o de extracción de tópicos, y, más recientemente, junto a la generación de texto se han desarrollado modelos capaces de comprender textos para sostener interacciones fluidas con humanos en el desarrollo de tareas generales. El entrenamiento de estos modelos siempre ha sido a partir de textos escritos, los cuales generalmente fueron editados, y el modelo los utiliza como insumo de forma lineal y uniforme tal como le fueron presentados. Sin embargo, si bien la lectura es generalmente lineal, existen tanto variaciones en los tiempos de lectura de cada palabra como regresiones a secciones del texto anterior. Estas variaciones en la forma de la lectura están asociadas principalmente a la dificultad o ambigüedad del texto leído.

El objetivo de la tesis es cambiar el foco de los modelos de la persona que escribe a la que lee, incorporando información de los movimientos oculares durante el entrenamiento. A partir de datos experimentales recolectados de 76 personas leyendo cuentos cortos, se procedió a extraer métricas clásicas sobre seguimiento ocular durante la lectura (como, por ejemplo, la duración de la mirada sobre una palabra). Esta información se incorporó a un modelo de lenguaje basado en redes LSTM (*Long Short-Term Memory*) a través de su predicción y de alimentar al modelo con el texto en el mismo orden que fue leído. Al extraer las representaciones vectoriales de las palabras (*embeddings*), se observó que la distancia coseno entre pares de palabras correlacionaron menos con juicios de similitud humanos sobre esos mismos pares de palabras con respecto a un modelo base de referencia ($0,12$ frente a $0,19$, con una distancia intercuartil de 0,5 para ambos). No obstante, la adición de información de movimientos oculares mejoró levemente esta correlación frente a no poseer dicha información ($0,13$ vs $0,12$, con una distancia intercuartil de $0,5$ para ambos).

El reentrenamiento con texto bajo el orden leído por las personas no proporcionó mejoras frente a su equivalente con el orden original del texto, posiblemente debido a su pre entrenamiento con texto de *Wikipedia*. Por otro lado, la incorporación de información de movimientos oculares pareciera acercar levemente al espacio vectorial de las palabras a los juicios de similitud humanos. Trabajo a futuro incluye distintas maneras de incorporar esta información, así como la adición de otras métricas, y la utilización de tareas más extrínsecas para la evaluación. La presente tesis es una prueba de concepto de los avances que se podrían lograr incorporando más información de la persona que lee, y escalando a modelos más complejos.

Palabras claves: NLP, Fijaciones, LLM, Movimientos Oculares, LMM

Índice general

1. INTRODUCCIÓN

1.1. Procesamiento del Lenguaje Natural

Con el avance errático de la tecnología, el ser humano ha buscado formas no solo de solucionar problemas, sino también de resolverlos de la manera más eficiente posible utilizando el poder de cómputo de las computadoras. En particular, se ha buscado que estas pudieran reconocer y comprender el lenguaje utilizado por los seres humanos y realizar acciones en base al mismo. A partir de esta idea es donde nace un subcampo de la informática y la inteligencia artificial llamado procesamiento del lenguaje natural (NLP, por sus siglas en inglés).

El NLP permite que las computadoras y los dispositivos digitales reconozcan, comprendan y generen texto y habla al combinar la lingüística computacional, la modelación basada en reglas del lenguaje humano, junto al modelado estadístico, el aprendizaje automático y el aprendizaje profundo (Jurafsky & Martin, 2000).

La investigación en NLP ha sido uno de los actores claves que han dado lugar a la era de la IA generativa, desde las habilidades comunicativas de los grandes modelos de lenguaje (LLMs) hasta la capacidad de ciertos modelos para poder generar imágenes a partir de una descripción escrita. El NLP ya forma parte de la vida cotidiana para muchos, alimentando motores de búsqueda, impulsando chatbots para servicio al cliente con comandos de voz, sistemas de GPS operados por voz y asistentes digitales en teléfonos inteligentes.

1.1.1. Representación Vectorial de palabras

Una de las primeras cuestiones que surgen cuando hablamos sobre NLP es como representar las palabras del lenguaje para que las computadoras sean capaces de reconocerlas y utilizarlas de manera eficiente. En un principio, uno estaría tentado de sugerir la idea más simple posible, y representar las palabras como comúnmente las observamos, como secuencias de caracteres. Sin embargo, es importante tener en cuenta que existe una cantidad inmensa de información extra fuera de estas cadenas de caracteres que nos permiten entender el significado de las mismas, como puede ser el contexto o la semántica de la propia palabra. Esto genera que utilizar texto plano para identificar estas palabras y alimentar a los modelos de NLP no lleve a buen puerto.

Para suplir estas falencias se crearon los llamados vectores de palabras, o *word embeddings*, los cuales son representaciones numéricas de palabras en un espacio de alta dimensión. Cada palabra está representada por un vector y la posición de cada palabra en este espacio se determina por el contexto en el que aparece en los textos de entrenamiento. Así, palabras con significados similares tendrán vectores cercanos en este espacio (Figura 1.1). Incluso, complejizando estos *embeddings*, se puede captar información para poder reconocer de manera correcta palabras que tienen más de un significado (palabras polisémicas) (Liu et al., 2020).

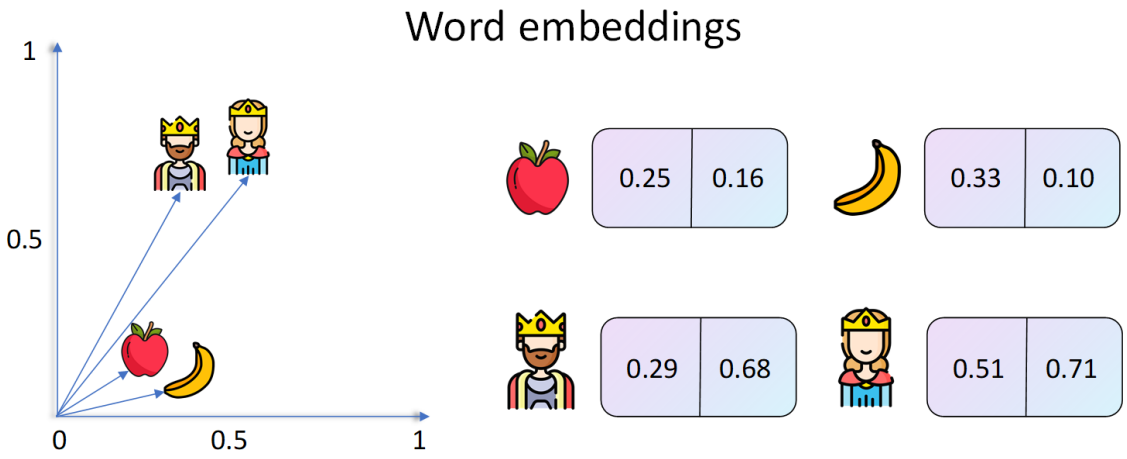


Fig. 1.1: Los vectores de palabras permiten asociar palabras mediante su cercanía dimensional, así palabras como rey o reina, o manzana y banana, se encuentran cerca en el plano.

Las representaciones vectoriales de palabras han revolucionado tareas de procesamiento de lenguaje natural como la traducción automática, el análisis de sentimientos y la recuperación de información (Chiruzzo et al., 2020). Estas representaciones permiten a las computadoras entender el significado de las palabras y sus relaciones, lo que les permite realizar tareas lingüísticas complejas.

1.1.1.1 Evaluación

Diferentes modelos de *word embeddings* generan representaciones vectoriales distintas. Sin embargo, existen algunas propiedades que todas estas representaciones deben buscar para considerarse útiles y de calidad (B. Wang et al., 2019):

- No-Fusión: Los diferentes contextos locales alrededor de una palabra deben dar lugar a propiedades específicas de la palabra, como el plural o singular, los tiempos verbales, etc. Los modelos de *word embeddings* deben ser capaces de discernir las diferencias en los contextos y codificar estos detalles en una representación significativa en el subespacio de la palabra (Yaghoobzadeh & Schütze, 2016).
- Robustez frente a la ambigüedad léxica: Deben estar representados todos los sentidos (o significados) de una palabra. Los modelos deben poder distinguir el sentido de una palabra a partir de su contexto y encontrar el embedding apropiado (Yaghoobzadeh & Schütze, 2016).
- Demostración de multifaceticidad: Los aspectos fonéticos, morfológicos, sintácticos y otras propiedades de una palabra deben contribuir a su representación final. Esto es importante ya que los modelos de palabras deberían generar representaciones significativas y quizás encontrar relaciones entre diferentes palabras. Por ejemplo, la representación de una palabra debería cambiar cuando se cambia el tiempo verbal o se añade un prefijo (Yaghoobzadeh & Schütze, 2016).
- Confiabilidad: Los resultados de un modelo de *embeddings* de palabras deben ser confiables. Esto es importante ya que los vectores de palabras se inicializan aleatoriamente durante el entrenamiento. Incluso si un modelo crea representaciones

diferentes del mismo conjunto de datos debido a la inicialización aleatoria, el rendimiento de varias representaciones debería ser consistente (Hellrich & Hahn, 2017).

- Buena geometría: Un espacio de *embeddings* debe estar correctamente distribuido. En términos generales, un conjunto pequeño de palabras frecuentes y no relacionadas debería distribuirse uniformemente en todo el espacio, mientras que un conjunto mayor de palabras raras debería agruparse alrededor de las palabras frecuentes (Gladkova & Drozd, 2016).

Por lo tanto, con estas propiedades en mente, se han definido varias formas de evaluar a los *embeddings* resultantes de un modelo, utilizando tanto distintas tareas de NLP para evaluarlos (métodos extrínsecos) como tareas totalmente independientes a estas (métodos intrínsecos).

Uno de los métodos intrínsecos más usados es la distancia entre vectores. Al interpretar la distancia entre dos vectores como medida de similitud, se evalúa su correlación con la similitud semántica percibida por los humanos. El objetivo es medir qué tan bien las representaciones vectoriales de palabras capturan la noción de similitud percibida por los humanos y validar la hipótesis distribucional, donde el significado de las palabras está relacionado con el contexto en el que ocurren (B. Wang et al., 2019).

Un evaluador comúnmente utilizado para obtener esta métrica es la similitud coseno, definida por:

$$\text{Similitud} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Donde $\mathbf{A}$ y $\mathbf{B}$ son dos vectores de palabras y $\|\mathbf{A}\|$ y $\|\mathbf{B}\|$ son las normas 2 de los mismos. Esta prueba calcula la correlación entre todas las dimensiones de los vectores, independientemente de su relevancia para un par de palabras dado o para un grupo semántico.

Debido a que sus puntuaciones están normalizadas por la longitud del vector, es robusta frente a la escala. Además, es computacionalmente barata. Por lo tanto, es fácil comparar múltiples similitudes de un modelo y se puede usar en el prototipado y desarrollo de modelos de palabras.

Ahora, para poder medir la correlación entre dos nociones de similitud, en este caso en particular entre la noción de similitud entre los *embeddings* de pares de palabras y la similitud semántica percibida por los humanos se utiliza lo que se conoce como correlación de Spearman. Esta misma examina la relación entre dos variables de una manera un poco distinta a otras correlaciones, ya que no se vale de los datos per se, sino del ranking de los mismos para ser calculada. Esto permite poder comparar dos distribuciones de datos sin importar la escala en la que se encuentren, haciendo más hincapié en su distribución dentro del ranking. En particular dadas dos distribuciones de datos, la correlación se calcula de la siguiente forma:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

donde:

- d_i es la diferencia entre los rangos de las variables X y Y para el i-ésimo par de observaciones.
- n es el número total de observaciones.

Esta correlación ha sido utilizada en investigaciones para poder comparar modelos de *embeddings* con resultados de tareas de juicios de valor humanos, pudiendo así constatar que los *embeddings* son capaces de captar similitudes entre pares de palabras de manera similar a los humanos (Chandrasekaran & Mago, 2021).

1.1.1.2 Word2Vec

Para obtener estas representaciones vectoriales, se han desarrollado un gran número de técnicas y modelos, las cuales hablaremos de algunas a continuación.

Una de las técnicas más reconocidas para la obtención de *word embeddings* es la de Word2Vec (Mikolov, Chen et al., 2013). Esta permite obtener vectores de palabras de gran calidad capaces de ser entrenados con textos de millones de palabras. Además, los vectores que representan palabras con significados similares se encuentran posicionados cerca unos de otros en este espacio de alta dimensión.

Técnicamente hablando, esta arquitectura está compuesta por una red neuronal unicapa que procesa texto al recibir lotes (*batches*; es decir, subconjuntos de los datos) del mismo sin procesar, generando un espacio vectorial de varias centenas de dimensiones. Cada palabra única en los datos recibe un vector correspondiente en el espacio. La posición de estos vectores en el espacio está determinada por los significados de las palabras y su proximidad a otras palabras, en el contexto en el que los *embeddings* fueron entrenados.

Esta técnica para obtener *word embeddings* se puede implementar utilizando dos diseños arquitecturales: el modelo de *Continuous Bag of Words* (CBOW) y el modelo *Skip-Gram*. Ambos tienen como objetivo reducir la dimensionalidad de los datos y crear vectores de palabras densos, pero abordan el problema de manera diferente.

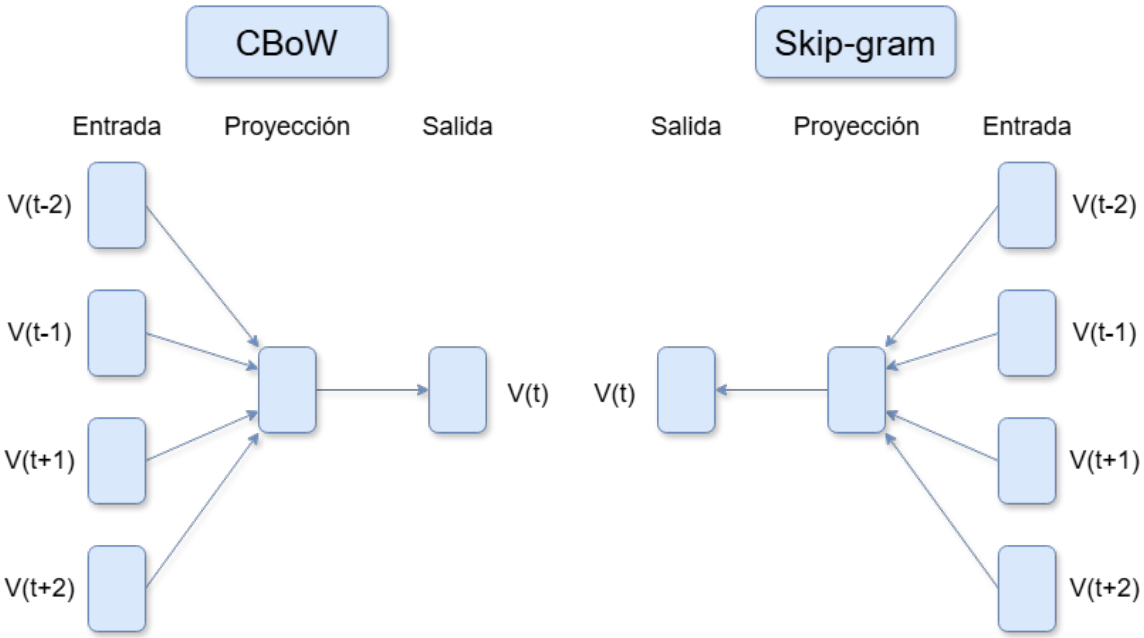


Fig. 1.2: Arquitecturas de Word2Vec. La arquitectura CBOW predice la palabra actual en función del contexto, mientras que Skip-gram predice las palabras circundantes dada la palabra actual.

CBOW

El modelo CBOW predice la palabra objetivo a partir de las palabras de contexto circundantes (Figura 1.2). Dicho de otro modo, utiliza las palabras del entorno para predecir la palabra en particular. Este modelo toma todas las palabras de contexto, las procesa y usa el vector resultante para predecir la palabra objetivo. Este modelo resulta beneficioso para tareas de índole más sintáctica, acompañado por un costo de computo menor (Mikolov, Chen et al., 2013).

Skip-gram

El modelo Skip-Gram predice las palabras de contexto circundante a partir de una palabra objetivo (Figura 1.2). En resumen, utiliza una sola palabra para predecir su contexto circundante. Este modelo funciona bien para tareas más semánticas (Mikolov, Chen et al., 2013). Sin embargo, es computacionalmente más costoso que el modelo CBOW debido a su tarea de predecir múltiples palabras de contexto.

Entrenamiento

Cada palabra en el corpus se representa inicialmente como un vector de alta dimensión, previamente fijado con valores aleatorios. Estos vectores sirven como punto de partida para el proceso de entrenamiento. A medida que avanza el entrenamiento, estos vectores se actualizan en función de la función objetivo del modelo. En consecuencia, estos se posicionan más cerca entre sí en el espacio vectorial a los vectores de palabras que aparecen en contextos similares.

Otro aspecto crítico en el entrenamiento de Word2Vec es la elección del tamaño de la ventana. La ventana actúa como una ventana deslizante que pasa sobre el texto y determina qué palabras se analizan en el contexto de una palabra objetivo. Las palabras dentro de la ventana se consideran parte del contexto, mientras que las que están fuera se ignoran. La elección del tamaño de la ventana genera efectos distintos en los vectores. Un tamaño de ventana más pequeño permite que el modelo realice un aprendizaje más enfocado en la palabra objetivo, mientras que un tamaño de ventana más grande ayuda al modelo a comprender el contexto en el que fue utilizada la palabra objetivo mucho más en detalle (Levy & Goldberg, 2014). Sin embargo, un tamaño de ventana más grande aumenta la complejidad computacional del modelo, ya que se deben procesar más palabras de contexto para cada palabra objetivo.

Además, otra de las características que define a Word2Vec es la utilización de *Negative Sampling* (Mikolov, Sutskever et al., 2013), el cual aborda el problema de la eficiencia computacional actualizando solo un pequeño porcentaje de los pesos del modelo en cada paso en lugar de todos ellos. Esto se logra seleccionando un pequeño número de palabras "negativas" (palabras que no están en el contexto) para actualizarlas en cada palabra objetivo.

Por último, en el modelo se realiza un submuestreo de palabras frecuentes, el cual ayuda a mejorar la calidad de los vectores de palabras (Mikolov, Sutskever et al., 2013). La idea básica es reducir el impacto de las palabras de alta frecuencia en el proceso de entrenamiento, ya que a menudo contienen información menos significativa en comparación con las palabras no frecuentes.

Al descartar aleatoriamente algunas instancias de palabras frecuentes, el modelo se ve

obligado a centrarse más en las palabras raras, lo que lleva a vectores de palabras más equilibrados y con información más relevante.

1.1.2. Modelos de Lenguaje

A los *embeddings* no solo los podemos encontrar como resultado de arquitecturas como Word2Vec, sino también como partes clave de otros modelos cuyo objetivo es distinto al de generar una serie de vectores de palabras; tal es el caso de algunos modelos de lenguaje, como el de la arquitectura AWD-LSTM, que será introducida en secciones siguientes.

Yendo más en detalle, un modelo de lenguaje es un tipo de modelo de aprendizaje automático entrenado para establecer una distribución de probabilidad sobre las palabras. Básicamente, un modelo intenta predecir la siguiente palabra más adecuada para llenar un espacio en blanco en una oración o frase, basándose en el contexto del texto dado (Figura 1.3).

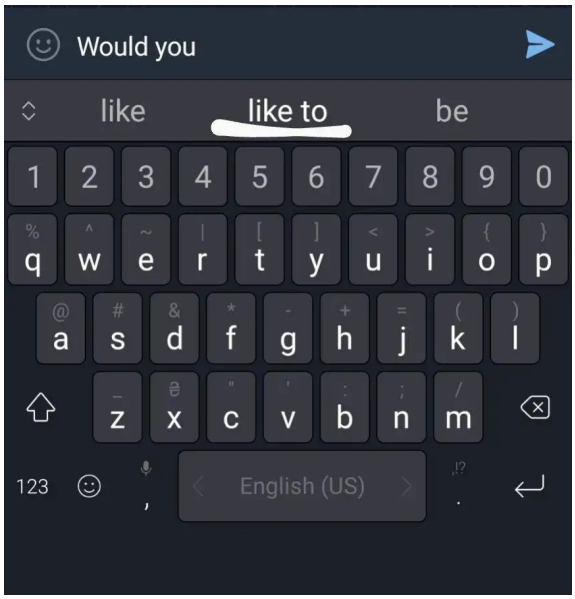


Fig. 1.3: Los modelos de lenguaje se utilizan en una variedad de tareas de NLP, como el reconocimiento de voz, el autocompletado de textos y la creación de resúmenes de textos.

