Índice

1. Introducción

El análisis de sentimiento, también conocido como minería de opinión, es el uso de procesamiento de lenguaje natural (NLP por sus siglas en inglés), análisis de texto y lingüística computacional con el objetivo de identificar, extraer, cuantificar y estudiar estados afectivos e información subjetiva.

Algunos casos de uso del análisis de sentimiento son:

- Análisis de redes sociales respecto de tópicos que interesen a un usuario en particular.
- Determinar el sentimiento de clientes en encuestas de satisfacción.
- Predicción de fluctuaciones en el precio alguna acción basándose en el sentimiento percibido sobre una empresa asociada a dicha acción en redes sociales como *Twitter*.

El presente documento describe el avance en investigación de un proyecto en curso para implementar un modelo de redes neuronales recurrentes y utilizar *word embeddings* para predecir la polaridad de *tweets* en español utilizando tres categorías: positivo, negativo o neutro.

Los datos que utilizaremos son los vectores pre–entrenados con el modelo GloVe en (Etcheverry & Wonsever, 2016) y los datos anotados de (TASS: Workshop on Semantic Analysis, 2020) para entrenar la red neuronal.

La mayor parte de la presentación de este material sigue de cerca los libros de (Zhang et al., 2021) y (Goodfellow et al., 2016).

2. Marco Teórico

En esta sección hacemos un repaso de la teoría necesaria para entender la arquitectura final que utilizaremos para hacer la predicción de polaridad.

2.1. *Recurrent Neural Networks*

Las redes neuronales recurrentes —RNNs por sus siglas en inglés— son una clase relativamente basta de arquitecturas de redes neuronales que son utilizadas principalmente para resolver tareas que involucran el uso de datos secuenciales, como por ejemplo: predicción de textos o precios de acciones en el tiempo.

Las RNNs que introduciremos en esta sección son conocidas como redes recurrentes *unidireccionales*, que nos servirán para explicar la arquitectura general de las redes neuronales recurrentes.

Un problema popular que motiva el uso de RNNs unidireccionales es el de estimar el paso x_t de una secuencia $\{x_i\}_{i\in\mathbb{Z}^+}$ dados $x_1,\ldots,x_{t-1}$; es decir, que intentamos estimar la distribución que nos permita hallar x_t tal que dicha variable aleatoria se distribuya según:

$$x_t \sim P(x_t \mid x_{t-1},\ldots,x_1). \tag{1}$$

Aproximaremos la anterior estimación del siguiente modo:

$$P(x_t \mid x_{t-1},\ldots,x_1) \approx P(x_t \mid h_{t-1}), \tag{2}$$

donde h_{t-1} es un *estado escondido* que servirá para codificar la información de la secuencia hasta el tiempo $t-1$. En general, el estado escondido en cualquier tiempo t puede computarse basados en la entrada actual x_t y el estado escondido anterior h_{t-1}:

$$h_t = f(x_t, h_{t-1}),$$

donde f es una función que podemos aprender.

Así, asumamos que tenemos un minilote de ejemplos $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ al tiempo t; es decir, que tomamos un conjunto de n subsecuencias con d pasos cada una. Denotamos por $\mathbf{H}_t \in \mathbb{R}^{n\times h}$ a la variable escondida al tiempo t.

Calcularemos el estado escondido al tiempo t del siguiente modo:

$$\mathbf{H}_t = \phi(\mathbf{X}_t\mathbf{W}_{xh} + \mathbf{H}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h), \tag{3}$$

donde $\mathbf{W}_{xh} \in \mathbb{R}^{d\times h}$ y $\mathbf{W}_{hh} \in \mathbb{R}^{h\times h}$ son matrices de pesos, $\mathbf{b}_h \in \mathbb{R}^{1\times h}$ es un arreglo de sesgo y ϕ representa la función de activación.

Las RNNs toman su nombre de esta estructura recursiva del cálculo de los estados escondidos que codifican la información pasada de la secuencia. Las capas que realizan el anterior cálculo se conocen como *capas recurrentes*.