Son un componente fundamental en el ámbito del NLP porque permiten que las máquinas comprendan, generen y analicen el lenguaje humano. Se entrenan principalmente utilizando grandes corpus de texto. Los modelos luego utilizan los patrones que aprenden de estos datos de entrenamiento para predecir la siguiente palabra en una oración o generar nuevo texto que sea gramaticalmente correcto y semánticamente coherente.

Existen diferentes tipos de Modelos de Lenguaje, los cuales se pueden clasificar en dos categorías: **modelos estadísticos** y modelos basados en **redes neuronales profundas**.

Por un lado, los modelos estadísticos de lenguaje son un tipo de modelo que utilizan patrones estadísticos en los datos para hacer predicciones sobre la probabilidad de secuencias específicas de palabras. Por ejemplo, un enfoque básico para construir un modelo de lenguaje probabilístico es la utilización de n-gramas, es decir conjuntos de n palabras consecutivas en un texto.

Por otro lado, los modelos de lenguaje neuronales, como su nombre indica, utilizan redes neuronales para predecir la probabilidad de una secuencia de palabras. Estos modelos se entrenan con un gran corpus de datos de texto y cumplen el objetivo de predecir palabras dado un contexto al minimizar una función de costo (Figura 1.4).

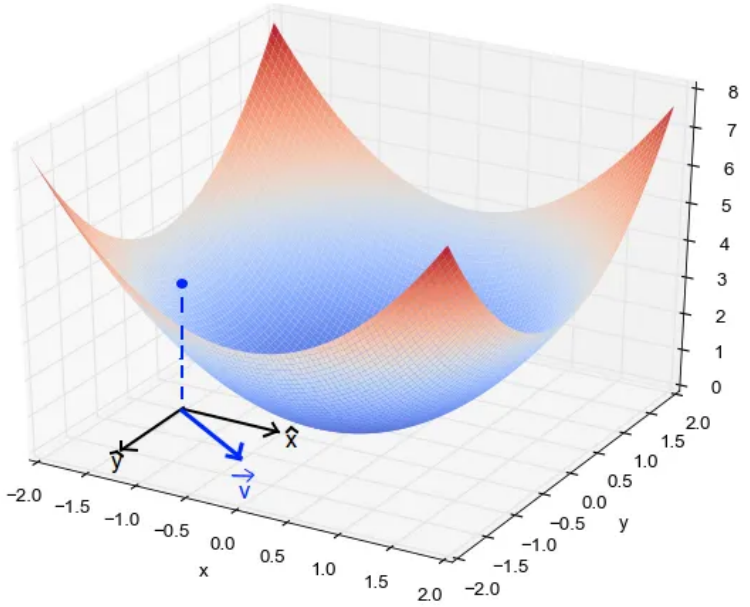


Fig. 1.4: La idea general de los modelos basados en redes neuronales es minimizar una función de costo generada a partir de los pesos de la red. Para esto se calcula el gradiente de la función en un punto, o sea la pendiente de la tangente a la función de coste. Este gradiente puede ser calculado de a batches o de a una entrada a la vez, o sea de manera estocástica.

Una vez entrenados estos modelos, los mismos permiten captar mejor dependencias de largo alcance entre palabras que con los modelos estadísticos tradicionales (Z. Wang, 2018). Por ejemplo, al utilizar un modelo estadístico basado en n-gramas, el contexto que presenta el modelo es limitado, mientras que los modelos basados en redes neuronales presentan diversidad de estrategias para poder recordar contextos lejanos a lo largo del entrenamiento. Esta es la razón por la cual nos decantamos por ellas. Una de las arquitecturas más reconocidas que utiliza estas estrategias son las Redes neuronales recurrentes (RNNs).

1.1.2.1 Redes neuronales recurrentes

Una red neuronal recurrente, o RNN, es una red neuronal profunda entrenada con datos secuenciales o de series temporales para crear un modelo de aprendizaje automático que pueda hacer predicciones o conclusiones basadas en estas mismas.

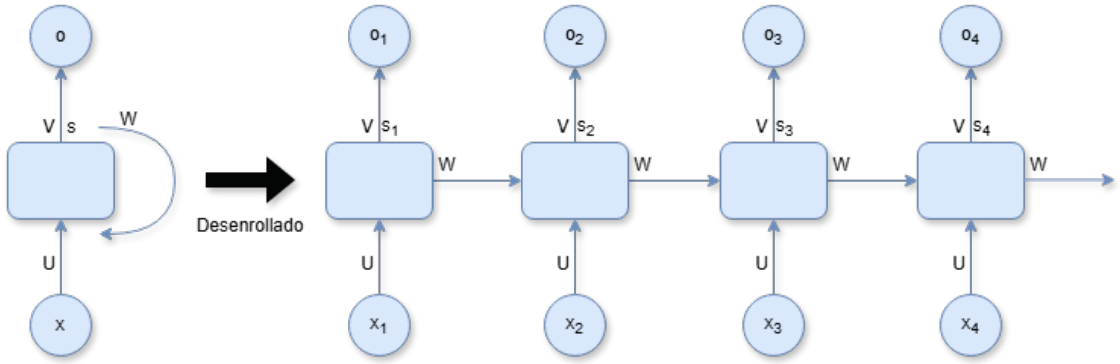


Fig. 1.5: Diagrama de una Red neuronal recurrente mostrando su versión compacta a la izquierda y su versión desenrollada a la derecha, ilustrando cómo la red se expande a lo largo del tiempo para procesar secuencias de datos.

Se distinguen por su "memoria", ya que toman información de entradas anteriores para influir en la entrada y salida actuales. Mientras que las redes neuronales profundas tradicionales suponen que las entradas y salidas son independientes entre sí, la salida de las redes neuronales recurrentes depende de los elementos anteriores dentro de la secuencia (Figura 1.5). Incluso existen tipos de RNNs cuya salida actual no solo se ve influenciada por los resultados anteriores sino también por los resultados futuros.

Otra característica distintiva de las redes recurrentes es que comparten parámetros en cada capa de la red. Mientras que las redes tradicionales tienen diferentes pesos en cada nodo, las redes neuronales recurrentes comparten pesos en cada capa de la misma. Dichos pesos se ajustan a través de los procesos de *backpropagation* y descenso por el gradiente para facilitar el aprendizaje.

Las redes neuronales recurrentes utilizan algoritmos de *backpropagation* a través del tiempo (BPTT, de sus siglas en inglés) para determinar los gradientes, lo cual es ligeramente diferente del algoritmo tradicional, ya que es específico para datos secuenciales. Los principios de BPTT son los mismos que los de *backpropagation* tradicional, donde el modelo se entrena calculando errores desde su capa de salida hacia la capa de entrada. Estos cálculos permiten ajustar y adaptar los parámetros del modelo adecuadamente. La diferencia con el enfoque tradicional es que BPTT suma los errores en cada paso temporal, mientras que las redes normales no necesitan hacerlo, ya que no comparten parámetros entre las capas.

Este tipo de redes, sin embargo, se ven afectadas por diversas cuestiones. Una de ellas, aunque no de manera tan tajante como puede ser en el caso de los modelos basados en n-gramas, es la incapacidad de recordar dependencias a largo plazo; es decir, si el estado anterior que influye en la predicción actual no está en el pasado reciente, el modelo RNN puede no ser capaz de predecir correctamente el estado actual.

Por ejemplo, supongamos que queremos predecir las palabras en la siguiente oración:

...Alicia es alérgica a los frutos secos. Ella no puede comer nueces...

El contexto de una alergia a los frutos secos nos ayuda a anticipar que el alimento que no se puede comer contiene frutos secos. Sin embargo, si ese contexto se mencionara varias oraciones antes, sería difícil para la RNN inferir esa información, debido a su falta de capacidad para asociar relaciones entre oraciones muy distantes.

A su vez, se considera que las RNNs, las cuales en su mayoría dividen la información del dataset en batches de tamaño fijo, presentan algunos inconvenientes a la hora de manejar la información de entrenamiento de manera eficiente. Esto se debe a que al utilizar un tamaño fijo para el batch, independientemente en la época que esté del entrenamiento, siempre van a existir un grupo de elementos del batch los cuales no van a poder aprovecharse de la capacidad recurrente de la red. Por ejemplo, volviendo al caso de la oración anterior, si entrenáramos nuestro modelo con batches de tamaño 4, nos pasaría que palabras como 'Alicia', 'los' y 'no' no se aprovecharían de la información previa a ellas, a pesar de ser esto una pieza clave de las RNN (Merity et al., 2017).

Al mismo tiempo, durante este proceso, las RNN tienden a enfrentar otros dos problemas, conocidos como *exploding gradients* y *vanishing gradients*. Estos problemas se definen por el tamaño del gradiente. Cuando el gradiente es demasiado pequeño, este mismo sigue disminuyendo hasta que los parámetros de peso se vuelven insignificantes, y en ese punto, el algoritmo deja de aprender. Por otro lado, los gradientes que explotan ocurren cuando el gradiente es demasiado grande, creando un modelo inestable. En este caso, los pesos del modelo crecerán demasiado y eventualmente se llegará a problemas de convergencia.

Una solución utilizada para mitigar estos problemas, especialmente el de los gradientes altos, es la implementación del concepto de *gradient clipping*, el cual se basa en la idea de establecer un límite, el cual si es pasado por el gradiente durante el entrenamiento se procede a reducir proporcionalmente el mismo para evitar que el vector se vuelva demasiado grande (Pascanu et al., 2013). A su vez también han surgido otras arquitecturas de modelos de lenguaje que buscan afrontar estos problemas.

1.1.2.2 LSTM

Las redes *Long Short-Term Memory* (LSTM) son una serie de arquitecturas populares de RNNs surgidas con el objetivo de solucionar el problema del *exploding gradients* y *vanishing gradients*. Además, con esta arquitectura se busca resolver otros inconvenientes como el de la incapacidad de asociar dependencias a largo plazo (Hochreiter & Schmidhuber, 1997).

Para solucionar esto, las redes LSTM introducen un nuevo tipo de células de memoria, distintas a las presentes en las RNNs y capaces de retener información a lo largo de secuencias extensas. Cada célula de memoria presenta tres componentes principales: una compuerta de entrada, una de olvido y una de salida (Figura 1.6). Estas puertas ayudan a regular el flujo de información dentro y fuera de la célula de memoria.

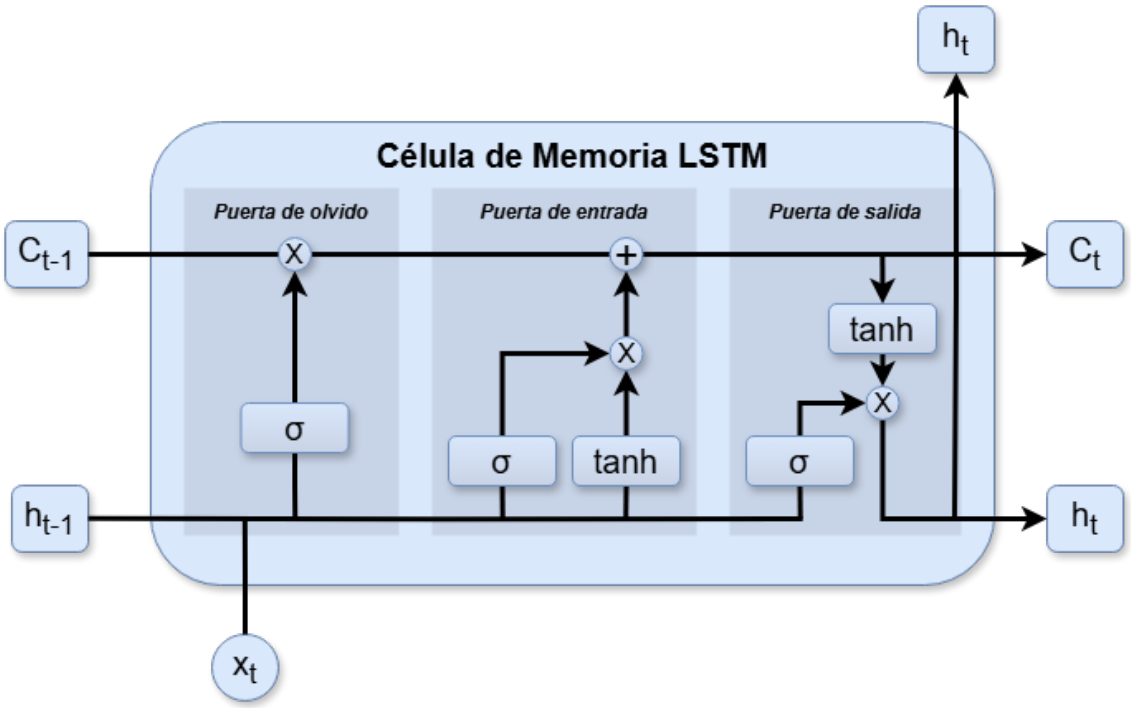


Fig. 1.6: Diagrama de una LSTM junto a sus puertas. Las LSTM utilizan estas para regular el flujo de información, lo que les permite aprender dependencias a largo plazo en los datos, haciéndolas particularmente efectivas para tareas que involucran datos secuenciales como la predicción de series temporales, el procesamiento del lenguaje natural, el reconocimiento de voz, y más.

- **La compuerta de entrada** determina cuánta de la nueva información debe almacenarse en la célula de memoria. Toma la entrada actual y el estado oculto anterior como entradas, y genera un valor entre 0 y 1 para cada elemento de la célula de memoria.
- **La compuerta de olvido** decide qué información debe descartarse de la célula de memoria. Al igual que la compuerta de entrada, toma la entrada actual y el estado oculto anterior y emite un valor entre 0 y 1. Un valor de 0 significa que la información se ignora, mientras que un valor de 1 indica que se retiene.
- **La compuerta de salida** controla cuánta de la información de la célula de memoria debe usarse para calcular el estado oculto. También toma la entrada actual y el estado oculto anterior como entradas y emite un valor entre 0 y 1 para cada elemento de la célula de memoria.

Al controlar y memorizar información a lo largo de secuencias largas, las LSTM pueden mitigar los problemas de *exploding gradients* y *vanishing gradients*, permitiendo un entrenamiento más efectivo y una mejor captura de patrones a largo plazo en los datos secuenciales.

Durante los años, muchas arquitecturas han florecido utilizando a las LSTM como base, ya sea como motivación o literalmente. Por ejemplo, una de ellas son las GRU, o *Gated Recurrent Units* de sus siglas en inglés, las cuales tenían como objetivo presentar una

alternativa que pudiera solucionar los problemas resueltos por las LSTM, con la ventaja de tener un diseño más simple y una complejidad computacional mucho menor. Esto lo lograban, entre otras cosas, condensando el trabajo de las compuertas de entrada y olvido de la LSTM en una sola compuerta llamada compuerta de actualización (Chung et al., 2014). Sin embargo, si dejamos de lado la rapidez para entrenar los modelos, las GRU generan peores resultados que las LSTMs fuera de datasets pequeños (Yang et al., 2020).

1.1.2.3 AWD-LSTM

Otro ejemplo de modelo de lenguaje basado en LSTMs es la red *ASGD Weight-Dropped LSTM* (AWD-LSTM) (Figura 1.7). A diferencia de la arquitectura LSTM básica, la AWD-LSTM implementa varias técnicas de regularización a lo largo de su red neuronal, con el objetivo de mejorar la capacidad predictiva del modelo, especialmente en la capacidad del modelo para poder generalizar. Además, estas técnicas mejoran la estabilidad del modelo, ya que reducen la probabilidad de sobreajuste del mismo.

Muchas de estas técnicas de regularización están asociadas a la aplicación de *dropout* en porcentajes distintos en distintas partes de la red neuronal. Estas partes en concreto son:

- En la capa de *embeddings*, los pesos de algunas palabras dentro del vocabulario de la capa son desactivados.
- Luego de la capa de *embeddings*, una vez obtenidos los *embeddings* de todas las palabras del texto de entrada, ciertos pesos de los vectores son desactivados de manera aleatoria.
- Previo al ingresar a cada celda LSTM.
- Dentro de cada una de las celdas LSTM.
- Al finalizar la activación de la última celda.

Adicionalmente, la arquitectura introduce un algoritmo de optimización nuevo, llamado NT-ASGD (de sus siglas *Non-monotonically Triggered Averaged Stochastic Gradient Descent*). En resumen, se implementa un algoritmo parecido al de descenso por el gradiente estocástico, con la diferencia de que los pesos de la red no se ven afectados solamente por el gradiente actual, sino por el promedio de los últimos n gradientes, donde n es un hiperparámetro. Además, este promedio no se realiza a partir del primer batch de entrada, sino que el algoritmo actúa como un descenso por el gradiente estocástico normal hasta que el modelo empeora o se estanca durante varias entradas seguidas, a partir del cual se empieza a calcular el gradiente final en base al promedio de los anteriores hasta finalizar el entrenamiento.

Asimismo, anteriormente habíamos mencionado que las RNNs podían llegar a no ser 100 % eficientes en el uso del dataset de entrenamiento, debido a que, en el caso de las tareas de predicción de texto, existen algunas palabras que no aprovechan la capacidad recurrente de la red si se utiliza un tamaño de batch fijo durante el entrenamiento. Este problema es atacado por la AWD-LSTM variando el tamaño del batch que se utiliza en cada época. En particular esta variación la realiza partiendo de un tamaño base, el cual llamaremos *seq*. En un primer paso, se elige como valor intermedio con probabilidad p a este valor *seq* y con probabilidad $1 - p$ a $\frac{seq}{2}$, con p un valor muy cercano a 1. Esto se

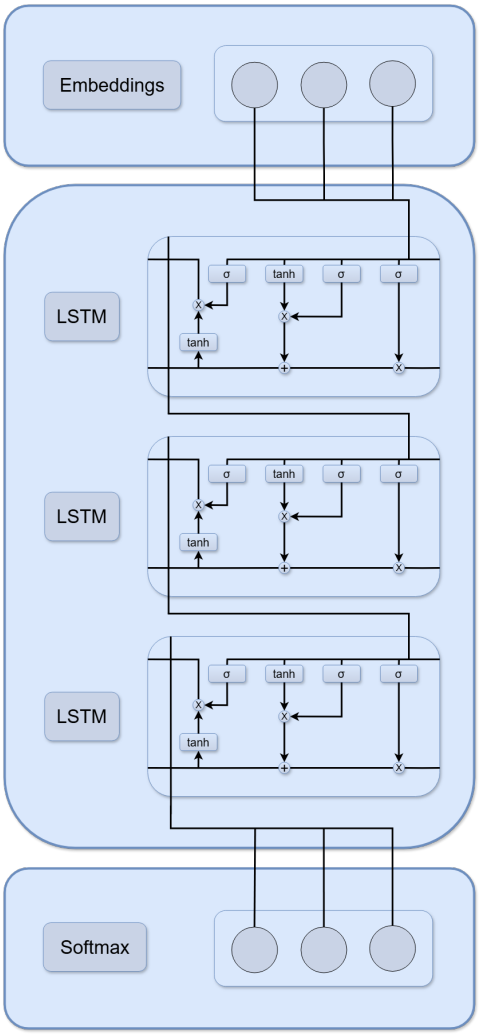


Fig. 1.7: Estructura clásica de un modelo basado en AWD-LSTM. El mismo presenta una capa de *embeddings*, por el cual la entrada pasará y verá sus palabras transformadas en vectores de palabras. Unas 3 células LSTM y una capa softmax la cual permite obtener la distribución de probabilidades para la siguiente palabra del texto dado el vocabulario.

hace para aumentar el rango de valores usados durante el entrenamiento. Para finalizar, se selecciona el tamaño de batch final a partir de una distribución normal con media en ese valor intermedio y un desvío estándar. De esta manera, se aprovecha mucho más la capacidad recurrente de la AWD-LSTM.

Por último, el *learning rate* del modelo es re-escalado dependiendo del tamaño de batch resultante, ya que se ha encontrado una tendencia a favorecer a las oraciones más cortas durante el entrenamiento si se utiliza un tamaño de batch variable junto a un *learning rate* fijo (Merity et al., 2017).

Debido a estas mejoras introducidas por esta arquitectura y porque fue una de las más populares previo al surgimiento de otros modelos como los *transformers*, decidimos tomar a la AWD-LSTM como nuestro eje principal en ese trabajo.

1.1.2.4 Evaluación

Una vez se tiene elegido el tipo de arquitectura de modelos de lenguaje que se utilizara para llevar a cabo nuestro propósito, es lógico preguntarse de qué manera podemos evaluar nuestros modelos para ver si se comportan de manera satisfactoria y son capaces de comprender el lenguaje humano. Para esto, fuera de las típicas métricas utilizadas para evaluar modelos de aprendizaje automático como el *Accuracy* o el *F1 Score*, existen otras, como la *Perplexity*, la cual es considerada como una medida utilizada para cuantificar la incertidumbre asociada con una distribución de probabilidad. Esta métrica a lo largo de los años se ha utilizado como una manera de evaluar distintos modelos de lenguaje (Merity et al., 2017).

Yendo más en detalle, esta métrica se define como la exponenciación de la entropía de una distribución de probabilidad. Se puede expresar como:

$$PP(W) = 2^{H(W)}$$

donde el $H(W)$ es la entropía cruzada del modelo con respecto a la secuencia de palabras W. Un nivel de *perplexity* más bajo indica un mejor modelo predictivo, ya que implica que el modelo tiene más confianza en sus predicciones. Por el contrario, un nivel de *perplexity* más alto sugiere una mayor incertidumbre y predicciones menos efectivas. Además, al basarse en la entropía cruzada, la cual es una métrica normalizada en la longitud de la secuencia, permite comparar modelos con distintos tamaños de corpus. Sin embargo, cabe aclarar que la *perplexity* presenta ciertas limitaciones. Por ejemplo, es sensible al vocabulario del modelo, obligando a que la misma sea comparable solo entre modelos que presentan el mismo vocabulario. Por otro lado, es una métrica de evaluacion intrínseca, por lo que a pesar de ser considerada el estándar para comparar modelos de NLP, una mejoría en la misma no necesariamente implica una mejora en la tarea que se desea evaluar (Jurafsky & Martin, 2000).

1.2. Movimientos oculares durante la lectura

La lectura se puede considerar como un desarrollo relativamente reciente en la historia de la humanidad, existiendo sólo desde hace unos pocos miles de años (Immordino-Yang & Deacon, 2007). Sin embargo, se ha convertido en una habilidad esencial en la vida moderna, que se desarrolla a lo largo de años de exposición, instrucción formal y práctica. Una buena habilidad de lectura es fundamental para el logro académico (para una discusión, ver Renandya (2007)), y para los estudiantes de un segundo idioma, la lectura es una compuerta de entrada para aprender nuevo vocabulario, más lenguaje coloquial y nuevas construcciones gramaticales (Wilkinson, 2012).

Sabemos que para los lectores el objetivo principal es identificar palabras, comprender su significado e integrarlas en su comprensión progresiva de una oración y/o de un discurso más amplio. Sin embargo, ¿Qué ocurre exactamente cuando leemos? ¿Cómo se mueven nuestros ojos?

Tal y como menciona Brown (1895), nuestros ojos a la hora de leer se asemejan al movimiento del segundero dentro de un reloj, se realiza un tirón y una pequeña pausa, luego otro tirón, y así sucesivamente; solo que nuestros ojos no son tan regulares, los tirones a veces tienen una amplitud mayor o menor, y las pausas varían en duración, aunque, a menos que hagamos un esfuerzo, siempre son breves. Durante los tirones prácticamente

no vemos nada, por lo que no tenemos ante nosotros un panorama en movimiento, sino una serie de imágenes fijas que se suceden rápidamente.

Estos "tirones" del ojo — en otras palabras, sus movimientos — son lo que comúnmente denominamos sacadas. El intervalo entre los movimientos de los ojos, cuando estos se detienen, se llama fijación. Ambos son un tipo de respuesta fisiológica automática, lo que significa que no están bajo nuestro control consciente (Rayner et al., 2012).Cualquier secuencia completa de estos dos se denomina como un *scanpath*. Sin embargo, al leer, las sacadas no siempre mueven el ojo hacia adelante en el texto. Aproximadamente entre el 10 y el 15 por ciento de las veces, los lectores mueven los ojos hacia atrás (regresan) a secciones del texto previamente vistas. Estos movimientos hacia atrás se conocen como regresiones. Las regresiones pueden ser cortas o largas. Las regresiones cortas suelen deberse a un exceso de movimiento sobre el objetivo. Por otro lado, las regresiones largas son atribuidas comúnmente a la dificultad del texto leído, que puede deberse a diversos factores.

Básicamente, cuando leemos o miramos una escena o imagen, nuestros ojos se detienen para procesar la información en esa ubicación y luego se mueven a otro punto donde hay otra información disponible. Durante las fijaciones, el sistema cognitivo percibe y procesa la información visual, además de planificar cuándo y hasta dónde mover los ojos a continuación. En la mayoría de las circunstancias normales, durante un movimiento sacádico, los ojos se mueven tan rápido que no obtenemos nueva información visual (Rayner, 2009). Sin embargo, mientras no se codifica nueva información visual durante los movimientos sacádicos, el procesamiento de la información ya percibida previamente continua (Irwin, 1998; Irwin y Carlson-Radvansky, 1996).

Entonces, ¿de qué manera podría interesar el rastreo y medición de los movimientos oculares de las personas dentro de la lectura? Las fijaciones, los movimientos sacádicos y las regresiones ocurren generalmente de forma "automática", sin que seamos conscientes de ello. De esta forma, el seguimiento de los movimientos oculares nos ofrece una ventana a un comportamiento en gran medida inconsciente[1]. Además, se ha demostrado que en tareas de procesamiento complejo, como es el caso de la lectura, la ubicación de los ojos proporciona un índice de atención (Rayner, 2009). Esto significa que nuestros ojos indican en qué estamos prestando atención y cuánta energía cognitiva se está invirtiendo para procesar la información en el punto de fijación. Así, la dificultad y la complejidad de lo que miran los ojos influyen en las fijaciones y los movimientos sacádicos (Castelhano & Rayner, 2008). Cuando el estímulo es más difícil, aumentan las duraciones de fijación y las regresiones, mientras que el tamaño de los movimientos sacádicos disminuye. Esto significa que, en la lectura, los textos más difíciles provocan fijaciones y regresiones más frecuentes y prolongadas, mientras que los sacádicos se acortan. Al observar escenas o imágenes más densas o complejas, las fijaciones también se alargan y los movimientos sacádicos se acortan.

Estas, en definitiva, son algunas de las razones por las cuales el análisis de movimientos oculares se ha vuelto una de las herramientas más poderosas para estudiar la manera en que la información visual es procesada por la mente humana y una de las herramientas más estándares a la hora de realizar estudios sobre lectura en los campos de la psicolingüística, la psicología cognitiva y la lingüística aplicada (Rayner, 2009, p. 1474). Sin embargo, cabe aclarar que todas estas ventajas brindadas por el análisis de los movimientos oculares se apoyan en gran medida en lo que se conoce como la hipótesis ojo-mente, la cual afirma que hay una estrecha relación entre los movimientos del ojo y el procesamiento cognitivo dentro

[1] donde como inconciente no nos referimos al inconciente *freudiano*

de nuestro cerebro a la hora de leer (Just & Carpenter, 1980). A su vez, esta hipótesis se basa en dos ideas subyacentes:

1. Primero, está la suposición de que lo que se está fijando es lo que se está considerando. Esto significa que cuando los ojos se fijan en una palabra "x", el cerebro está trabajando para descifrar y entender esta palabra y no otra palabra "z" que apareció tres palabras antes. Es decir, los lectores intentan interpretar las palabras a medida que las encuentran. No obstante, esta suposición termina resultando ser, aunque acertada, algo simplista (Ehrlich & Rayner, 1983). Por ejemplo, en una oración como "Maria vendió su caballo a José porque ella decidió dejar de montar", cuando los ojos se posan en la palabra "ella", para interpretarla, el cerebro necesita considerar entidades previamente encontradas que podrían ser referentes potenciales del pronombre. Así, cuando los ojos se detienen en "ella", la mente está trabajando en esta palabra; sin embargo, también considera elementos previos de la oración que podrían ser referentes potenciales (por ejemplo, "Maria").

2. La segunda parte, estipula que la cantidad de tiempo dedicado a fijar un elemento o región refleja el esfuerzo cognitivo necesario para procesarlo. Esto significa que fijaciones más largas y frecuentes indican un mayor esfuerzo de procesamiento, y las fijaciones más breves y/o los saltos indican un menor esfuerzo de procesamiento. Además, es importante aclarar que estos tiempos deben tomarse en términos relativos: una mayor o menor duración y un mayor o menor esfuerzo de procesamiento deben compararse con algo. En general, estas métricas suelen asociarse directamente a una palabra a la vez dentro de un texto, pero en el caso de que se quiera analizarlas en contextos mas amplios, vease n-gramas u oraciones enteras, estas mismas se definen sobre una región de interés (conocida como ROI, de sus siglas en inglés)

1.2.1. Métricas asociadas a movimientos oculares

Habiendo previamente mencionado todas las ventajas de la utilización de movimientos oculares para obtener información sobre el comportamiento inconsciente de las personas a la hora de leer, parecería lógico preguntarse de qué manera se puede extraer esta información.

A partir de esto, la tecnología de seguimiento ocular nos permite indicar, entre otras cosas, dónde se posan los ojos de las personas, cuántas veces se posan en esa posición o región (conteo de fijaciones) y cuánto dura cada fijación (duración de fijación), además de medir la duración y la longitud de los movimientos sacádicos. Estas medidas se toman en experimentos utilizando el equipamiento adecuado, dividiendo los mismos en distintas pruebas, donde el equipamiento va recolectando información sobre los movimientos oculares de un sujeto a medida que este va realizando la lectura de un corpus de texto.

Ahora, dado que las fijaciones (y específicamente la duración de las mismas) son más sensibles a los factores lingüísticos que los movimientos sacádicos (Staub & Rayner, 2007), estas tienden a ser las métricas que más nos interesan en estudios basados en textos. Estas métricas se clasifican en "tempranas", "intermedias" o "tardías" y se entiende que reflejan diferentes etapas del procesamiento de la lectura. Las medidas tempranas se consideran principalmente como un reflejo de procesos automáticos de reconocimiento de palabras y acceso léxico, mientras que las medidas tardías tienden a reflejar procesos más conscientes, controlados y estratégicos (Altarriba et al., 1996; Inhoff, 1984; Paterson et al., 1999; Staub y Rayner, 2007).

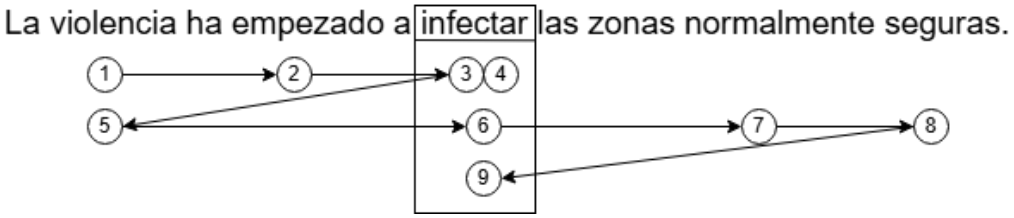


Fig. 1.8: Ejemplo de scanpath. Cada círculo debajo de una palabra indica una fijación en el orden el cual fue hecha mientras que las flechas indican las sacadas realizadas. En este caso la palabra crítica tenida en cuenta es "infectar".

1.2.1.1 Métricas tempranas

Cada una de estas métricas puede considerarse como un índice de acceso léxico, o de qué tan fácilmente se reconoce y se recupera la palabra del léxico mental. Algunos factores que se sabe que afectan estas métricas incluyen la frecuencia y familiaridad de la palabra, la ambigüedad de significado, la predictibilidad y la asociación semántica.

- *Skipping Rate*: Se utiliza para determinar la proporción de palabras que no reciben fijación, o sea las palabras que son salteadas durante la primera pasada de lectura. Esta métrica se suele reportar como una probabilidad (o porcentaje) y se calcula como el número total de pruebas en los que la palabra se omitió durante la primera pasada de lectura dividido el número total de pruebas. Por ejemplo, en la figura 1.8 vemos que la palabra crítica no fue salteada, por lo que esta prueba contribuiría a la cantidad total de pruebas pero no al numerador de la división.

- *First Fixation Duration (FFD)*: Se refiere al tiempo de la primera fijación realizada en una palabra o región de interés. En el caso de ejemplo se consideraría solamente el tiempo de la fijación número 3.

- *Single Fixation Duration*: Métrica muy parecida a la anterior, con la diferencia de que solo se tienen en cuenta las pruebas donde se ha realizado una sola fijación sobre la palabra en cuestión o ROI. En el caso de la palabra crítica de la figura 1.8 en el ejemplo no se vería contada al presentar más de una fijación.

- *First Pass Reading Time*: También conocida como *Gaze Duration* o FPRT, considera todas las fijaciones hechas en una palabra o ROI antes de que la mirada salga (ya sea a la izquierda o a la derecha) de la misma. Dentro del ejemplo presentado, la suma se realizaría entre los tiempos de las fijaciones 3 y 4.

1.2.1.2 Métricas intermedias

Existen métricas que son difíciles de clasificar como tempranas o tardías, como las relacionadas a las regresiones dentro de la lectura, ya que pueden indicar dependiendo de cómo se las utilice como dificultad al encontrar un elemento por primera vez, así como el tiempo posterior necesario para superar esa dificultad (Clifton et al., 2007).

- *Regression Path Duration*: también conocida como *go past time*, es una medida del tiempo dedicado a la palabra en sí y a cualquier parte anterior de la oración antes

de que el lector avance más allá de la palabra crítica hacia la derecha. También se puede medir una métrica parecida en términos de cantidad o como un porcentaje, es decir, cuántas pruebas tuvieron una regresión dentro del ROI. Volviendo a la figura, la métrica en este caso estaría compuesta por la suma de la duración de las fijaciones 3, 4, 5 y 6.

1.2.1.3 Métricas tardías

Las medidas tardías pueden no reflejar factores puramente léxicos y estar más influenciadas por propiedades contextuales, sintácticas o de nivel de discurso de lo que se está leyendo. Por ejemplo, la ambigüedad sintáctica puede llevar a fijaciones más prolongadas en una palabra o región crítica, así como a más regresiones al contexto anterior a medida que el lector se ve obligado a reevaluar el análisis inicial (Frazier & Rayner, 1982).

- *Total Reading Time*: Es la suma de todas las fijaciones realizadas en una palabra o ROI durante un ensayo. En el ejemplo, el *Total Reading Time* de la palabra crítica sería la suma de las fijaciones 3, 4, 6 y 9.
- *Re-reading Time*: Existen diferentes definiciones de la misma dentro de la literatura. Una de ellas se calcula a partir de la resta entre dos métricas anteriormente mencionadas, el *Regression Path Duration* menos el *First Pass Reading Time*. Abstrayendo al caso en particular, haciendo la resta de las dos métricas previas se tendría que el *Re-reading Time* es la suma de la fijación 5 y 6.
- *Second Pass Reading Time*: Se define como la suma de todas las fijaciones dentro de un ROI luego de haber abandonado la región de interés por primera vez. En el caso de ejemplo, la suma se realizaría entre las fijaciones 6 y 9.
- *Fixation Count*: Cantidad de fijaciones realizadas sobre un ROI. En este caso, un total de 4 fijaciones sobre la palabra crítica.

Es fundamental recordar que las medidas no son independientes entre sí: *First Fixation Duration* es parte de *First Pass Reading Time*, que a su vez es parte de *Total Reading Time*. De manera similar, *Total Reading Time* y *Fixation Count* generalmente están altamente correlacionados. Por lo tanto, a la hora de utilizar estas métricas, el objetivo debe ser analizar una gama de medidas para investigar el patrón general, y si un efecto sólo aparece en una de nuestras medidas, esto debe interpretarse con precaución.

1.3. Movimientos oculares en NLP

A medida que los modelos de NLP se vuelven cada vez más frecuentes en la sociedad, los investigadores han visto la necesidad de buscar maneras de cómo mejorar estos modelos, por ejemplo aprovechando información recopilada de manera pasiva de los lectores humanos, como las señales de movimientos oculares.

Esto genera que recientemente los movimientos oculares hayan incursionado en el campo de NLP, a partir de las relaciones existentes entre la duración de la mirada sobre las palabras (*gaze duration*) y la capacidad de predecirlas (Rayner, 1998, Kliegl et al., 2006). Particularmente, se realizaron análisis sobre los modelos del lenguaje en conjunto a datos de MO para capturar la relación entre ellos (Bianchi et al., 2020, Hofmann et al., 2017),

revelando información sobre la influencia de otras variables (como la frecuencia de la palabra) y permitiendo obtener una mejor comprensión de qué mecanismos actúan en el cerebro en determinado momento. Otros trabajos se han enfocado en la incorporación de los datos de MO para mejorar los modelos del estado del arte, en campos como el etiquetado de palabras (*Part of Speech Tagging*) (Barrett et al., 2016), compresión de oraciones (*Sentence compression*) (Klerke et al., 2016), traducción automática (*Machine translation*) (Sajjad et al., 2016), análisis de sentimiento (*Sentiment analysis*) (Mishra et al., 2017) y reconocimiento de entidades nombradas (*Named-entity Recognition*) (Hollenstein & Zhang, 2019). En todos los casos, se observaron mejoras frente a los modelos que no tenían en consideración estos datos.

Dentro de este trabajo en particular, con el objetivo de poder evaluar la incidencia de los movimientos oculares, especificamente de los procesos más inconcientes e inmediatos dentro de los modelos de lenguaje, se tomó la decisión de utilizar algunas de las métricas tempranas mencionadas anteriormente, como el *First Fixation Duration* y el *First Pass Reading Time*.

Curiosamente, la manera de incorporar esta información cognitiva a los modelos de NLP varía según el caso, por ejemplo existen casos en donde la información es concatenada a los *embeddings* del modelo (Hollenstein & Zhang, 2019), mientras que en otros casos se aprovecha de un tipo de entrenamiento conocido como aprendizaje multitarea (Klerke et al., 2016).

1.3.1. Aprendizaje Multitarea

¿En qué consiste entonces esta técnica? El aprendizaje multitarea (MTL, por sus siglas en inglés) consiste en entrenar una red neuronal para ejecutar múltiples tareas compartiendo algunas de las capas y parámetros de la red entre las mismas (Figura 1.9C). El objetivo es mejorar la capacidad de generalización del modelo aprovechando la información compartida entre las tareas, prediciendo varias cosas a la vez (Caruana, 1997).

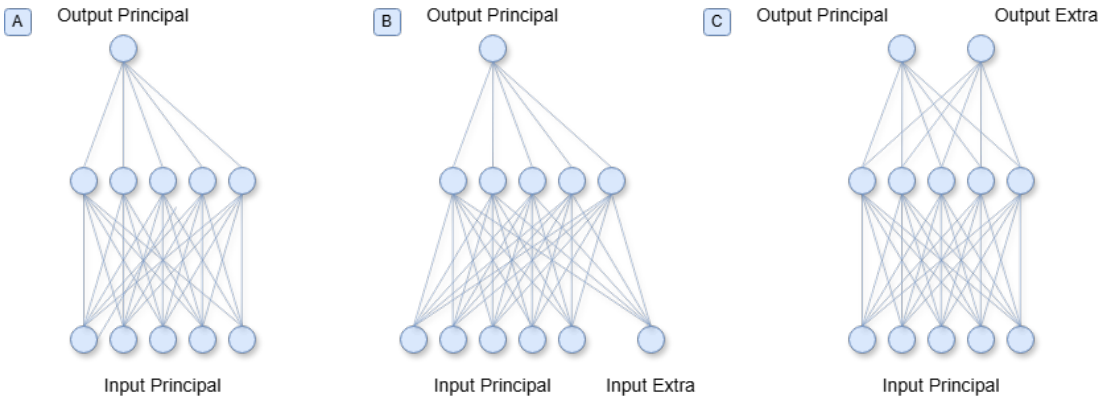


Fig. 1.9: Aprendizaje multitarea como es descrito por Caruana (1997). La red neuronal A es una red estándar la cual no utiliza ningún tipo de información extra. Por otro lado la red B incorpora comportamiento extra añadiendolo como input de la red. En el caso de la red C se aplica el aprendizaje multitarea: el comportamiento extra es añadido como output de la red y no como input.