La capa de salida que nos permitirá comparar las predicciones de la red con el valor real en la secuencia está dada por el siguiente cálculo:

$$\mathbf{O} = \mathbf{H}_t\mathbf{W}_{hq} + \mathbf{b}_q, \tag{4}$$

con $\mathbf{W}_{hq} \in \mathbb{R}^{h\times q}$ y $\mathbf{b}_q \in \mathbb{R}^{1\times q}$.

Una visualización de esta arquitectura puede observarse en la figura 1.

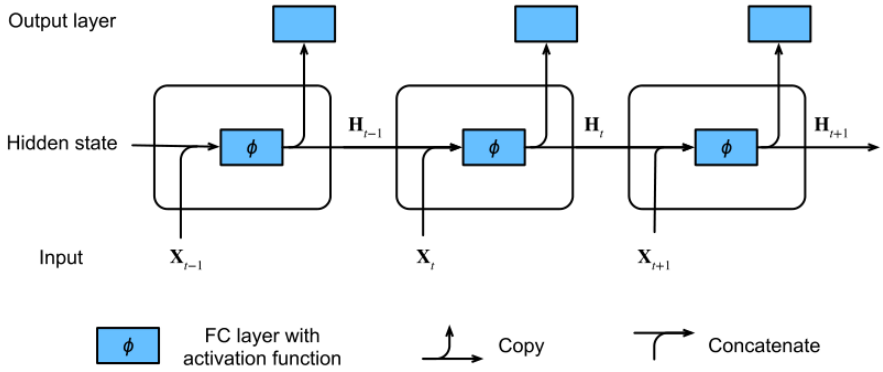

Figura 1: Una RNN con un estado escondido.

2.1.1. *Gated Recurrent Units*

Uno de los principales desafíos en la predicción de datos secuenciales es el de lograr preservar información de largo plazo al tiempo de ser capaces de ignorar entradas en el corto plazo que sean irrelevantes o que aporten poca información.

Estas consideraciones son de particular importancia cuando hallamos secuencias que tienen saltos estructurales que deseamos aprender. Por enunciar un ejemplo, consideremos lo que sucede cuando pasamos en un texto de un capítulo a otro. La ruptura lógica en el contexto de la secuencia es un factor que deseamos que nuestras redes neuronales sean capaces de capturar.

Uno de los primeros intentos para enfrentar este problema fue el diseño de las redes recurrentes conocidas como *Long–Short Term Memory* (LSTM) (Hochreiter & Schmidhuber, 1997). En esta sección describimos una arquitectura más moderna que pretende resolver esos mismos problemas, consiguiendo resultados comparables (Chung et al., 2014).

Las *Gated Recurrent Units* —GRU por sus siglas en inglés— son un tipo de RNN que toman aparente inspiración de las puertas lógicas de circuitos, al incorporar unidades escondidas que funcionan de manera aproximada a puertas, y que permiten a la red neuronal controlar cuánta información del estado actual $\mathbf{X}_t$ se pasa al siguiente estado y cuánta del estado escondido anterior $\mathbf{H}_{t-1}$ se retiene.

Las primeras unidades que introducimos son la *puerta de reinicio* y la *puerta de actualización*. La puerta de reinicio le permite controlar a la red cuánta información del anterior estado deseamos preservar. A su vez, la puerta de actualización nos permite controlar en qué medida el nuevo estado es una copia del anterior.

En términos matemáticos, para un tiempo t dado y un minilote $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ —con n ejemplos y d dimensiones por ejemplo– y un estado escondido del anterior paso $\mathbf{H}_{t-1} \in \mathbb{R}^{n\times h}$ —con h unidades escondidas—, la puertas de reinicio y actualización $\mathbf{R}_t \in \mathbb{R}^{n\times h}$, $\mathbf{Z}_t \in \mathbb{R}^{n\times h}$, respectivamente, están dadas por:

$$\mathbf{R}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xr} + \mathbf{H}_{t-1}\mathbf{W}_{hr} + \mathbf{b}_r), \tag{5}$$

$$\mathbf{Z}_t = \sigma(\mathbf{X}_t\mathbf{W}_{xz} + \mathbf{H}_{t-1}\mathbf{W}_{hz} + \mathbf{b}_z), \tag{6}$$

donde $\mathbf{W}_{xr}, \mathbf{W}_{xz} \in \mathbb{R}^{d\times h}$ y $\mathbf{W}_{hr}, \mathbf{W}_{hz} \in \mathbb{R}^{h\times h}$ son parámetros de pesos y $\mathbf{b}_r, \mathbf{b}_z \in \mathbb{R}^{1\times h}$ son