Llevándolo al ámbito de NLP con la utilización de movimientos oculares, el aprendizaje

multitarea surge como una oportunidad, ya que al saber que el tiempo dedicado a fijarse en una palabra es un indicador de la atención, forzar al modelo de lenguaje a predecir la duración de la fijación de las palabras actúa como una forma de incorporar la atención (cognitiva) en él. En este trabajo, proponemos incorporar dicha información dentro de los *word embeddings* generados.

2. OBJETIVOS

El presente trabajo se inserta dentro de una línea de investigación llevada adelante por el equipo del Laboratorio de Inteligencia Artificial Aplicada (Juan Kamienkowski, Bruno Bianchi, y Fermín Travi). El objetivo general de esta línea de trabajo es el de mejorar modelos del campo del NLP a partir de adicionarles información cognitiva a los mismos durante el entrenamiento.

Para avanzar en línea con ese objetivo general, la presente tesis se propone los siguientes objetivos particulares:

- Preprocesar datos de movimientos oculares: Para el cual se realizaron una serie de experimentos registrando los movimientos oculares de los sujetos mientras realizaban tareas de lectura sobre una serie seleccionada de textos breves de fuentes literarias e informativas. A partir de estos resultados se buscará acondicionar los mismos para ser utilizados en el entrenamiento de los modelos de lenguaje.

- Seleccionar un modelo de lenguaje pertinente a la tarea a realizar: Partiendo en este caso de la arquitectura AWD-LSTM, se buscará adaptarla a la tarea de modelado de lenguaje a partir de los datos de movimientos oculares, ademas de realizarle distintas modificaciones para mejorar su rendimiento en diversos aspectos.

- Realizar diferentes entrenamientos del modelo de lenguaje elegido: Generando a partir de esta arquitectura ya adaptada a nuestras necesidades distintos modelos de lenguaje, ajustando los hiperparámetros y las condiciones de entrenamiento para obtener los mejores resultados posibles y asi obtener un modelo base con el cual trabajar.

- Analizar los modelos obtenidos: Comparando los *embeddings* obtenidos a partir de los modelos de lenguaje reentrenados con los movimientos oculares con distintos repositorios de juicios de similitud humanos, con el objetivo de observar que tan bien estos capturan las relaciones entre ciertas palabras.

3. METODOLOGÍAS

Para poder cumplir con los objetivos de este trabajo, se ha decidido dividir el mismo en 3 ejes principales. Cada uno de ellos serán presentados en los próximos capítulos:

1. **Preprocesamiento de los datos cognitivos (Sección 4):** en distintos formatos, los cuales servirán como entrada para el reentrenamiento del modelo de lenguaje .

2. **Adaptación de la implementación del modelo de lenguaje elegido (Sección 5):** partiendo de una implementación base a la cual se le realizaran diversos cambios con el objetivo de poder entrenar el modelo a nuestro gusto, además de arreglar distintos problemas que pueda presentar la implementación en el camino.

3. **Entrenamiento del modelo de lenguaje (Sección 6.4):** en busca de un modelo base adecuado el cual se pueda usar para reentrenar sobre el mismo. Una vez encontrado el mismo, también se reentrenará el modelo utilizando los datos obtenidos de los experimentos cognitivos.

4. **Exploración de técnicas de validación (Sección 6.5):** Luego se evaluará qué tan bien se comportan los *embeddings* de estos modelos con la información cognitiva comparándolos con dos *datasets* que nos permitirán obtener información sobre juicios de similitud por parte del ser humano:

    a) *SWOW-RP*: Base de datos obtenida a partir de tareas de asociación de palabras. Es decir, tareas en las cuales a partir de una palabra (señal), se asocia a la misma la primer palabra que al sujeto le salga de la mente. En el caso de SWOW-RP, se obtiene información de las primeras 3 palabras luego de mostrada la señal (Cabana et al., 2023).

    b) *Multi-Simlex*: Repositorio que provee conjuntos de pares de palabras a los cuales a cada una se le asigna una similitud a partir del juicio humano (Vulić et al., 2020).

4. PREPROCESAMIENTO DE LOS DATOS COGNITIVOS

4.1. Selección de los textos

El corpus que se utilizó no solo para entrenar y mejorar la AWD-LSTM, sino también como base para los experimentos que generaron los datos cognitivos constó de una serie de 20 textos cortos (400 - 1500 palabras) seleccionados de diferentes fuentes literarias e informativas. La mayoría de ellos fueron extraídos del libro "100 covers de cuentos clásicos" de Hernán Casciari. Las historias originales fueron escritas por diversos autores, las cuales fueron simplificadas, traducidas en el caso de ser necesario y reescritas al español por Casciari. De esta manera, se persiste una diversidad en los estilos literarios, a su vez manteniendo tanto la dificultad como el uso de palabras autóctonas del dialecto rioplatense. El objetivo de esta selección es disponer de textos naturales (no diseñados específicamente para ser parte de este corpus), que puedan ser leídos en un tiempo de alrededor de 5 minutos cada uno, sin mayores inconvenientes, maximizando la cantidad de palabras únicas utilizadas. Es decir textos cortos de escritura amena. Además de la longitud, se tuvieron en cuenta sesgos culturales y de género en la selección del material, balanceando la presencia y las actividades de los y las protagonistas.

Para la selección de textos, se consideraron las restricciones impuestas por el procesamiento de las mediciones. Al procesar datos de experimentos de seguimiento ocular, es crucial tener en cuenta la pérdida de datos al eliminar fijaciones en los extremos de líneas y párrafos, ya que estas fijaciones no se asocian directamente con el procesamiento del texto (Rayner, 1998). Por ello, se evitaron textos con muchas líneas cortas, oraciones breves o diálogos, que generan saltos de línea frecuentes. Además, para maximizar la utilidad del corpus comportamental, se priorizaron textos con baja proporción de palabras de muy baja frecuencia (Inhoff, 1984).

4.2. Experimentos de seguimiento ocular

Los experimentos realizados se dividieron en sesiones independientes de hasta 1 hora, asegurando un nivel de atención adecuado. El participante se encargaba de realizar una prueba, donde debía leer uno de los cuentos del corpus separado en diversas pantallas, mientras se obtenían registros de los movimientos oculares (fijaciones) de la persona a medida que iba avanzando. Al participante se le permitía volver a leer fragmentos ya visitados o incluso volver a pantallas anteriores, con la idea de simular la lectura del cuento de la manera más real posible. Al finalizar, se hicieron preguntas para verificar la atención del mismo durante el experimento. Si el participante no respondía correctamente, se descartaba el texto o la sesión.

El registro de movimientos oculares se realizó con equipos *EyeLink 1000* (SR Research, Ontario, Canadá) ya disponibles. Estos equipos poseen la mayor precisión temporal y espacial del mercado. Siguiendo las especificaciones del fabricante, se realizó una calibración del equipo antes y después de la lectura de cada texto. En esta se le pide al participante que fije la vista en determinados puntos en la pantalla. Sólo se aceptaron calibraciones/validaciones con un error promedio menor a 0.5 grados de ángulo visual. Luego, cada una de las sesiones fue analizada manualmente para verificar que los movimientos oculares

sigan los patrones esperados y no haya descalibraciones severas. Durante estos análisis se tomaron ciertas libertades, por ejemplo, debido a que los datos de movimientos oculares suelen estar sujetos a descalibración en el eje y (filas), ocurrió que existieron varias fijaciones que se registraron por encima de las palabras en las que los ojos están enfocados. Dado este escenario, se estableció un umbral manual para definir las líneas (es decir, a qué línea de texto pertenece cada fila de fijaciones) en cada pantalla para cada prueba. De esta manera se reubicaron fijaciones mal asignadas para preservar el mayor número posible de fijaciones.

Luego, si la prueba generada por el participante fue aceptada, se tenían en cuenta ciertos recaudos a la hora de descartar fijaciones:

- Las fijaciones iniciales se descartaban si generaban regresiones, así como fijaciones extremadamente largas o cortas ($\sim 1000ms$ y $\sim 50ms$), ya que tienden a corresponder a momentos en los que el participante no está prestando atención o a detecciones incorrectas de fijaciones, respectivamente.
- En cada pantalla, y en cada línea se descartaron las primeras y últimas fijaciones automáticamente.
- Para cada línea, cualquier fijación regresiva entre la primera y la más a la izquierda se considera el resultado de alteraciones oculomotores (barridas de retorno) y se descartaban.
- Palabras que contienen cualquiera de los siguientes caracteres:

¿, ?, ¡, !, ., -, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0

Al finalizar las experimentaciones y su posterior preprocesamiento, se contaron con los siguientes datos:

4.2.1. Métricas generales

El total de participantes del experimento de seguimiento ocular ($N = 76$) dió lugar a un total de 1126 pruebas. Del total, unas 110 pruebas fueron descartadas, ya sea debido a descalibraciones severas durante la realización del experimento, o debido a la falta de atención del sujeto durante el mismo. Por lo tanto, se puede observar que para el entreno de la AWD-LSTM se contó con el 91,12 % de las pruebas que originalmente se recabaron quedándonos con un total de 1016 pruebas (Tabla 4.1).

Tab. 4.1: Métricas generales asociadas a los experimentos de movimientos oculares. A pesar del alto porcentaje de pruebas finales, se descartaron 110 pruebas debido a descalibraciones severas o falta de atención de los participantes.

Participantes	Pruebas	Pruebas descartadas	Pruebas finales
76	1126	110	1016

A partir de estos filtros se obtuvieron un promedio de $56,3$ pruebas por cada texto con un desvío estándar de $4,30$ pruebas (Tabla 4.2). Esto nos da la pauta de que los datos que

fueron utilizados para entrenar el modelo de lenguaje están repartidos equitativamente en lo que a cuentos respecta. En el caso de querer ver más en detalle los resultados obtenidos durante la experimentación por cuento, consultar el anexo B.

Tab. 4.2: Cantidad de pruebas por cuento. Se puede observar que la cantidad de pruebas por cuento es equitativa, con un promedio de 56,3 pruebas por cuento y un desvío estándar de 4,30 pruebas.

Cuento	Pruebas
La noche de los feos	53
Cómo funcionan los bolsillos	50
La máscara de la Muerte Roja	53
Las fotografías	50
La salud de los enfermos	61
Buenos Aires	54
Wakefield	61
Cómo funciona caminar en la nieve	50
Ahora debería reírme, si no estuviera muerto	59
El espejo	60
Embarrar la magia	53
La lluvia de fuego	61
Educar para escalar y bucear	52
El golpe de gracia	54
La gallina degollada	60
Rubí y el lago danzante	60
La canción que cantábamos todos los días	60
El almohadón de plumas	61
Una rosa para Emilia	60
La de la Obsesión por la Patineta	54

4.3. Generación de textos de entrenamiento

Con todos los datos obtenidos a partir del corpus de texto y de los experimentos, se procedió a generar los textos de entrenamiento que se utilizaron para entrenar al modelo de lenguaje con el objetivo de notar una mejoría en su *performance* y en la representación de sus *embeddings* (Figura 4.1). En particular, se armaron 4 conjuntos de textos distintos para entrenar el modelo, diferenciándose en qué métrica se extrajo a partir de las fijaciones (en el caso de que se haya extraído alguna) hasta en la forma de generar el texto en sí.

1. *Textos*: En este caso directamente no se utiliza la información obtenida a partir de los experimentos y solamente se presentan los textos del corpus separados en 1 línea distinta por cada oración.

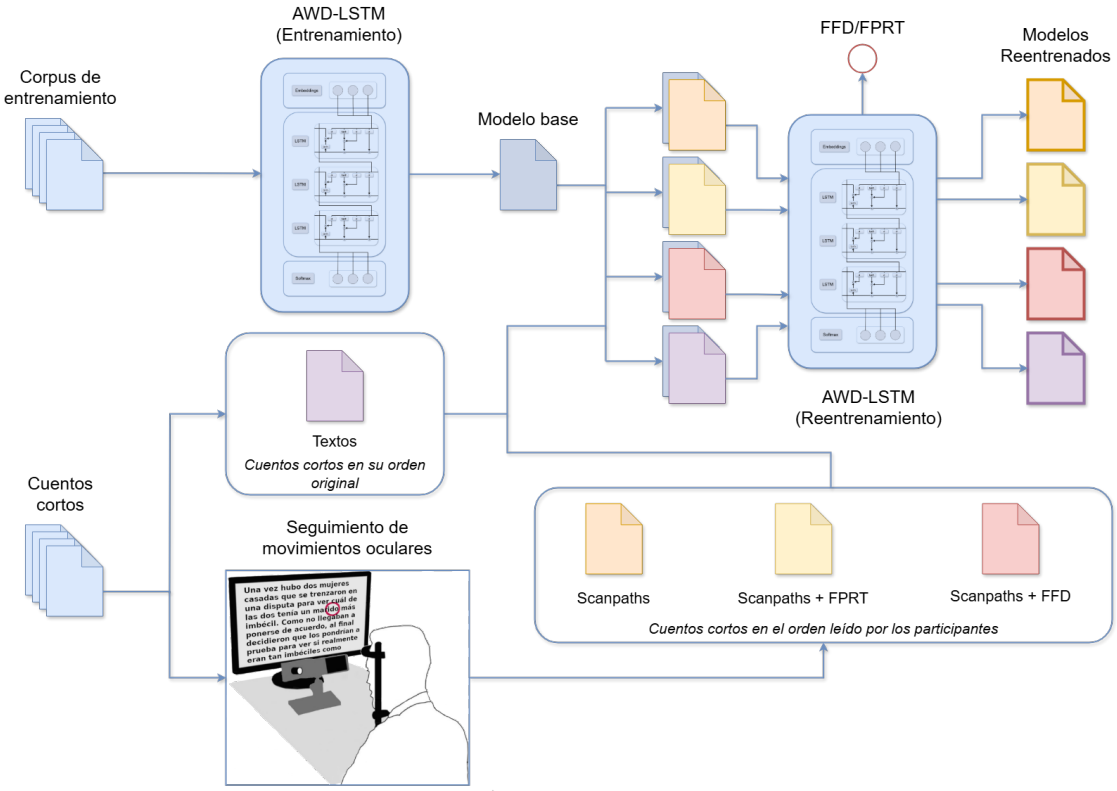


Fig. 4.1: Flujo de trabajo durante este proyecto. Se puede observar la generación de los textos basados en los cuentos acompañados con los movimientos oculares para reentrenar un modelo base

Para los conjuntos restantes, se decidió utilizar como texto base los *scanpaths* generados por los sujetos durante las pruebas. De esta manera, se puede tener una idea por sujeto de como fue realizándose la lectura de los cuentos. Ahora, para lograr que el modelo AWD-LSTM realice un mejor aprendizaje de las métricas extraídas de estos *scanpaths*, debido a la variedad de valores entre un sujeto y otro, se tomó la decisión de asociar cada palabra dentro de los *scanpaths* con el promedio de las métricas de todos los sujetos en vez de con el valor de la métrica para ese sujeto en particular. Con esto dicho, los conjuntos de reentrenamiento restantes fueron los siguientes:

2. *Scanpaths*: Representación de los textos en formato *scanpaths*, sin ninguna métrica en particular que lo acompañe. Por defecto, el valor de la métrica es reemplazado por un 0 para todas las palabras del *scanpath*, pero no se aprende sobre ella.

3. *Scanpaths_fprt*: Textos en formato *scanpaths* utilizando la métrica *Gaze Duration* o *First Pass Reading Time*, donde se suman las duraciones de todas las fijaciones en una palabra (antes de salir hacia la derecha o la izquierda).

4. *Scanpaths_ffd*: En este caso, los *scanpaths* se ven influenciados por la métrica *First Fixation Duration*, donde solo se conserva la primera fijación en una palabra (antes de salir hacia la derecha o la izquierda).

El objetivo con estos textos es analizar el comportamiento del modelo al añadir in-

formación cognitiva, en particular, métricas tempranas sobre los movimientos oculares de los sujetos, las cuales están asociadas entre otras cosas, a la asociación semántica de las palabras.

5. ADAPTACIÓN DE LA IMPLEMENTACIÓN

5.1. Implementación Base

Para poder empezar a adaptar la AWD-LSTM a los requerimientos necesarios, se decidió arrancar con una implementación del modelo de lenguaje obtenida de *GitHub*[1]. Esta implementación se eligió frente a otras debido a que esta se encontraba actualizada a las nuevas versiones de la librería *pytorch*, permitiendo así una mejor capacidad de adaptación a las mejoras futuras. En particular el repositorio presentaba una estructura bastante simple, compuesto por 4 archivos principales que daban forma a la arquitectura completa de la AWD-LSTM:

- *main.py*: Archivo el cual contenía la lógica del entrenamiento del modelo, permitiendo tunear los hiperparámetros del mismo. Previamente al entrenamiento, en este mismo también se extraían los *datasets* de entrenamiento, testing y validación a partir de importarlos localmente. Luego, estos se dividían en *batches* para poder entrenar al modelo final y validarlo durante las distintas épocas. Para finalizar, el estado del modelo se guardaba cada vez que se lograba encontrar un modelo mejor al finalizar una época. En el caso de que se quisiera interrumpir el script durante el entrenamiento, el código que englobaba al mismo presentaba manejo de excepciones frente a una interrupción voluntaria del usuario, deteniendo el entrenamiento y devolviendo la *perplexity* del último modelo guardado frente al *dataset* de testing.

- *finetune.py*: Archivo casi idéntico al anterior, utilizado para reentrenar el modelo en el caso que se preste necesario. Sus principales diferencias recaían en que este archivo permitía inicializar el modelo con alguno ya entrenado previamente, cargando los pesos de la red neuronal a partir de un diccionario que almacenaba el estado de la misma. Además otra característica de este proceso de reentrenamiento es que no estaba implementado de manera que se le pueda indicar al mismo que este reentrene al modelo por una cantidad fija de épocas, sino que él mismo las haría indefinidamente hasta que la condición no monotónica del optimizador se cumpla.

- *ntasgd.py*: Archivo el cual definía el optimizador del modelo.

- *model.py*: Archivo el cual albergaba todas las clases que en su conjunto representaban a la red neuronal completa que define a la AWD-LSTM. En este mismo se definen todas las capas de la arquitectura, además de especificar cómo se propaga para adelante la entrada de la red, alimentando así los pesos de la misma durante el entrenamiento.

Esta implementación presentaba una serie de hiperparámetros los cuales se podían variar para generar distintos modelos de lenguaje. Para más información sobre los mismos o sus valores predeterminados, consultar el anexo A. A menos que se indique lo contrario, durante la realización de este trabajo estos serán los valores por defecto que se utilizaran para cada uno de los experimentos.

[1] https://github.com/ahmetumutdurmus/awd-lstm

Por otro lado, yendo más en detalle sobre el archivo *main.py*, en particular sobre de qué manera el programa prepara el corpus de texto para poder alimentar al modelo durante el entrenamiento, se pudo observar que este funciona de una manera un poco particular (Figura 5.1).

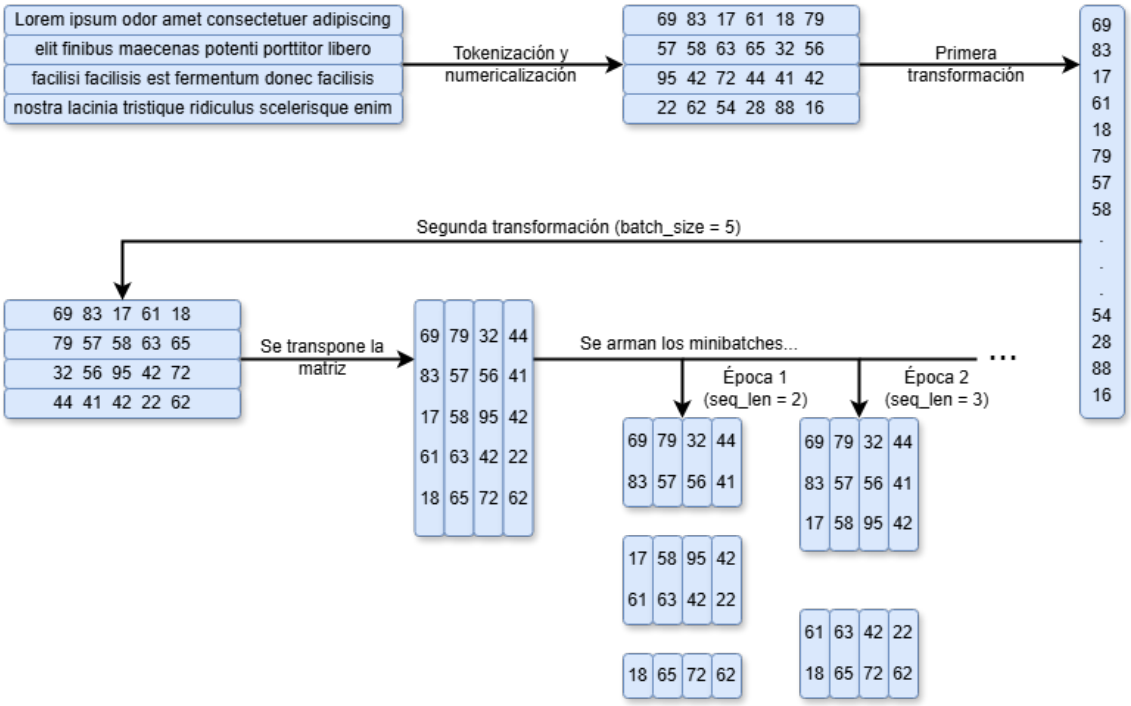


Fig. 5.1: Pipeline usado para generar el *batch* que alimentará al modelo durante el entrenamiento.

En primer lugar, se separa al corpus de texto en oraciones, donde cada oración se divide en una lista de *tokens*: palabras (sin signos de puntuación, en minúscula, y sin dígitos ni caracteres no romanos) y signos de puntuación, más la introducción de ciertos caracteres extra para brindar más información sobre el contexto de la palabra, desde caracteres que permiten indicar que la palabra siguiente empieza con mayúsculas hasta caracteres que indican el fin de una oración (tokenización). Cada uno de estos *tokens* está representado por un único número (numericalización). Estas listas de números (es decir, las oraciones) son transformadas en un tensor de una única columna con tantas filas como *tokens* haya presentes en la Figura 5.1. Este tensor luego es nuevamente transformado, esta vez en un tensor con una cantidad de filas igual a la cantidad de *batches*. Previamente, con el objetivo de que la cantidad de palabras sea divisible por esta cantidad de lotes y el redimensionamiento del tensor sea posible, se descartaba un pedazo del corpus del final del texto. Una vez hecho esto, se transpone el tensor, quedando una cantidad de columnas igual al tamaño del *batch*.

Por último, para alimentar al modelo, se separan estos *batches* en lotes aún más pequeños, llamados *minibatches*, dependiendo del tamaño de la secuencia que se use en esa época, ya que este tamaño es variable y se recalcula cada vez que se inicia una de ellas (Merity et al., 2017).

5.2. Validación de Implementación Base

Una vez presentada la implementación base, se buscó estar seguros de que la misma se comportaba de manera similar en cuanto a capacidad de predicción con una implementación de AWD-LSTM. Para poder averiguar empíricamente esta cuestión, se decidió comparar la implementación base obtenida con la implementación proveída por la librería de *Python* denominada *Fastai*, la cual presenta diversas arquitecturas de modelos de lenguajes. Sin embargo, como esta no permite la modificación de su estructura a nivel red neuronal, se la había descartado como implementación base. Previamente a esto, se modificó la forma en la cual la LSTM tokeniza y numericaliza los textos, usando también la librería *Fastai* para tokenizarlos, no solo por comodidad, sino también debido a que esta genera nuevos *tokens* de acuerdo a distintos contextos. A futuro esta modificación también serviría en términos de tiempo de ejecución y espacio de memoria, ya que la implementación base obtenía los *tokens* seleccionando el texto completo y transformándolo en un conjunto de palabras, lo cual para corpus muy grandes no sería factible. Los resultados de la comparación se pueden encontrar en la experimentación correspondiente.

Con nuestra implementación empíricamente validada, se continuó la adaptación modificando la forma en la que el modelo recibe el corpus de entrenamiento. En un principio este recibía el corpus a partir de un archivo local. Con el objetivo de poder entrenar la LSTM con textos sustancialmente más grandes en tamaño, lo que se hizo fue adaptar la implementación con una clase llamada *Corpora*, utilizada en trabajos previos en el laboratorio para generar conjuntos de corpus de texto, permitiendo agregar los mismos de manera local o de manera remota a partir de un repositorio de *Huggingface*. De esta manera, sería posible no solo entrenar el modelo con un corpus remoto, como es el caso de *Wikipedia*, sino también reentrenar el modelo con los *scanpaths* generados por los experimentos de movimientos oculares.

Otro cambio que se aplicó a la implementación relacionado a los corpus, fue la eliminación del corpus de testing, para simplificar el código, ya que no lo veíamos pertinente para las experimentaciones futuras. Con esto dicho, en resumidas cuentas para el entrenamiento del modelo la implementación quedó con dos corpus, uno de entrenamiento y otro de validación, correspondientes al 80 % y 20 % del *dataset* original respectivamente. Esta división se hizo de manera aleatoria utilizando una semilla para permitir la replicabilidad de los experimentos en un futuro.

5.3. Métricas

Para poder probar el comportamiento de nuestro modelo a futuro con las experimentaciones, en este trabajo nos hemos centrado en dos métricas generales:

- *Perplexity*: La cual se utilizó para medir la capacidad predictiva entre distintos modelos de LSTM, basados en un mismo vocabulario.
- *Correlación con juicios de valor humanos*: Utilizando la correlación de *Spearman* y la distancia coseno para averiguar si los *embeddings* generados por la LSTM son capaces de captar las mismas similitudes entre pares de palabras que los capturados por los seres humanos. En caso de querer ver más en detalle cómo se genera esta correlación, ver sección 6.1.2.

Para la primera métrica, la implementación base ya presenta con la capacidad de poder calcular la *perplexity* del modelo a lo largo del entrenamiento. Sin embargo, para obtener la segunda métrica se necesita una manera de obtener *embeddings* a partir de la LSTM. A priori, el modelo no es un modelo dedicado a devolver los *embeddings* de un vocabulario en particular como Word2Vec, sino más bien es un modelo dedicado a predecir la siguiente palabra dado un contexto. Afortunadamente, la arquitectura AWD-LSTM presenta una capa de *embeddings* dentro de la red neuronal, por lo cual el siguiente cambio que se realizó a la arquitectura es el de, una vez finalizado el entrenamiento del modelo, extraer esta capa de *embeddings*, la cual es una matriz de tamaño vocabulario por dimensión del *embedding*, el cual es un hiperparámetro del modelo.

Una vez extraídos los *embeddings* de la red, estos eran copiados en un archivo de texto en un formato que permita utilizar los *embeddings* para ser importados por librerías como *Gensim*. Así, será sencillo hacer la comparación con otros *embeddings*. El formato (Figura 5.2) consiste en:

- Como primera línea del archivo se encuentran las dimensiones de la capa de *embeddings*.
- En las siguientes líneas aparecen, una por cada palabra del vocabulario, la palabra en sí junto a su *embedding*, separadas por un espacio en blanco.

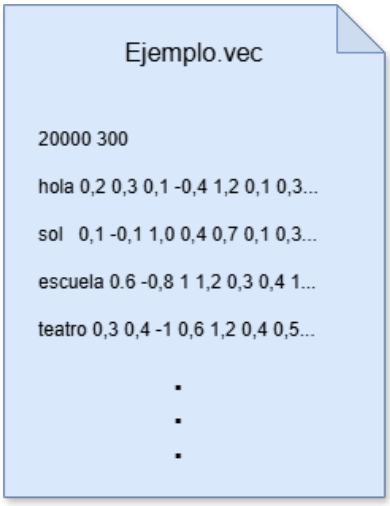


Fig. 5.2: Ejemplo de archivo de texto donde se guardarán los *embeddings* resultante del entrenamiento. En este caso, 20000 sería el tamaño del vocabulario mientras que 300 es el tamaño del *embedding*.

Además, para poder tener una idea más detallada de cómo evolucionan estos *embeddings* a medida que se entrena la LSTM, se tomó la decisión de no solo extraer los *embeddings* finalizado el entrenamiento, sino al terminar cada época del mismo. Luego, estos serían comparados con los juicios de similitud de *Multi-Simlex*, obteniendo la correlación entre los mismos en cada época.

5.4. Incorporación de movimientos oculares al modelo

Como mencionamos previamente, el objetivo principal de este trabajo es el de observar la performance del modelo, luego de añadirle información sobre los movimientos oculares de personas durante la lectura del texto con el cual se está alimentando al mismo. Para poder cumplir esto entonces, se decidió hacer una modificación a la red neuronal clásica de la arquitectura AWD-LSTM. Basados en la idea del aprendizaje multitarea, se tomó la decisión de obligar al modelo a que no solo prediga la siguiente palabra dado un contexto, sino que también prediga las métricas relacionadas con movimientos oculares que más se han estudiado en la literatura: *First Fixation Duration* (FFD) y *First Pass Regression Time* (FPRT, también llamado *Gaze Duration*) (ver 1.2.1 para su definición).

Yendo más en detalle, se añadió una capa extra en el modelo, compuesta por una regresión lineal, la cual recibe el *batch* con el cual se está alimentando en ese momento a la red y devuelve un resultado. Una vez hecha la predicción, con el objetivo de que la misma vaya mejorando a lo largo de las épocas, se la compara con las métricas reales obtenidas del experimento utilizando el error absoluto medio como función de pérdida. En los casos en donde no existieron métricas relacionadas a movimientos oculares se tomó la decisión de directamente no comparar esta predicción generada, para así no influir en el entrenamiento del resto de la red.

5.5. Problemas de memoria

5.5.1. Primera transformación

Luego de haberse realizado testeos preliminares para asegurar el funcionamiento de la LSTM de manera correcta, el proyecto se topó con un problema importante a solucionar. El código original de la implementación, especialmente en la sección en donde se acondiciona el corpus para transformarlo en un *dataset* adecuado, para luego ser separado en *batches* de texto que serían enviados a la LSTM durante el correr de las épocas, no estaba preparado para poder manejar corpus de gran tamaño.

Como se mencionó anteriormente, durante la ejecución los corpus eran transformados en un tensor de 1 sola columna y 1 fila por cada palabra que compusiera el mismo, para luego ser nuevamente transformados de acuerdo al tamaño del *batch*. Luego de examinar el código detenidamente, se llegó a la conclusión que existía un problema en esta primera transformación, ya que se hacía utilizando las herramientas básicas del lenguaje de programación y no apoyándose en las librerías externas que dieran un mejor uso de la memoria, haciendo que a medida que se iba generando el tensor, la memoria de la computadora se fuera llenando, llegando a terminar el programa abruptamente cuando el tamaño del corpus era lo suficientemente grande.

Como esto no permitía que la LSTM pudiera ser entrenada con textos extensos, se buscó una manera de mejorar el uso de la memoria en este paso del programa. La solución propuesta fue la de aprovecharse de la interfaz de la librería *Huggingface*, la cual no solo permite la importación de corpus de entrenamiento, sino que también provee una simple interfaz para poder realizar transformaciones sobre el *dataset*, permitiendo así poder replicar la primer transformación del corpus, a la vez de reducir en gran medida el consumo de la memoria en este paso en particular, ya que la herramienta permite ejecutar la transformación de manera paralela dentro de la memoria. Incluso se aprovechó esta herramienta para poder realizar la tokenización y numericalización del corpus, reduciendo no sólo es-

pacio sino tiempo, al también proveer la posibilidad de ejecutar las transformaciones de manera paralela entre varios hilos de ejecución.

Estas modificaciones permitieron que la LSTM se pudiera entrenar con porcentajes cercanos al 30 % de un corpus de texto que contenía alrededor de 28 millones de oraciones. Sin embargo, como los experimentos realizados con otros modelos como Word2Vec se habían llevado a cabo con la totalidad de ese corpus, se decidió probar de entrenar el modelo con un porcentaje mucho mayor.

5.5.2. Segunda transformación

Desafortunadamente, se pudieron observar de vuelta problemas de memoria al entrenar con la totalidad del corpus. Al analizar más en detalle el porqué de esto, nos dimos cuenta que el programa se detenía previo a la segunda transformación mencionada anteriormente, o sea, la que dividía al *dataset* en *batches*. También se encontró un pico de uso de la memoria cuando se accedía al *dataset* en particular, dando la pauta de que el acceso al *dataset* tampoco se estaba haciendo de manera eficiente en este caso.

Un primer enfoque el cual se tomó fue el de recurrir a la librería *PyTorch* y utilizar los *dataloaders* que esta provee, permitiendo un acceso del *dataset* de a *batches* y evitando tener que colocarlo entero en memoria. Esta idea traía un problema crucial: el *dataloader* va generando lotes cuyo texto es contiguo en el corpus, lo cual es evitado en el código original utilizando las transformaciones anteriormente mencionadas, ergo, el conjunto final de *minibatches* resultaba distinto.

Como se buscó mejorar el uso de la memoria del modelo sin alterar íntegramente la lógica del código, se decidió ir por otra solución. Esta solución implicó la utilización de nuevo de la librería *Huggingface* y otro de sus elementos que provee para la alimentación de modelos de lenguaje con corpus de datos extensos, el uso del *sharding* (Figura 5.3).

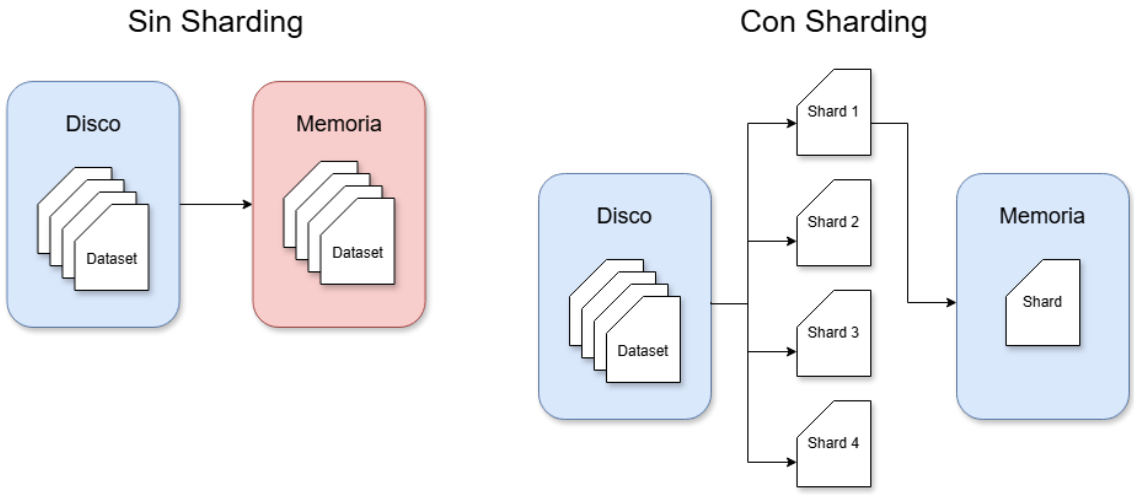


Fig. 5.3: Sharding, consiste en dividir el *dataset* en fragmentos más pequeños, permitiendo un uso más eficiente de la memoria, ya que lo que ingresa en la memoria es el fragmento y no el *dataset* entero.

En particular, en este caso previo a la realización de las transformaciones para generar los *minibatches*, el *dataset* era dividido en una cantidad fija de fragmentos (modificable como parámetro del *script* de entrenamiento), para luego realizar el entrenamiento del

modelo de manera normal con el primero de ellos, hasta llegar al último, donde terminado este se daría como finalizada la época. Esto generó mejoras sustanciales en el uso de la memoria, permitiendo el entrenamiento de la LSTM con el 100 % del *dataset* con el uso de 5 fragmentos. No solo eso, sino también permite a futuro poder aumentar la cantidad de fragmentos en el caso de que este modelo se quiera entrenar con un corpus aún más grande.

5.5.3. Red neuronal

Para finalizar, también se hizo una modificación a cómo computa la red neuronal los resultados. En particular, previo a la capa en donde se encuentran las celdas LSTM, se encuentra la capa de *embeddings*, donde el *batch* de palabras que se envían como entrada es transformado en sus respectivos *embeddings*, luego de aplicarse un *dropout* de algunos de estos *embeddings* con el objetivo de reducir el sobreajuste. En esta capa, sin embargo, se descubrió que la manera en que se hacía esto en la implementación original era ciertamente ineficiente en términos de memoria.

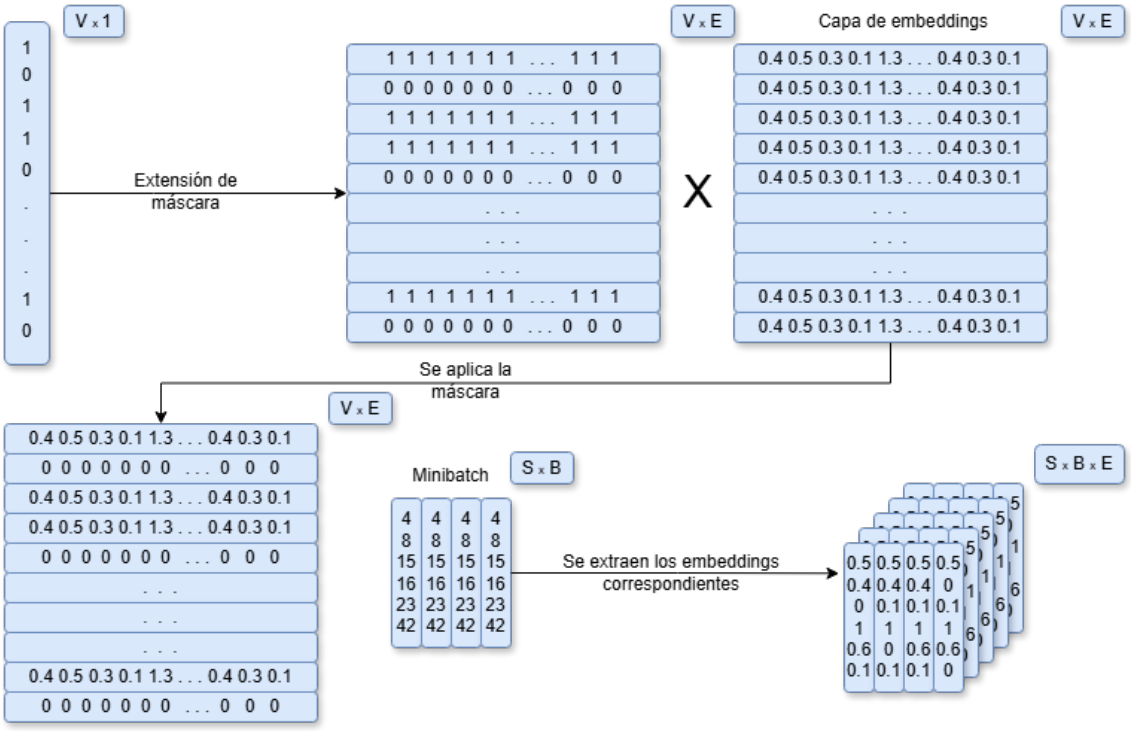


Fig. 5.4: Algoritmo original para extraer los *embeddings* de la capa de *embeddings*, donde V es el tamaño del vocabulario, E es el tamaño del *embedding*, B es el tamaño del *batch* y S es el largo de la secuencia

Para empezar, en la implementación se genera una máscara de una dimensión, del tamaño del vocabulario (V), en la cual cada punto corresponde a una palabra en particular y es marcada con un 1 o un 0 de acuerdo a una distribución de bernoulli. Luego esta máscara es extendida a una matriz de dimensión $V + E$ (donde E es el tamaño de los *embeddings*), literalmente copiando los valores anteriores en memoria cuantas veces sean necesarios. No solo eso, además esta máscara es multiplicada celda por celda con la capa

de *embeddings* de la red neuronal, deshabilitando algunos *embeddings* de manera aleatoria, pero generando a su vez otra matriz provisoria de tamaño $V + E$. Por último se extraen los *embeddings* de esta matriz provisoria para cada uno de los *tokens* del *batch*. Esta implementación nos pareció ineficiente, especialmente en casos en donde V es grande, pudiendo llegar a ocupar gran parte de la memoria de la GPU innecesariamente (Figura 5.4).