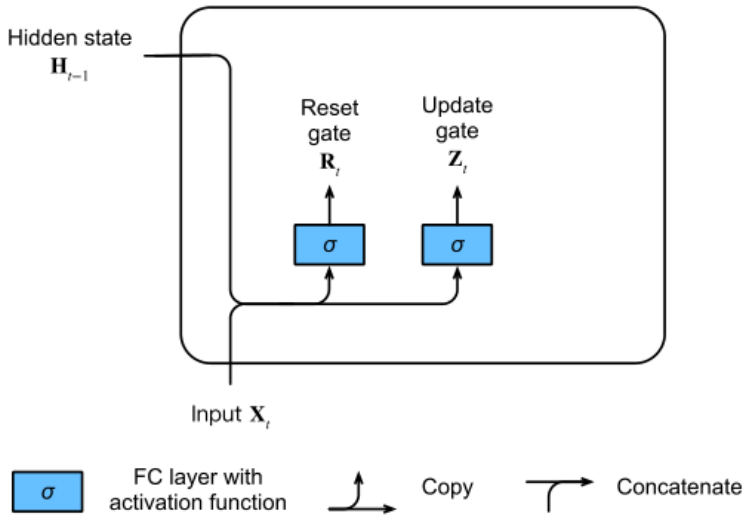

Figura 2: Cálculo de las puertas de reinicio y actualización en un modelo GRU.

vectores de sesgo. Refiérase a la figura 2.

Notemos que utilizamos funciones sigmoideas para la activación de estas unidades elemento a elemento, de modo que las salidas se encontrarán siempre en el intervalo $(0, 1)$.

El siguiente paso consiste en integrar la información en la puerta de reinicio con la contenida en el estado escondido y la entrada del tiempo actual. En términos matemáticos, generamos un *candidato a estado escondido* $\tilde{\mathbf{H}}_t \in \mathbb{R}^{n \times h}$ al tiempo t, dado por:

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \mathbf{b}_h), \tag{7}$$

donde $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$ y $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ son parámetros de pesos, $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$ es el sesgo, y el símbolo $\odot$ es el producto de Hadamard elemento a elemento entre dos matrices de las mismas dimensiones. Observemos primero que utilizamos la no–linealidad de tanh para hacer que los valores del candidato a estado escondido permanezcan en el intervalo $(-1, 1)$.

Notemos que si las entradas de la puerta de reinicio $\mathbf{R}_t$ son todas 1, recuperamos la RNN que presentamos en la ecuación (3). Análogamente, si todas las entradas de $\mathbf{R}_t$ son 0, entonces el candidato a estado escondido contiene únicamente información de la entrada $\mathbf{X}_t$. En otras palabras, cualquier estado escondido preexistente se *reinicia*. Refiérase a la figura 3.

Finalmente, incorporamos el efecto de la puerta de actualización $\mathbf{Z}_t$. Esta determina en qué medida el nuevo estado escondido $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ contiene información del anterior estado $\mathbf{H}_{t-1}$, y en qué medida contiene información del candidato a estado escondido $\tilde{\mathbf{H}}_t$. Logramos ésto haciendo una combinación lineal convexa entre $\mathbf{H}_{t-1}$ y $\tilde{\mathbf{H}}_t$:

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t. \tag{8}$$

De manera análoga al anterior paso, observemos que si las entradas de la puerta de actualización son todas 1, entonces el estado escondido $\mathbf{H}_t$ contendrá información únicamente del anterior estado escondido $\mathbf{H}_{t-1}$, mientras que si son todas 0, el nuevo estado escondido contendrá únicamente información del candidato a estado escondido $\tilde{\mathbf{H}}_t$. Refiérase a la figura 4.