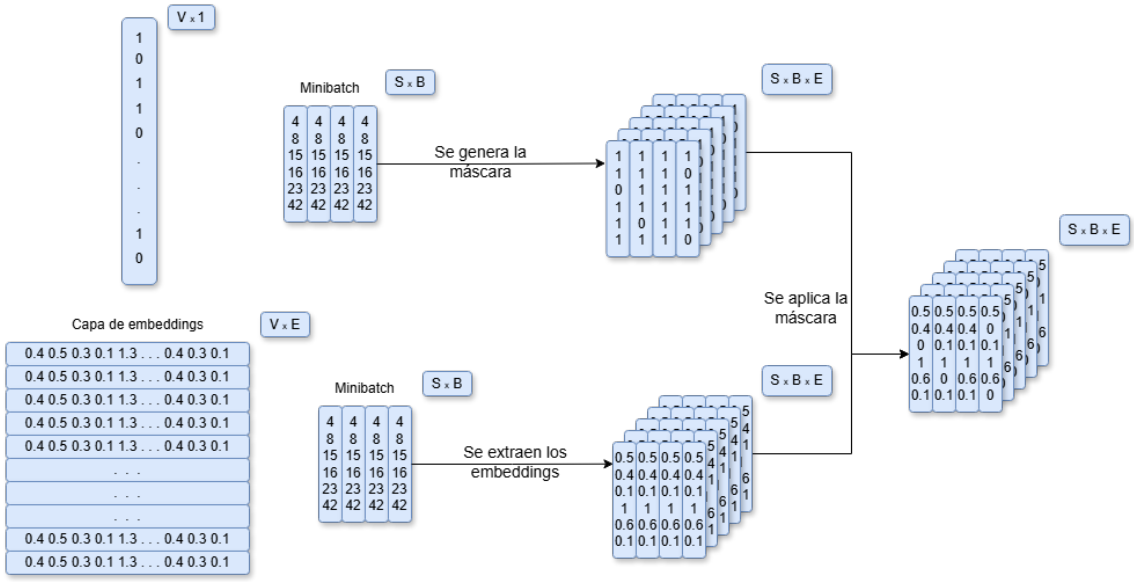


Fig. 5.5: Optimización del algoritmo para extraer los *embeddings*. Se definen las mismas variables que en el caso anterior.

Para solucionarlo, se decidió ir por otra implementación más optimizada. En este caso, no se extendió la máscara y se mantuvo como un tensor de una sola dimensión. Luego, directamente se extraen para el *batch* en cuestión los *embeddings* de la capa, generando un tensor de 3 dimensiones, esta vez de tamaño $S \times B \times E$ (donde S es el largo de la secuencia y B el tamaño del *batch*). Además, se genera otro tensor, esta vez solamente de tamaño $S \times B$, donde dentro de él se encuentran los resultados de la máscara para *token* del *batch*, indicando si esa palabra debería ser deshabilitada o no. Por último, se multiplican celda por celda ambos tensores, en definitiva multiplicando ya sea por 1 o por 0 cada uno de los *embeddings* del mismo. Si analizamos detenidamente las dimensiones resultantes de los dos algoritmos, podemos ver, teniendo en cuenta que $S \times B \ll V$, que los tensores intermedios del segundo algoritmo son mucho más pequeños que los del algoritmo original, ergo ocupando menos espacio de la memoria de la GPU.

5.6. Abstracción del código

Pudiendo ya entrenar sin problemas el modelo con el corpus, se decidió realizar varios cambios estructurales a los archivos con el objetivo de simplificar el código, mejorar la lectura del mismo y poder incorporarlo al sistema original del proyecto, el cual permitía el entrenamiento de otras arquitecturas de manera general, como Word2Vec.

El primero de los cambios fue el de abstraer el código del archivo *main.py* dentro de una clase *AwdLSTM*. Luego, al notar que el código perteneciente al archivo *finetune.py*

era idéntico al de su contraparte, se decidió abstraer aún más la clase, transformándola en la interfaz de una jerarquía de clases. Esta jerarquía estaría conformada por una clase *AwdLSTMForFinetuning* y otra llamada *AwdLSTMForTraining*, encargadas del reentrenamiento y entrenamiento del modelo respectivamente. Dentro de estas clases sólo se encuentran las funcionalidades específicas a esa etapa del modelo. Las funcionalidades que se compartan entre estas dos serían englobadas por la interfaz *AwdLSTM*, evitando así código repetido.

Además, aprovechándose de la similitud entre las dos clases, se incorporó polimorfismo a la jerarquía, permitiendo que se simplifique el código aún más al utilizar en funcionalidades que comparten las dos, métodos cuya implementación dependerá del estado del modelo en el que esté. El código se encuentra publicado en *Github*[2] para su consulta.

5.7. Modificaciones dentro del reentrenamiento

Una vez ya adaptada toda la etapa de entrenamiento, nuestro siguiente objetivo fue el de modificar la etapa de reentrenamiento para poder arrancar con la experimentación. Un primer cambio fue el de modificar la LSTM para que esta implemente un método estático que funcione como *factory method*, pudiendo así englobar la funcionalidad de generar la clase que define al modelo en un contexto de entrenamiento o en un contexto de finetuning en un solo lugar. Otra cosa que se hizo fue modificar el criterio de corte del modelo cuando este está finetuneando. En un principio, el reentrenamiento no se hacía durante una cantidad fija de épocas, sino que este terminaba cuando la LSTM alcanzaba la condición no monotónica del optimizador, lo cual con la idea en mente de simplificar el código, se decidió modificar y dejarlo igual a como actúa la LSTM en un contexto de entrenamiento. Por último, se aseguró que el vocabulario presente en los textos de reentrenamiento estuviera presente en el vocabulario del modelo final. Para eso, en la etapa de entrenamiento se añadió como parte del vocabulario no solo las palabras presentes al corpus de entrenamiento, sino también todas las palabras presentes en el corpus utilizado durante los experimentos de movimientos oculares.

[2] https://github.com/FerminT/LMET

6. EXPERIMENTACIÓN

6.1. Materiales

En la presente sección enunciamos los diversos recursos que hemos utilizado y ayudarán a entender cómo se llevaron a cabo los experimentos, tanto previos en el laboratorio como los del presente trabajo.

6.1.1. Corpus de entrenamiento

Para poder entrenar no solo los modelos de lenguaje generados por la AWD-LSTM, sino también generar los *embeddings* dentro del modelo de Word2Vec se utilizó un corpus grande de texto en español. En particular, se decidió utilizar el corpus llamado *all_wikis*, proveniente del repositorio de textos en español *large_spanish_corpus*, creado por José Cañete[1]. Este corpus está compuesto por fragmentos extraídos de distintas wikis en español, como *Wikipedia*, *Wikinews*, *Wikiquotes* y muchas más, completando un corpus de 28,109,484 filas, correspondiente a una oración por cada una. A menos que se indique lo contrario, los modelos serán entrenados con este corpus.

Además, todos los gráficos relacionados a la métrica *perplexity* fueron realizados sobre el *dataset* de validación del corpus, dejando asi el *dataset* de entrenamiento exclusivamente para el entrenamiento de los modelos. Para más informacion sobre como se dividió el corpus, ver la sección 5.2.

6.1.2. Experimentos de similitud

Como mencionamos anteriormente, el eje principal de este trabajo es el de analizar el comportamiento de modelos generados a partir de la arquitectura AWD-LSTM luego de incorporarles información cognitiva. Para esto, se decidió comparar los *embeddings* obtenidos de los modelos resultantes con dos bases de datos que nos permiten obtener información sobre juicios de similitud humanos entre pares de palabras: SWOW-RP y Multi-Simlex. A continuación, se muestra cómo fueron preprocesados estos repositorios y de qué manera luego fueron comparados con los *embeddings* generados tanto en el presente trabajo como los de trabajos previos en el laboratorio.

6.1.2.1 Multi-Simlex

Como habíamos denotado previamente, Multi-Simlex es un repositorio que cuenta con respuestas sobre juicios de similitud entre pares de palabras brindados por 10 anotadores distintos. Estos sujetos dieron, para cada par de palabras, un valor de similitud entre 0 y 6, donde 0 se considera muy poca similitud y 6 extremadamente similares. Para poder tener una métrica más compacta con la cual comparar los *embeddings*, se decidió tomar el promedio de los valores dados por estos anotadores como nuestra medida de similitud final.

Luego, para realizar la comparación con los *embeddings* de los modelos, se calcula la correlación de *Spearman* de todos los pares de palabras del repositorio, comparando, por

[1] https://huggingface.co/datasets/josecannete/large_spanish_corpus

un lado, el promedio de los valores dados por estos anotadores y, por el otro, la similitud coseno entre los *embeddings* de ambas palabras, extraídos a partir del modelo.

6.1.2.2 SWOW-RP

Por otro lado, SWOW-RP es una base de datos resultante de varias tareas de asociación de palabras provenientes del dialecto rioplatense del español (Cabana et al., 2023), donde a partir de una palabra inicial (*cue*), los sujetos contestaban con las primeras tres palabras que se les venían a la mente. A partir de ahí, el repositorio brinda información ya procesada sobre estas tareas, ya sea para únicamente la primera respuesta o para las tres. Por ejemplo, brinda la frecuencia de una respuesta sobre una *cue* o incluso la probabilidad condicional de haber elegido una respuesta a partir de una *cue* en particular. En nuestro caso, se tomó la decisión de tomar esta última como medida de similitud entre dos pares de palabras, ya que tomar la frecuencia hubiera sido imposible, al tener cantidades distintas de respuestas para cada *cue*. Además se eligió contar con la versión de los datos que tiene en cuenta las tres respuestas para mayor abarcabilidad.

Luego, para tener una idea más en profundidad del efecto que la información cognitiva puede generar sobre los *embeddings*, en este caso se optó por dividir los pares de palabras de este repositorio en dos. Por un lado, se agruparon los pares de palabras en donde ambas palabras estuvieran presentes en los cuentos; es decir, las palabras que recibieron atributos cognitivos durante el reentrenamiento. En el caso de que algunas de las palabras del par no incluyera información cognitiva, sería agrupada en el grupo restante.

Por último, se calculó la correlación de *Spearman* de la misma manera que en el caso anterior, con la salvedad de que, con el objetivo de obtener mayor rigurosidad estadística, se decidió hacer esta comparación no sobre el espacio de pares de palabras total, sino sobre un muestreo de 1000 pares generados con reposición unas 100 veces.

6.2. Trabajos previos

Previo a este trabajo, dentro del laboratorio de la mano del doctorando Fermin Travi se estuvo investigando sobre la incidencia de la información cognitiva sobre otro tipo de modelos previos a la arquitectura AWD-LSTM, en este caso, Word2Vec. Haciendo hincapié en el mismo, se utilizó una arquitectura de Word2Vec basada en *skip-grams* y se entrenó al modelo utilizando como corpus base el mismo corpus aprovechado en este trabajo, valiéndose del 100 % del mismo durante el entrenamiento por una cantidad de 5 épocas. Luego, los *embeddings* resultantes del modelo fueron comparados con los repositorios de juicios de valor humanos presentados anteriormente, calculando la correlación entre los *embeddings* y los datos de los repositorios (Figura 6.1)

En particular, para el caso de Multi-Simlex, el modelo de Word2Vec generado a partir de *skip-grams* reportó una correlación de *Spearman* de $0,458$, la cual se utilizará en los experimentos venideros como base a la cual comparar los modelos generados por la AWD-LSTM. Además, se evidencia que la correlación empeora levemente al reentrenar el modelo con la información cognitiva, lo cual se considera esperable que ocurra al haber una gran cantidad de palabras que no se vieron influenciadas por el reentrenamiento. Sin embargo, se puede observar que este decaimiento es levemente menor en los modelos con las métricas presentes.

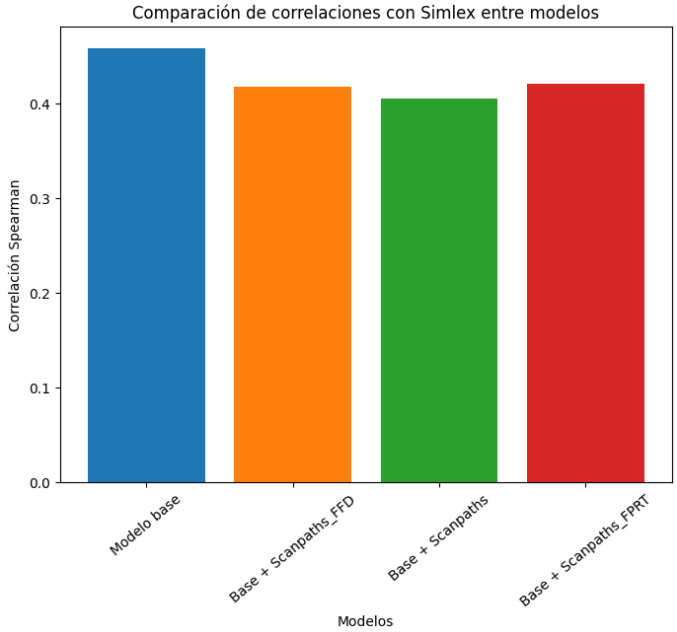


Fig. 6.1: Correlaciones entre distintos modelos de Word2Vec con los juicios de valor generados por Multi-Simlex. Entre los modelos comparados se encuentran un modelo base implementado a partir de *skip-grams*, y 3 modelos reentrenados a partir de este, uno con los *scanpaths* sin ninguna métrica asociada y otros dos con los *scanpaths* y las métricas FFD y FPRT asociadas respectivamente.

6.2.1. Reentrenamiento

Con este modelo base, a su vez, se decidió reentrenar el modelo de *embeddings* añadiendo información generada a partir de movimientos oculares, haciendo uso de los *scanpaths* generados por los experimentos de información cognitiva. Estos fueron reentrenados por una cantidad de 5 épocas valiéndose del 100 % de estos textos, introduciendo información sobre los movimientos oculares de los sujetos durante la lectura de los mismos.

Los *embeddings* generados por estos modelos fueron comparados con los datos del repositorio SWOW-RP (Figura 6.2). Como se puede observar, los modelos reentrenados con los *scanpaths* muestran una peor correlación con SWOW-RP comparados con el modelo original, indicando de cierta manera que con esta forma de comparación los *embeddings* reentrenados parecen captar peor las similitudes entre los pares de palabras obtenidos en el repositorio. Esto al menos, para el caso de las palabras en estímulo, o sea las palabras dentro del vocabulario de los *scanpaths*. Para el resto de palabras dentro del vocabulario del modelo, no parecen haber cambios sustanciales.

6.3. Validación de implementación base

6.3.1. Comparación entre implementaciones de AWD-LSTM

Tal y como se comentó en secciones anteriores, luego de obtener la implementación base de la arquitectura AWD-LSTM, se propuso validarla con otra implementación confiable que nos permita constatar que el código funciona correctamente. En este caso, se utilizó la implementación de la librería *Fastai*. Para realizar esta comparación, se decidió realizar un

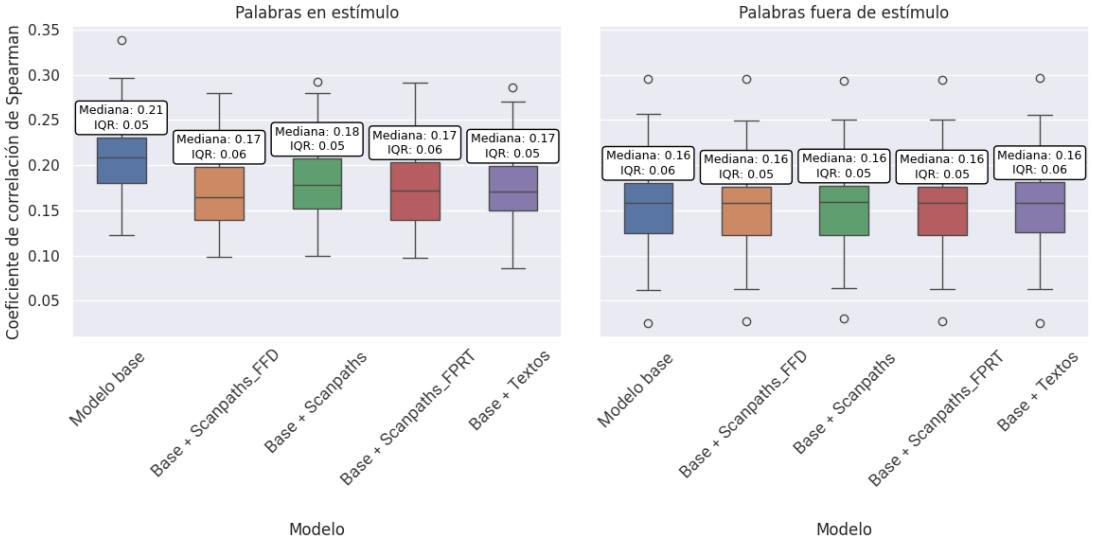


Fig. 6.2: Correlaciones entre distintos modelos de Word2Vec y los juicios de similitud generados a partir del repositorio SWOW-RP. La comparación parte con los mismos modelos comparados con Multi-Simlex en la figura 6.1, sin embargo en este caso, se le añade el modelo reentrenado con los textos originales de los cuentos.

experimento entrenando dos modelos distintos: uno a partir de la herramienta brindada por *Fastai* y otro entrenandolo a partir de la implementación base, con el objetivo de poder corroborar que ambos modelos sean parecidos en términos de *performance* (validando empíricamente contra la implementación base). Durante el correr de las épocas de entrenamiento de ambos modelos, se tomó nota de la *performance*, utilizando la *perplexity* como métrica de los mismos frente al dataset de validación. Ambos modelos fueron entrenados utilizando uno de los corpus empleados en la experimentación del trabajo original de la AWD-LSTM (Merity et al., 2017), llamado *Penn Treebank Dataset*, el cual engloba una serie de artículos en inglés del *Wall Street Journal*, llegando a cubrir una cantidad de 38,219 oraciones. En particular, ambos modelos fueron entrenados durante 500 épocas y se los configuró de manera tal que los modelos resultantes fueran los más parecidos posibles. Incluso se utilizó el mismo tokenizador, proveniente de la librería *Fastai*, para que el vocabulario resultante fuera el mismo y los modelos pudieran ser comparables.

A pesar de esto, debido a las limitaciones de ambas implementaciones, hubo ciertas diferencias en la configuración:

- Mientras que el *learning rate* del modelo base se mantuvo en su valor por defecto (30), en el caso del modelo de Fastai se decidió ejecutar con un *learning rate* de 5e-3 (valor recomendado en el ejemplo de uso de la herramienta). Valores mucho más altos, cercanos al *learning rate* de la implementación base, no concluían en resultados satisfactorios, ni siquiera comparables a los del trabajo original.

- Por otro lado, mientras que el modelo base se entrenó utilizando como optimizador el algoritmo de NTASGD (ver Merity et al. (2017)), el modelo de *Fastai* se entrenó utilizando un optimizador ADAM, ya que la librería no presenta dicho optimizador.

Como se puede observar en la Figura 6.3, con el correr de las épocas, no solo la *perfor-*

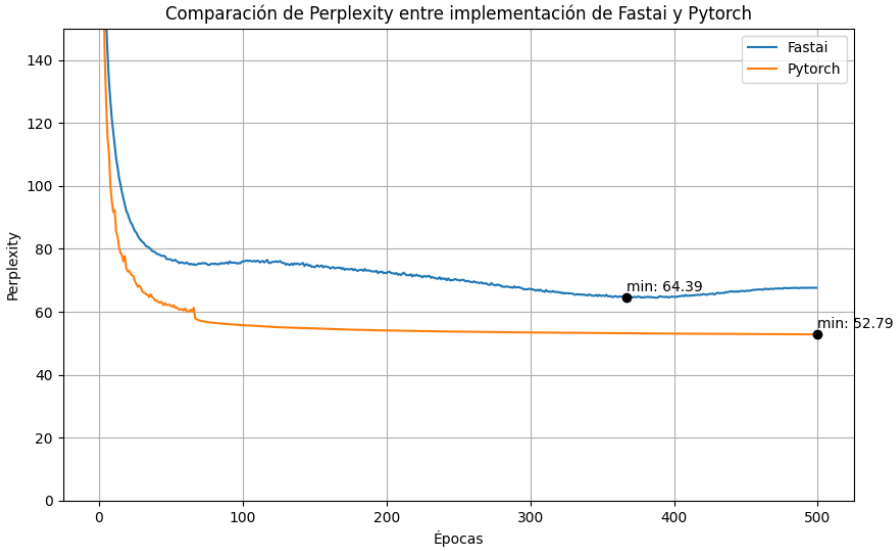


Fig. 6.3: Comparación de la capacidad predictiva de dos modelos generados a partir de una arquitectura AWD-LSTM a lo largo de las épocas. Uno de ellos fue generado haciendo uso de la librería *Fastai* mientras que el otro fue originado a partir de nuestra implementación base en *Pytorch*.

mance del modelo base generado en *PyTorch* se asemeja al de su contraparte implementado con la librería *Fastai*, sino que también se denota una mejora en la *performance* frente a su contraparte a lo largo de las épocas.

A partir de esto, se concluyó que la implementación base de la AWD-LSTM genera resultados parecidos al de la implementación generada a partir de herramientas más consolidadas como las presentadas por la librería *Fastai*, pudiendo constatar a esta implementación como un punto de partida válido para nuestras experimentaciones.

6.4. Generación del modelo base

Ya contando con nuestra implementación de la AWD-LSTM adaptada a las necesidades de este trabajo, para poder cumplir con nuestro objetivo de analizar la performance de la arquitectura luego del reentrenamiento con datos provenientes de movimientos oculares fue necesario contar con un modelo base. Para poder obtener el mismo, entrenamos la LSTM con el corpus *all_wikis* (rejunte de textos provenientes de *Wiki-media*) presentado previamente.

6.4.1. Comparación con Word2Vec

Como primer acercamiento para hallar nuestro modelo base, se decidió generar un modelo utilizando toda la información brindada por el corpus, sin variar ningún otro hiperparámetro en particular. El entrenamiento del mismo se realizó durante 8 épocas y se extrajeron los *embeddings* resultantes de la capa correspondiente en la red neuronal.

Luego, se comparó la similitud entre los *embeddings* con la información de los juicios de Multi-Simlex al igual que en el caso de Word2Vec, en un principio con la hipótesis de poder observar una similitud mayor a los juicios humanos en comparación al modelo

previo, ya que estamos hablando de una arquitectura posterior en el tiempo.

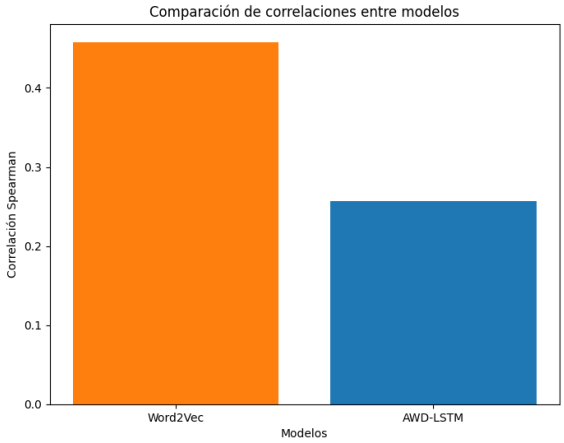


Fig. 6.4: Comparación de correlaciones con Multi-Simlex entre el modelo base generado a partir de Word2Vec y un modelo construido sobre la arquitectura AWD-LSTM, utilizando el 100 % del corpus de entrenamiento.

Se puede evidenciar que, al contrario de lo que se suponía, el modelo generado a partir de la AWD-LSTM presenta una correlación peor $(0,257)$ con los *embeddings* humanos que su contraparte generada con Word2Vec $(0,458)$, al menos para el caso de Multi-Simlex (Figura 6.4).

A partir de lo visto, este resultado evidencia (aún de manera no concluyente) que nuestra hipótesis inicial sobre la mejor capacidad de captar similitudes entre pares de palabras del modelo AWD-LSTM con respecto a Word2Vec podría ser errónea. Sin embargo, intuimos a su vez, que será necesario generar otros modelos de AWD-LSTM para tener resultados definitivos.

6.4.2. Exploración de hiperparámetros

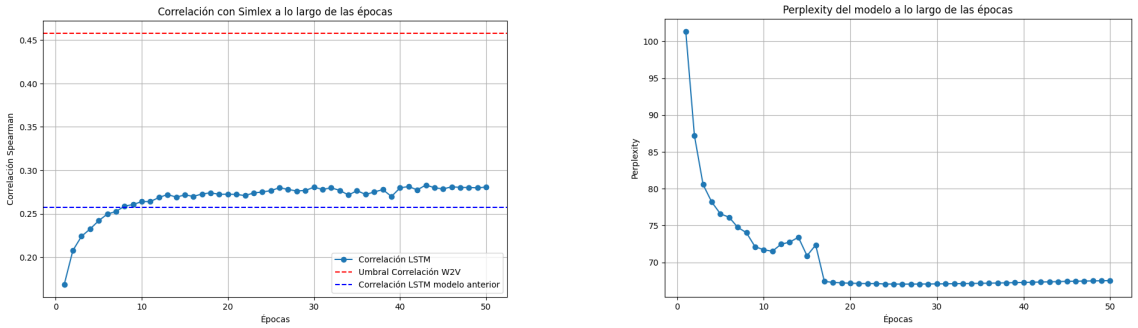
6.4.2.1 Tamaño del Corpus y Épocas

En vista de los resultados del primer modelo, se tomó la decisión de variar los hiperparámetros ligeramente. El primero de ellos que nos interesó variar fue la cantidad de épocas con las cuales se entrena al mismo. Sin embargo, esto no nos pareció viable debido al gran tamaño del corpus utilizado, ya que esto implicaba un consumo de tiempo muy grande en el entrenamiento del modelo, de alrededor de 12 horas cada época con un tamaño de batch de 30. Por lo tanto, se optó por también reducir el porcentaje del corpus empleado, permitiendo así aumentar la cantidad de épocas.

En consecuencia, para el siguiente modelo se decidió permanecer con los mismos hiperparámetros, únicamente aumentando la cantidad de épocas de 8 a 50 y disminuyendo el porcentaje del corpus utilizado de 100 % a 10 %, con el objetivo de observar si la similitud con Multi-Simlex aumenta o se mantiene en un número comparable con el modelo anterior.

Adicionalmente, para este experimento calculamos la correlación con los *embeddings* humanos al correr de las épocas, junto con la *perplexity*, para ver de qué manera estas evolucionan a lo largo del tiempo.

Dentro del gráfico que mide correlación con Multi-Simlex (Figura 6.5A) podemos observar que, a pesar de que no se llega a pasar el umbral generado por los resultados con



(A) *(B)*

Fig. 6.5: En A, correlación con Multi-Simlex a lo largo de las épocas para el modelo con una menor porcentaje de corpus de entrenamiento pero una mayor cantidad de épocas entrenadas. Se grafica consigo la correlación de los modelos de la Figura 6.4 para facilitar la comparación. En B, se grafica la *perplexity* del mismo modelo con el correr de las épocas.

Word2Vec, la correlación a lo largo de las épocas termina superando ligeramente aquella del primer modelo AWD-LSTM presentado, finalizando el entrenamiento con una correlación de $0,281$.

Por otro lado, la *perplexity* a lo largo de las épocas (Figura 6.5B) evidencia una reducción constante de la misma, mostrando que el modelo está mejorando su capacidad predictiva a lo largo de las épocas (al menos hasta la época 17, donde el modelo parece estancarse).

En síntesis, observamos que reducir el tamaño del corpus y aumentar la cantidad de épocas mejoró ligeramente la correlación entre los *embeddings*, sin llegar aún al umbral estipulado por Word2Vec. Sin embargo, nos permite evidenciar que aumentar el tamaño del corpus no es proporcional a la correlación con los juicios de similitud humanos, por lo que se tomó la decisión de a partir de ahora seguir insistiendo con un tamaño de corpus del $10\,\%$ mientras se prueban otras opciones.

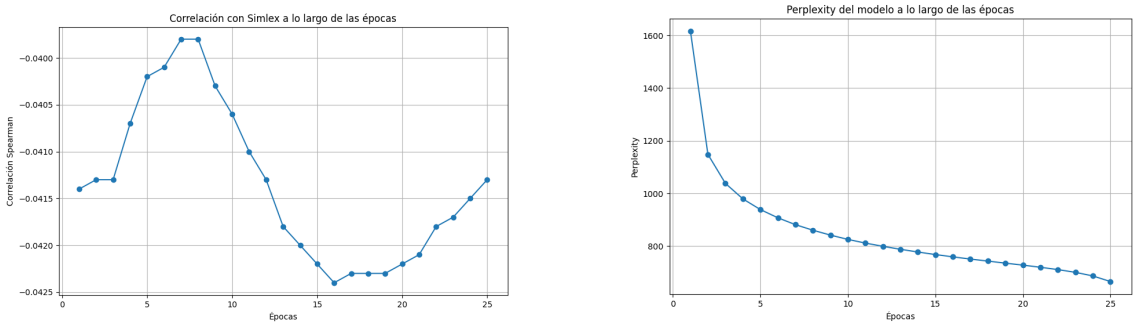
6.4.2.2 Learning Rate

Para el siguiente modelo, valiéndose de los resultados anteriores, se eligió reducir el *learning rate* que se aplica en el entrenamiento de 30 a un número menor (3e-3), con el objetivo de intentar mejorar el aprendizaje del modelo, potencialmente evitando el estancamiento de su *perplexity*. En este caso, se decidió entrenar el modelo solamente con 25 épocas.

En lo que a *perplexity* del modelo se refiere (Figura 6.6B), se puede evidenciar que con un *learning rate* menor el modelo no llega a estancarse. Sin embargo, con la cantidad de épocas dada no se llega a los niveles de *perplexity* obtenidos en el modelo anterior, ya que se observa una mejora lenta de la misma.

Por otro lado, los resultados de la correlación con Multi-Simlex (Figura 6.6A) no parecen mostrar resultados comparables a los anteriores modelos, posiblemente debido al poco aprendizaje realizado por el modelo durante el entrenamiento.

Luego de contemplar los últimos modelos, se puede notar que un *learning rate* como el que se utiliza en la LSTM originalmente genera cierto estancamiento en la *performance* del modelo, incluso utilizando el algoritmo inherente de la AWD-LSTM que varía el *learning*



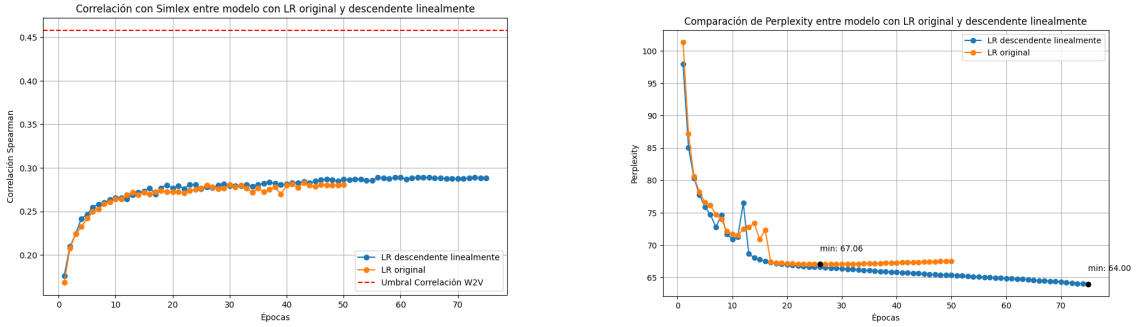
(A) *(B)*

Fig. 6.6: En A, correlación del modelo entrenado con un *learning rate* reducido con Multi-Simlex. Por otro lado, en B se grafica la capacidad predictiva del modelo durante el entrenamiento utilizando la *perplexity* como métrica.

rate de manera aleatoria con el pasar de las épocas. Por otro lado, si se reduce este hiperparámetro a un número más estándar dentro de la academia, el modelo no se llega a estancar, pero su *performance* no es comparable a la de sus predecesores. Con esto en mente, se pensó en modificar la forma en la que varía el *learning rate* con el fin de obtener lo mejor de los dos mundos.

6.4.2.3 Variación del Learning Rate

A partir de lo mencionado anteriormente, se propuso variar la manera en la que se elige el *learning rate* a lo largo de las épocas dentro de la AWD-LSTM. En particular, se tomó la decisión de partir del *learning rate* original planteado por la arquitectura e ir descendiendo el mismo de manera lineal hasta llegar al utilizado en el experimento anterior, buscando que el modelo no se estanque en su capacidad predictiva y sea capaz de generar mejores resultados que en los casos anteriores. Adicionalmente, se aumentó la cantidad de épocas a 75 para poder tener más evidencia del estancamiento de la *perplexity* del modelo pasadas las épocas. Se compararon los resultados de este modelo con aquellos del modelo que utiliza el *learning rate* original (30) para poder observar ambas cuestiones.



(A) *(B)*

Fig. 6.7: Comparación de la correlación con Multi-Simlex a lo largo del entrenamiento entre el modelo de AWD-LSTM presentado en la Figura 6.5 y un modelo el cual presenta un *learning rate* el cual desciende linealmente a lo largo de las épocas. En B, se compara la *perplexity* entre ambos modelos.

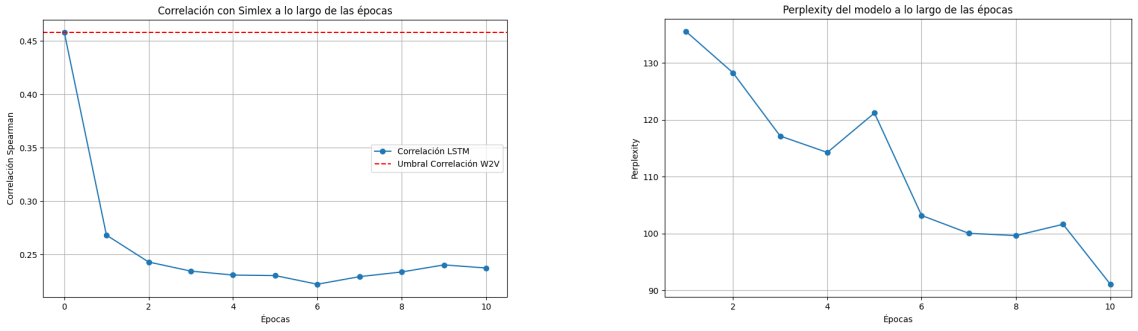
Como se puede observar en la Figura 6.7B, mientras que el modelo con el *learning rate* original se estanca a partir de una cierta época, el nuevo modelo con el *learning rate* descendiente linealmente no solo supera al modelo anterior en *perplexity* dentro de la marca de las 50 épocas, sino que también (al continuar el entrenamiento del modelo) sigue descendiendo lentamente gracias a la implementación de un *learning rate* más bajo.

Por otro lado, la correlación con Multi-Simlex del modelo nuevo parece ser idéntica a la del modelo anterior, sin presentar mejoras sustanciales. Se puede observar una cierta mejora en la correlación en las últimas épocas, pero esto se debe a que el modelo nuevo fue entrenado por una cantidad de épocas mayor que el anterior. Incluso al compararlo con el umbral obtenido por el modelo de Word2Vec, los resultados de los modelos quedan por debajo.

Luego de variar varios hiperparámetros e incluso cambiar la forma en la que se elige el *learning rate* del modelo a lo largo de las épocas, no se logró igualar los niveles de correlación generados por los *embeddings* de Word2Vec. En vista de esto, hipotetizamos que probablemente la diferencia entre ambos modelos no sea un tema de los hiperparámetros y su entrenamiento en sí, sino más bien que los *embeddings* resultantes de la AWD-LSTM son ciertamente distintos a los resultantes del modelo Word2Vec, generando resultados disímiles. También es una posibilidad que los valores iniciales de la capa de *embeddings* de la AWD-LSTM no ayuden al modelo a converger a los resultados esperados.

6.4.3. Embeddings preentrenados

Para poder poner en evidencia esto último, se optó por realizar un experimento adicional previo a la elección del modelo base. Uno de los métodos para ayudar a un modelo de lenguaje a mejorar su capacidad predictiva durante el entrenamiento es aquella de partir con *embeddings* preentrenados por otro modelo y dejar que el modelo actual ajuste estos durante el entrenamiento (C. Wang et al., 2020). A partir de esto, se decidió entrenar un modelo idéntico al modelo anterior durante 10 épocas, con la salvedad de que esta vez se lo entrenó inicializando la capa de *embeddings* de la red neuronal con los *embeddings* provenientes del modelo de Word2Vec.



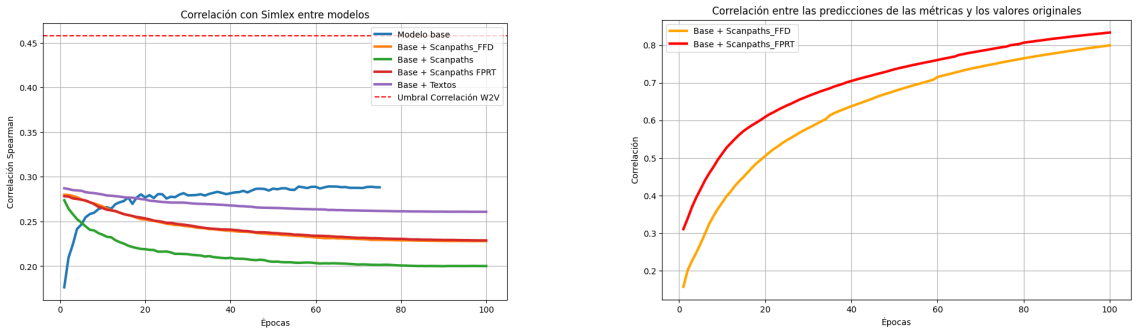
(A) *(B)*

Fig. 6.8: En A, se reporta la correlación con Multi-Simlex a lo largo de las épocas del modelo de AWD-LSTM el cual parte de embeddings preentrenados, generados a partir del modelo base de Word2Vec. Nótese que debido a esto, previo al entrenamiento, la correlación con Multi-Simlex es igual que en el modelo de Word2Vec. Por otro lado en B, se reporta la *perplexity* del modelo.

Tal y como se puede observar en la Figura 6.8A, la correlación de los *embeddings* disminuye drásticamente luego de finalizada la primera época de entrenamiento, para luego ir disminuyendo lentamente hasta llegar a la época 6, donde la misma empieza a mejorar levemente. Más aún, también se puede observar que la capacidad predictiva del modelo va mejorando con el correr de las épocas.

A partir de esta información, se concluye que efectivamente los *embeddings* extraídos del modelo resultante de la AWD-LSTM se comportan de una manera distinta a los de Word2Vec, evidenciado inicialmente en el brusco decrecimiento de la correlación con Multi-Simlex luego de terminada la primer época de entrenamiento. El modelo no parece estar aprovechando de ninguna manera los *embeddings* preentrenados, transformándolos a lo largo del correr de las épocas en *embeddings* parecidos a los generados por la AWD-LSTM. Por otro lado, al observar que la capacidad predictiva del modelo mejora a lo largo de las épocas (al contrario que la similitud con Multi-Simlex) nos da la pauta de que estos dos resultados no están correlacionados.

6.5. Reentrenamiento del modelo base



(A) *(B)*

Fig. 6.9: En A, comparación de la correlación con Multi-Simlex a lo largo de las épocas entre el modelo base elegido y los modelos reentrenados a partir del mismo. En B, para los modelos reentrenados con métricas de movimientos oculares, se reporta la correlación de las predicciones de las métricas generadas por el modelo con los valores originales a lo largo de las épocas.