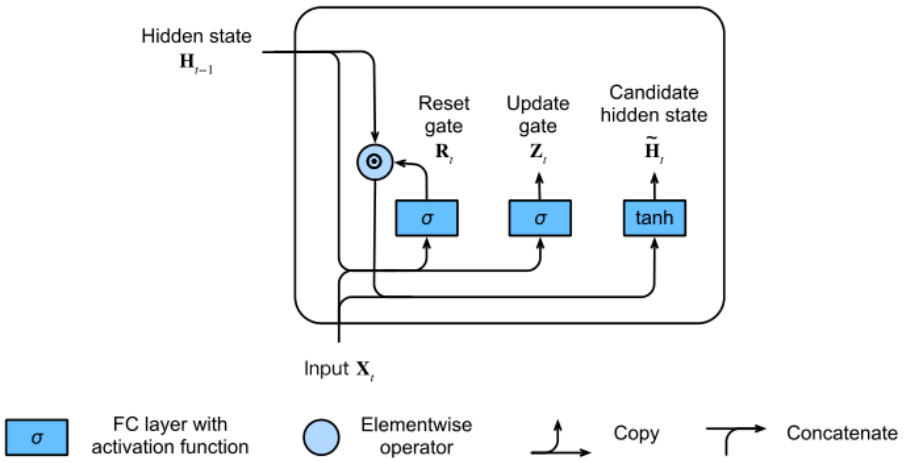

Figura 3: Cálculo del candidato a estado escondido en un modelo GRU.

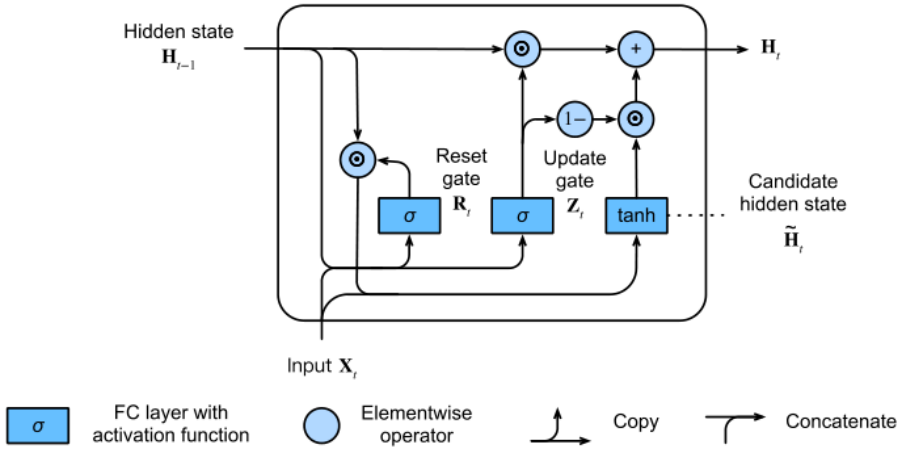

Figura 4: Cálculo del estado escondido en un modelo GRU.

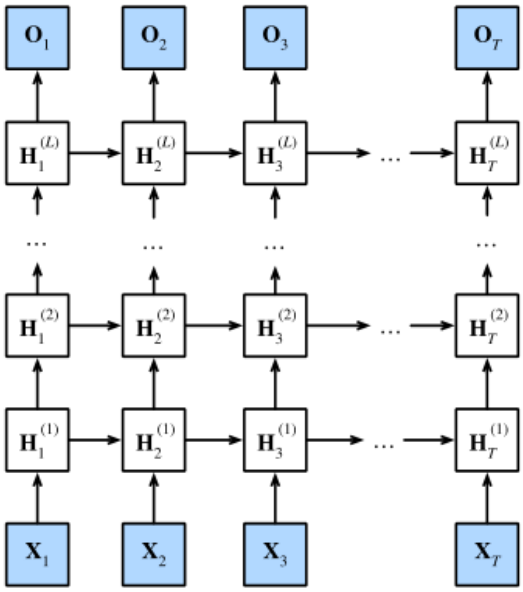

Figura 5: Arquitectura de una RNN profunda.

Estos diseños nos permiten capturar dependencias en secuencias muy largas. Por ejemplo, si la puerta de actualización ha sido cercana a 1 para todos los pasos de tiempo de alguna subsecuencia, el anterior estado escondido al tiempo de su inicio será mayormente retenido y pasado hasta el final de la subsecuencia, independientemente de la longitud de la subsecuencia.

En resumen, las GRUs tienen las siguientes características sobresalientes:

- Las puertas de reinicio capturan dependencias de corto plazo en secuencias.
- Las puertas de actualización capturan dependencias de largo plazo en las secuencias.

2.1.2. RNNs profundas

El siguiente paso para flexibilizar las redes recurrentes que hemos descrito hasta ahora consiste en agregar más capas de RNNs para aumentar la capacidad de estas redes de capturar dinámicas no–lineales más complejas.

Refiérase a la figura 5 para una representación gráfica de la arquitectura que planteamos.

Matemáticamente, para un minilote de ejemplos $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ (con n ejemplos y d dimensiones de entrada) al tiempo t, denominamos a la l–ésima capa escondida $\mathbf{H}_t^{(l)} \in \mathbb{R}^{n\times h}$ (con h unidades escondidas) para $l \in \{1, 2, \ldots, L\}$, donde $\mathbf{H}_t^{(0)} = \mathbf{X}_t$. A su vez, denominamos a la variable de la capa de salida $\mathbf{O}_t \in \mathbb{R}^{n\times q}$ (con q unidades de salida). Así, el estado escondido de la l–ésima capa de salida que usa la función de activación ϕ_t está dado por:

$$\mathbf{H}_t^{(l)} = \phi_t(\mathbf{H}_t^{(l-1)}\mathbf{W}_{xh}^{(l)} + \mathbf{H}_{t-1}^{(l)}\mathbf{W}_{hh}^{(l)} + \mathbf{b}_h^{(l)}), \tag{9}$$

donde $\mathbf{W}_{xh}^{(l)} \in \mathbb{R}^{d\times h}$ y $\mathbf{W}_{hh}^{(l)} \in \mathbb{R}^{h\times h}$ son parámetros de pesos y $\mathbf{b}_h^{(l)} \in \mathbb{R}^{1\times h}$ es un vector de sesgo.

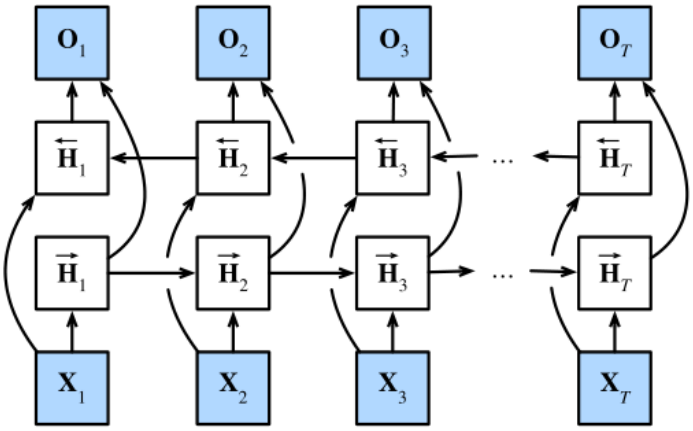

Figura 6: Arquitectura de una RNN bidireccional.

Después del cálculo de todas las capas escondidas, el cálculo de la capa de salida utiliza únicamente la información contenida en la L–ésima capa escondida. Es decir:

$$\mathbf{O}_t = \mathbf{H}_t^{(L)}\mathbf{W}_{hq} + \mathbf{b}_q, \tag{10}$$

donde $\mathbf{W}_{hq} \in \mathbb{R}^{h\times q}$ son los parámetros de pesos respectivos y el sesgo es $\mathbf{b}_q \in \mathbb{R}^{1\times q}$.