Luego de realizadas las experimentaciones pertinentes para lograr hallar un buen modelo base sobre el cual reentrenar con información cognitiva, se decidió contar con el modelo entrenado a partir de un learning rate linealmente descendente. A partir de esto, se decidió reentrenar un nuevo modelo utilizándolo como base para cada uno de los textos generados a partir de los experimentos con movimientos oculares. Estos nuevos modelos fueron reentrenados durante 100 épocas con el 100 % de estos textos, buscando analizar su correspondencia con los juicios de similitud humanos utilizados en este trabajo, además de compararlos tanto con el modelo base, como con los resultados obtenidos en otros modelos, como Word2Vec. En particular, se hicieron comparaciones tanto con Multi-Simlex como con SWOW-RP. Para validar el correcto aprendizaje de la información cognitiva, también se graficó la correlación de *Spearman* entre las predicciones realizadas por el modelo sobre las métricas de movimientos oculares para cada época, con el objetivo de ver si los modelos lograban aprender a predecir estas métricas a lo largo del entrenamiento.

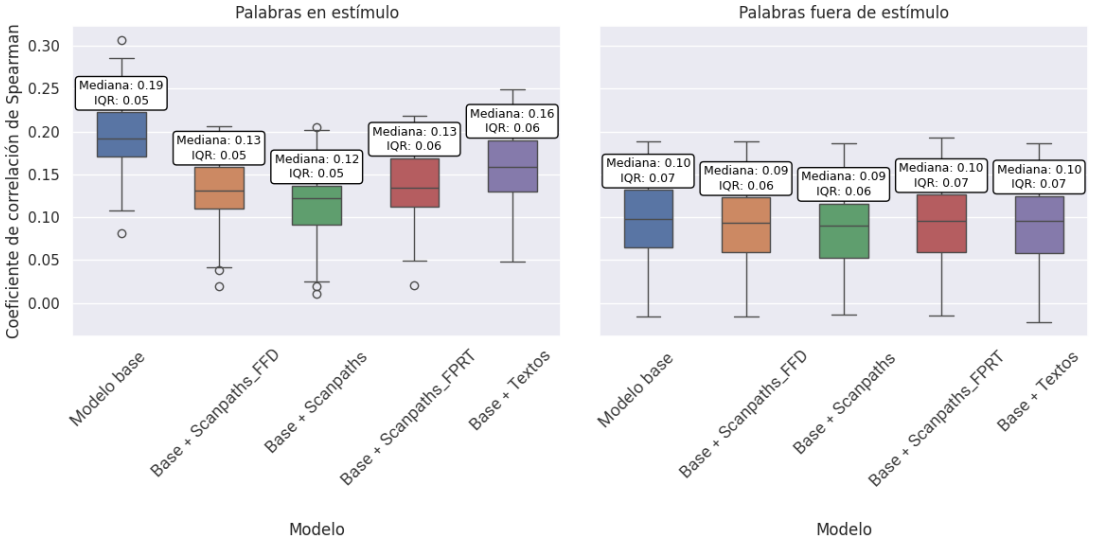


Fig. 6.10: Comparación de la correlación entre el modelo base y los modelos reentrenados a partir de él con SWOW-RP. Para más rigurosidad estadística, se realiza un muestreo de 1000 pares de palabras 100 veces con reposición en vez de comparar todos los pares de palabras una sola vez.

A partir de la Figura 6.9A podemos observar que la correlación con Multi-Simlex en los modelos reentrenados va disminuyendo paulatinamente a lo largo de las épocas, indicando que los *embeddings* generados se van alejando en similitud con los datos recabados sobre juicios de similitud humanos. Sin embargo, también se puede denotar una diferencia positiva cuando se comparan los modelos reentrenados con información cognitiva con el modelo reentrenado sin información cognitiva. Por otro lado, en la Figura 6.9B, se puede observar que la correlación entre las predicciones de la información cognitiva y sus valores originales van aumentando considerablemente con el correr de las épocas, dando indicios de que el modelo efectivamente está aprendiendo a predecirlas (y, por lo tanto, se integran sobre los *embeddings*).

Dentro de la correlación con SWOW-RP podemos hallar resultados parecidos (Figura 6.10), donde la correlación es ciertamente menor en los casos en donde el modelo es reentrenado, además de contar con una sutil mejora si a ese reentrenamiento se lo acompaña con la información relacionada a los experimentos sobre movimientos oculares. Otro resultado que se puede denotar de esto es que si prestamos atención a los resultados sobre SWOW-RP por parte de los modelos de Word2Vec (ver Sección 6.2), al igual que en el caso que con Multi-Simlex, los *embeddings* provenientes del mismo son más capaces de captar las similitudes entre pares de palabras puntuadas por los humanos que aquellos provenientes de la AWD-LSTM, aunque en el caso de SWOW-RP esta diferencia es menos notoria.

A partir de lo observado, se puede llegar a la conclusión que dentro de la arquitectura AWD-LSTM los embeddings reentrenados con los *scanpaths* empeoran la correlación sobre los juicios de similitud humanos utilizados en este trabajo. Sin embargo, se puede observar que la reducción de la similitud es menor cuando se incorporan a estos textos de reentrenamiento la información cognitiva, dándonos la pauta que quizás el problema no se encuentra en la información cognitiva sino en el formato del texto de reentrenamien-

to. Esta hipótesis se vuelve más sólida al observar que, durante el reentrenamiento, los *embeddings* efectivamente están aprendiendo a predecir las métricas relacionadas con los movimientos oculares, incorporando su información. Más aún, también se puede cuestionar la utilización de la similitud entre pares de palabras como la tarea correcta para evaluar a estos modelos, ya que su simpleza puede llegar a evitar que se puedan capturar estas mejoras obtenidas con la información cognitiva.

7. CONCLUSIONES

Luego de finalizados los experimentos y puestos en evidencia los resultados de los mismos, se decidió partir de dos ejes que se fueron encontrando a lo largo del trabajo para elaborar las conclusiones.

El primero de ellos es, al contrario de lo que se esperaba, la diferencia en los resultados de las similitudes con los juicios de valor a favor de la arquitectura Word2Vec frente a la de la arquitectura AWD-LSTM. Creemos que esto se debe principalmente a la estructura interna de estas y a la información que los *embeddings* de cada arquitectura albergan dentro de sus dimensiones. Mientras que el objetivo principal de un modelo como Word2Vec es devolver una serie de *embeddings*, el objetivo de un modelo de AWD-LSTM es aquel de predecir la siguiente palabra dado un contexto. Esto evidencia la necesidad de contar con información no sólo semántica de las palabras, sino también sintáctica y léxica, información que creemos se encuentra en parte dentro de los *embeddings* de la arquitectura, generando así resultados no tan positivos cuando se comparan los mismos con juicios de similitud netamente semánticos, como es el caso de Multi-Simlex. Por otro lado, se muestra que los *embeddings* generados por Word2Vec (*embeddings* cargados de información semántica) obtienen resultados considerablemente mejores cuando se comparan con los juicios de similitud humanos.

Esta hipótesis toma más fuerza al observar los resultados del entrenamiento con los *embeddings* preentrenados de Word2Vec, ya que desde la primer época se ve un descenso brusco de la correlación de los *embeddings* con Multi-Simlex, el cual creemos que se debe a que la red neuronal está forzando a los pesos de los *embeddings* a cambiar hacia unos *embeddings* con información tanto sintáctica como semántica de las palabras dado el contexto de entrenamiento.

Por otro lado, el otro eje, ya haciendo hincapié en la arquitectura AWD-LSTM, es, sorpresivamente, la disminución de la similitud con los juicios de valor al reentrenar el modelo con textos de los cuentos acompañados con información cognitiva. Creemos que esto se debe principalmente a la estructura de los *scanpaths*, los cuales al respetar la forma en la que se leyeron los textos, presentando saltos de palabras, repeticiones, regresiones, etc, la sintaxis del texto utilizado para reentrenar no es sobre lo que fue entrenado previamente el modelo, lo que intuimos que empeora los *embeddings* a la hora de compararlos.

Esto se evidencia más al observar que el modelo reentrenado con los textos planos es el que mejor similitud presenta frente al resto. De todas formas, ninguno de los modelos reentrenados logra superar al modelo base, creemos principalmente debido a que la información presente en los cuentos es demasiado específica al compararla con el gran corpus de texto con el cual fue entrenado el mismo.

No obstante, también se puede observar que los modelos reentrenados con las métricas de movimientos oculares aprenden información sobre las mismas, lo cual consideramos que justifica la mejora de la similitud con los juicios de valor de los modelos reentrenados con los scanpaths en comparación a su reentrenamiento sin dicha información.

Por lo tanto, vemos que existen cabos sueltos que podrían llegar a servir de puntapié para trabajos futuros, como por ejemplo:

- Cambiar la forma en la cual se incorpora la información cognitiva al modelo, conca-

tenando las métricas (Hollenstein & Zhang, 2019) en lugar de predecirlas, condicionando la red neuronal.

- Reentrenar el modelo base con otro formato de texto, el cual no conlleve consigo la pérdida de sintaxis de los cuentos y que además permita la incorporación de las métricas.
- Evaluar estos embeddings con otros métodos más extrínsecos, como podría ser tareas más relacionadas con el pensamiento humano o subjetividad humana, como análisis de sentimientos o comprensión lectora. Incluso se podrían probar otros métodos de evaluación de la misma índole que los juicios de similitud, como la utilización de analogías de palabras (B. Wang et al., 2019).

BIBLIOGRAFíA

Altarriba, J., Kroll, J., Sholl, A., & Rayner, K. (1996). The influence of lexical and conceptual constraints on reading mixed-language sentences: Evidence from eye fixations and naming times. *Memory and Cognition*, *24*(4), 477-492.

Barrett, M., Bingel, J., Keller, F., & Søgaard, A. (2016). Weakly Supervised Part-of-speech Tagging Using Eye-tracking Data. En K. Erk & N. A. Smith (Eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (pp. 579-584). Association for Computational Linguistics.

Bianchi, B., Bengolea Monzón, G., Ferrer, L., Fernández Slezak, D., Shalom, D. E., & Kamienkowski, J. E. (2020). Human and computer estimations of Predictability of words in written language. *Scientific Reports*, *10*(1), 4396.

Brown, A. C. (1895). *The Relation between the Movements of the Eyes and the Movements of the Head: Being the Fourth Robert Boyle Lecture Delivered before the Oxford University Junior Scientific Club*. Henry Frowde.

Cabana, Á., Zugarramurdi, C., Valle-Lisboa, J., et al. (2023). The "Small World of Words" free association norms for Rioplatense Spanish. *Behavior Research*, *56*, 968-985.

Caruana, R. (1997). Multitask Learning. *Machine Learning*, *28*.

Castelhano, M. S., & Rayner, K. (2008). Eye movements during reading, visual search, and scene perception: An overview. En K. Rayner, D. Shen, X. Bai & G. Yan (Eds.), *Cognitive and Cultural Influences on Eye Movements* (pp. 3-33). Tianjin People's Publishing House.

Chandrasekaran, D., & Mago, V. (2021). Comparative analysis of word embeddings in assessing semantic similarity of complex sentences.

Chiruzzo, L., Etcheverry, M., & Rosa, A. (2020). Sentiment analysis in Spanish tweets: Some experiments with focus on neutral tweets.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.

Clifton, C., Staub, A., & Rayner, K. (2007). Eye movements in reading words and sentences. En R. van Gompel, M. H. Fischer, W. S. Murray & R. L. Hill (Eds.), *Eye Movements: A Window on Mind and Brain* (pp. 341-372). Elsevier.

Ehrlich, K., & Rayner, K. (1983). Pronoun assignment and semantic integration during reading: Eye movements and immediacy of processing. *Journal of Verbal Learning and Verbal Behavior*, *22*(1), 75-87.

Frazier, L., & Rayner, K. (1982). Making and correcting errors during sentence comprehension: Eye movements in the analysis of structurally ambiguous sentences. *Cognitive Psychology*, *14*, 178-210.

Gladkova, A., & Drozd, A. (2016). Intrinsic Evaluations of Word Embeddings: What Can We Do Better?

Hellrich, J., & Hahn, U. (2017). Don't get fooled by word embeddings: better watch their neighborhood.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation*, *9*, 1735-80.

Hofmann, M., Biemann, C., & Remus, S. (2017, marzo). Benchmarking n-grams, Topic Models and Recurrent Neural Networks by Cloze Completions, EEGs and Eye Movements.

Hollenstein, N., & Zhang, C. (2019, febrero). Entity Recognition at First Sight: Improving NER with Eye Movement Information.

Immordino-Yang, M. H., & Deacon, T. W. (2007). An evolutionary perspective on reading and reading disorders. En K. W. Fischer, J. H. Bernstein & M. H. Immordino-Yang (Eds.), *Mind, brain, and education in reading disorders* (pp. 16-36). Cambridge University Press.

Inhoff, A. (1984). Two stages of word processing during eye fixations in the reading of prose. *Journal of Verbal Learning and Verbal Behavior*, *23*, 612-624.

Irwin, D. E. (1998). Lexical processing during saccadic eye movements. *Cognitive Psychology*, *36*(1), 1-27.

Irwin, D. E., & Carlson-Radvansky, L. A. (1996). Cognitive suppression during saccadic eye movements. *Psychological Science*, *7*(2), 83-88.

Jurafsky, D., & Martin, J. H. (2000). *Speech and Language Processing* (1st). Prentice Hall.

Just, M. A., & Carpenter, P. A. (1980). A theory of reading: From eye fixations to comprehension. *Psychological Review*, *87*(4), 329-354.

Klerke, S., Goldberg, Y., & Søgaard, A. (2016). Improving sentence compression by learning to predict gaze.

Kliegl, R., Nuthmann, A., & Engbert, R. (2006). Tracking the mind during reading: The influence of past, present, and future words on fixation durations. *Journal of experimental psychology. General*, *135*, 12-35.

Levy, O., & Goldberg, Y. (2014). Dependency-Based Word Embeddings.

Liu, Q., Kusner, M. J., & Blunsom, P. (2020). A Survey on Contextual Embeddings.

Merity, S., Keskar, N. S., & Socher, R. (2017). Regularizing and Optimizing LSTM Language Models.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality.

Mishra, A., Kanojia, D., Nagar, S., Dey, K., & Bhattacharyya, P. (2017, enero). Leveraging Cognitive Features for Sentiment Analysis.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training Recurrent Neural Networks.

Paterson, K., Liversedge, S., & Underwood, G. (1999). The influence of focus operators on syntactic processing of short relative clause sentences. *Quarterly Journal of Experimental Psychology*, *52A*, 717-737.

Rayner, K. (1998). Eye movements in reading and information processing: 20 years of research. *Psychological Bulletin*, *124*(3), 372-422.

Rayner, K. (2009). Eye movements and attention in reading, scene perception, and visual search. *Quarterly Journal of Experimental Psychology*, *62*(8), 1457-1506.

Rayner, K., Pollatsek, A., Ashby, J., & Clifton, C. (2012). *The Psychology of Reading (2nd edition)*. Psychology Press.

Renandya, W. (2007). The Power of Extensive Reading. *Relc Journal*, *38*, 133-149.

Sajjad, H., Guzmán, F., Durrani, N., Abdelali, A., Bouamor, H., Temnikova, I., & Vogel, S. (2016). Eyes Don't Lie: Predicting Machine Translation Quality Using Eye

Movement. En K. Knight, A. Nenkova & O. Rambow (Eds.), *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 1082-1088). Association for Computational Linguistics.

Staub, A., & Rayner, K. (2007). Eye movements and on-line comprehension processes. En M. G. Gaskell (Ed.), *The Oxford Handbook of Psycholinguistics* (pp. 327-342). Oxford University Press.

Vulić, I., Baker, S., Ponti, E. M., Petti, U., Leviant, I., Wing, K., Majewska, O., Bar, E., Malone, M., Poibeau, T., Reichart, R., & Korhonen, A. (2020). Multi-SimLex: A Large-Scale Evaluation of Multilingual and Cross-Lingual Lexical Semantic Similarity. *Scientific Reports*, *10*(1), 4396.

Wang, B., Wang, A., Chen, F., Wang, Y., & Kuo, C.-C. J. (2019). Evaluating word embedding models: methods and experimental results. *APSIPA Transactions on Signal and Information Processing*, *8*(1).

Wang, C., Nulty, P., & Lillis, D. (2020, diciembre). A Comparative Study on Word Embeddings in Deep Learning for Text Classification.

Wang, Z. (2018). N-gram and LSTM based Language Models.

Wilkinson, D. (2012). A Data-Driven Approach to Increasing Student Motivation in the Reading Classroom. *Language Education in Asia*, *3*, 252-262.

Yaghoobzadeh, Y., & Schütze, H. (2016). Intrinsic Subspace Evaluation of Word Embedding Representations.

Yang, S., Yu, X., & Zhou, Y. (2020, junio). LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example.

ANEXO

A. Hiperparámetros del modelo

Hiperparámetro	Descripción	Valor por defecto
layer_num	Cantidad de celdas LSTM que presenta la red neuronal	3
embed_size	Tamaño del embedding resultante de la capa de embeddings	300
hidden_size	Dimensión del resultado de la capa oculta de la celda LSTM, lo que se conoce como h_t	1150
w_drop	Weight drop correspondiente al interior de la capa de las celdas LSTM	0.5
dropout_i	Dropout que se aplica a los vectores de palabras resultantes de la capa de embeddings	0.4
dropout_l	Dropout que se aplica entre medio de las conexiones entre una celda LSTM y otra	0.3
dropout_o	Dropout que se aplica al resultado de la última capa LSTM	0.4
dropout_e	Dropout que se aplica a la capa de embeddings	0.1
winit	Valor de inicialización de los pesos de la capa de embeddings	0.1
batch_size	Tamaño del batch utilizado para entrenar	30
valid_batch_size	Tamaño del batch utilizado para medir performance en el corpus de validación	10
bptt	Tamaño de la secuencia* utilizado en el algoritmo de backpropagation through time	70
ar	Parámetro utilizado en el algoritmo de activation regularization	2
tar	Parámetro utilizado en el algoritmo de temporary activation regularization	1

Hiperparámetro	**Descripción**	**Valor por defecto**
weight_decay	Constante utilizada en la regularización L2	1.2e-6
epochs	Cantidad de épocas	500
lr	Learning rate	30
max_grad_norm	Threshold utilizado en el algoritmo de gradient clipping	0.25
non_mono	Tamaño de la ventana que se utiliza para generar el estado de no monotonicidad	5

*El tamaño de la secuencia es variable de acuerdo a la época, pero utiliza este valor como base. Para ver mas, consulte en 1.1.2.3

B. Experimentos movimientos oculares

Cuento	Autor	Palabras	Fijaciones	Fijaciones Excluidas	Regresiones	Saltos de palabra
La noche de los feos	Mario Benedetti	544	25774	10290	8046	11234
Cómo funcionan los bolsillos	Valentín Muro	972	45815	11677	16176	19705
La máscara de la Muerte Roja	Edgar Allan Poe	572	26641	6805	9092	11974
Las fotografías	Silvina Ocampo	618	26686	8034	8580	12636
La salud de los enfermos	Julio Cortázar	667	34486	7596	12189	17953
Buenos Aires	Hernán Casciari	607	28813	6855	10368	12932
Wakefield	Nathaniel Hawthorne	693	31610	9034	10467	17397
Cómo funciona caminar en la nieve	Valentín Muro	1066	47302	10650	16245	20937
Ahora debería reírme, si no estuviera muerto	Angela Carter	606	25629	7124	7022	15558
El espejo	Haruki Murakami	628	29851	9597	9170	16788
Embarrar la magia	Facundo Alvarez Heduan	683	34749	12290	12143	14400

Cuento	Autor	Palabras	Fijaciones	Fijaciones Excluidas	Regresiones	Saltos de palabra
La lluvia de fuego	Leopoldo Lugones	640	30960	9236	10121	15979
Educar para escalar y bucear	Andrés Rieznik	599	27797	7472	9500	12621
El golpe de gracia	Ambrose Bierce	602	27629	7567	9540	14387
La gallina degollada	Horacio Quiroga	659	30188	8958	9825	15769
Rubí y el lago danzante	Marcelo Cohen	641	30216	8112	9332	15857
La canción que cantábamos todos los días	Luciano Lamberti	620	28299	7247	8418	15386
El almohadón de plumas	Horacio Quiroga	579	28063	9453	8301	15087
Una rosa para Emilia	William Faulkner	643	33946	8968	12007	16178
La de la Obsesión por la Patineta	Hernán Casciari	579	29200	8516	10044	13171
Total	-	**13218**	**623654**	**175481**	**206586**	**305949**