2.1.3. RNNs bidireccionales

Como mencionamos antes, las RNNs que hemos descrito hasta ahora son *unidireccionales*, en el sentido de que la información fluye conforme avanza el tiempo t.

En aprendizaje de secuencias existen tareas para las que la salida de una predicción puede depender de *toda la secuencia de entrada*. Por ejemplo, en reconocimiento de voz, la correcta interpretación de un sonido como un fonema puede depender de los siguientes sonidos por un fenómeno conocido como co–articulación. Si hay varias interpretaciones de una palabra y ambas son acústicamente plausibles, probablemente necesitaremos obtener información del pasado y del futuro de la sucesión para desambiguarlas.

Este mismo fenómeno se presenta en reconocimiento de escritura a mano, en análisis de sentimientos (que en el fondo es una tarea de clasificación de secuencias), en traducción automática de textos, y en otras tareas de aprendizaje de secuencia a secuencia.

Las RNNs bidireccionales se inventaron para resolver este tipo de problemas (Schuster & Paliwal, 1997). Para obtener información de toda la secuencia, utilizaremos una RNN que pase información en la dirección del tiempo de la secuencia, y la acoplaremos a otra que pase información en la dirección contraria. La figura 6 muestra la arquitectura de una RNN bidireccional con una sola capa escondida.

Matemáticamente, para un minilote de ejemplos $\mathbf{X}_t \in \mathbb{R}^{n\times d}$ (con n ejemplos y d dimensiones de entrada) al tiempo t, denominamos por $\overrightarrow{\mathbf{H}}_t \in \mathbb{R}^{n\times h}$ al estado escondido que pasa información en la dirección del tiempo. Análogamente, denominamos por $\overleftarrow{\mathbf{H}}_t \in \mathbb{R}^{n\times h}$ al estado escondido que pasa información en la dirección contraria al tiempo, y nombramos

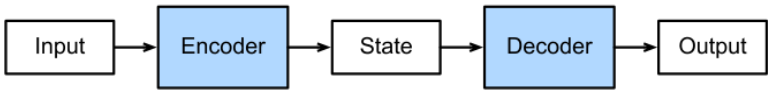

Figura 7: Arquitectura *encoder–decoder*.

a estos estados escondidos *estado escondido hacia adelante* y *estado escondido hacia atrás*, respectivamente. El cálculo de estos está dado por:

$$\overrightarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(f)} + \overrightarrow{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(f)} + \mathbf{b}_h^{(f)}), \tag{11}$$

$$\overleftarrow{\mathbf{H}}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh}^{(b)} + \overleftarrow{\mathbf{H}}_{t-1} \mathbf{W}_{hh}^{(b)} + \mathbf{b}_h^{(b)}), \tag{12}$$

donde los pesos $\mathbf{W}_{xh}^{(f)}, \mathbf{W}_{xh}^{(b)} \in \mathbb{R}^{d \times h}$, $\mathbf{W}_{hh}^{(f)}, \mathbf{W}_{hh}^{(b)} \in \mathbb{R}^{h \times h}$ $\mathbf{b}_h^{(l)} \in \mathbb{R}^{1 \times h}$ y los sesgos $\mathbf{b}_h^{(f)}, \mathbf{b}_h^{(b)} \in \mathbb{R}^{1 \times h}$ son todos los parámetros del modelo.

Después de hacer el cálculo de los estados escondidos hacia adelante y hacia atrás $\overrightarrow{\mathbf{H}}_t$ y $\overleftarrow{\mathbf{H}}_t$, concatenamos estos estados para obtener el estado $\mathbf{H}_t \in \mathbb{R}^{n \times 2h}$ que pasamos a la capa de salida. En RNNs bidireccionales con varias capas, esta información la pasamos como entrada a la siguiente capa bidireccional.

Para computar la salida de la red neuronal tenemos $\mathbf{O}_t \in \mathbb{R}^{n \times q}$ (con q salidas):

$$\mathbf{O}_t = \mathbf{H}_t^{(L)} \mathbf{W}_{hq} + \mathbf{b}_q, \tag{13}$$

donde $\mathbf{W}_{hq} \in \mathbb{R}^{2h \times q}$ son los parámetros de pesos respectivos y el sesgo es $\mathbf{b}_q \in \mathbb{R}^{1 \times q}$.

Es importante mencionar que el costo computacional de entrenar una RNN bidireccional es muy elevado, dado que la fase de propagación hacia adelante requiere hacer cálculos recursivos hacia adelante y hacia atrás en las capas bidireccionales, y que la retropropagación depende a su vez de las salidas de la propagación hacia adelante. Así, los gradientes tienen largas cadenas de dependencias.

2.1.4. Arquitecturas *encoder–decoder*

El último componente que introducimos es el de la arquitectura *encoder–decoder*, que es una de las arquitecturas idóneas para hacer aprendizaje de secuencia a secuencia.

Las arquitecturas *encoder–decoder* tienen dos grandes componentes: un codificador (*encoder*) y un decodificador (*decoder*). El diseño de estas redes puede observarse en la figura 7.

El codificador convierte las secuencias de entrada en una representación de un espacio latente; esta información se pasa como entrada al decodificador, que se encarga de convertir la representación obtenida en la salida que se requiera. Puede ser otra secuencia, una clasificación, el predictor continuo de una regresión, etc. Desde luego, en la elección de la arquitectura del decodificador debe tomarse en cuenta el tipo de salida.

2.2. *Word embeddings*

Cuando se realizan tareas de aprendizaje de secuencias debemos abordar necesariamente el problema de cómo representar estas secuencias como entradas de una red. En tareas de

procesamiento de lenguaje natural como la que abordamos, comenzamos siempre por generar un mapeo biyectivo entre todos los tokens únicos —palabras, letras, símbolos, fonemas, etc.— del corpus e índices que los identifiquen únicamente.

Si bien la representación numérica de tokens es mucho más eficiente en términos de memoria que si guardásemos los *strings* y los usáramos como entradas directamente a una red neuronal de alguna manera, una representación basada únicamente en números es posiblemente una mala representación de los tokens.

La forma más simple de representar los tokens es utilizando *one–hot encoding*. En esta representación codificamos cada token como un vector con tantas entradas como tokens únicos tiene el vocabulario del corpus cuyas entradas son todas 0 salvo por la entrada correspondiente al índice del token, que es 1. Es decir, que el *one–hot encoding* del i–ésimo token es el vector $(x_1, x_2, ..., x_d)$, donde $x_j = 1$ si $j = i$ y $x_j = 0$ si $j \neq i$.

Si bien esta representación facilita ciertos cálculos, como el cálculo de pérdida durante la fase de entrenamiento, la representación es altamente dispersa, y los vectores suelen ser de dimensión muy alta. Notemos además que al formar un conjunto ortogonal, somos incapaces de utilizar medidas de similitud para determinar si dos tokens comparados son similares o no.

Las técnicas que presentamos en esta sección tienen por objetivo hallar representaciones eficientes y dotadas de estructura que puede ayudar para mejorar el desempeño del modelo neuronal que construyamos.

2.2.1. `word2vec` (*skip–gram*)

La herramienta `word2vec` fue propuesta para abordar los anteriores problemas (Mikolov et al., 2013). El modelo mapea cada palabra en el vocabulario dado a un vector de longitud fija, y estos vectores pueden expresar mejor las relaciones de analogía y similitud entre diferentes palabras. La herramienta contiene dos modelos: *skip–gram* y *continuous bag of words* (CBOW), de los cuales nosotros describimos únicamente el primero.

El modelo *skip–gram* asume que una palabra se puede utilizar para generar sus palabras aledañas en una secuencia de texto. Tomemos la secuencia de texto «el», «hombre», «ama», «viajar», «mucho» como ejemplo, elijamos la palabra «ama» como *palabra central* y una *ventana de contexto* igual a 2.

Dada la palabra central «ama», el modelo *skip–gram* considera la probabilidad condicional de las *palabras contextuales*: «el», «hombre», «viajar», «mucho» que están a no más de 2 palabras de la palabra central:

$$P(\text{«el»}, \text{«hombre»}, \text{«viajar»}, \text{«mucho»} \mid \text{«ama»}). \tag{14}$$

Si asumimos que las palabras contextuales son generadas independientemente dada la palabra central (en el sentido de independencia condicional), entonces la ecuación (14) puede reescribirse como:

$$P(\text{«el»} \mid \text{«ama»}) \cdot P(\text{«hombre»} \mid \text{«ama»}) \cdot P(\text{«viajar»} \mid \text{«ama»}) \cdot P(\text{ «mucho»} \mid \text{«ama»}). \tag{15}$$

En el modelo *skip–gram* cada palabra tiene 2 vectores d–dimensionales para calcular las probabilidades condicionales. Concretamente, para alguna palabra con índice i en el

vocabulario, denotamos por $\mathbf{v}_i \in \mathbb{R}^d$ y $\mathbf{u}_i \in \mathbb{R}^d$ sus dos vectores: cuando es usada como palabra central, y cuando es usada como palabra de contexto, respectivamente.

La probabilidad condicional de generar cualquier palabra contextual w_o (con índice o en el vocabulario) dada la palabra central w_c (con índice c en el diccionario) puede modelarse con una operación *softmax* sobre productos punto como sigue:

$$P(w_o \mid w_c) = \frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)}, \tag{16}$$

donde el conjunto de índices del vocabulario es $\mathcal{V} = \{0, 1, \ldots, |\mathcal{V}| - 1\}$.

Para entrenar el modelo *skip–gram* utilizamos el método de máxima verosimilitud. Para ello recordemos que la función de verosimilitud es la probabilidad conjunta de observar los datos obtenidos como función de los parámetros del modelo.

En este caso, los parámetros son los vectores de palabra central y de palabra contextual de cada palabra en el vocabulario. La función de verosimilitud del modelo es, así, la probabilidad de generar todas las palabras contextuales dada cualquier palabra central como función de los vectores de palabra central y de palabra contextual de cada palabra en el vocabulario.

Matemáticamente, dada una secuencia de texto de largo T, donde la palabra al tiempo t se denota por $w^{(t)}$, si asumimos que las palabras contextuales son condicionalmente independientes de cualquier palabra central, entonces para una ventana de contexto m, la función de verosimilitud de *skip–gram* está dada por:

$$\prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m, \\ j \neq 0}} P(w^{(t+j)} \mid w^{(t)}), \tag{17}$$

donde cualquier tiempo menor que 1 o mayor que T es ignorado.

Observemos que maximizar la anterior función de verosimilitud es equivalente a minimizar la siguiente función de pérdida:

$$-\sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m, \\ j \neq 0}} \log P(w^{(t+j)} \mid w^{(t)}), \tag{18}$$

donde:

$$\log P(w^{(t+j)} \mid w^{(t)}) = \mathbf{u}_o^\top \mathbf{v}_c - \log \left(\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c) \right). \tag{19}$$

Para obtener los gradientes de la anterior función de pérdida respecto de los parámetros del modelo, diferenciamos normalmente. Como ejemplo, el gradiente respecto del vector $\mathbf{v}_c$ está dado por:

$$\frac{\partial \log P(w^{(t+j)} \mid w^{(t)})}{\partial \mathbf{v}_c} = \mathbf{u}_o - \frac{\sum_{j \in \mathcal{V}} \exp(\mathbf{u}_j^\top \mathbf{v}_c) \mathbf{u}_j}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \tag{20}$$

$$= \mathbf{u}_o - \sum_{j \in \mathcal{V}} \left(\frac{\exp(\mathbf{u}_j^\top \mathbf{v}_c)}{\sum_{i \in \mathcal{V}} \exp(\mathbf{u}_i^\top \mathbf{v}_c)} \right) \mathbf{u}_j \tag{21}$$

$$= \mathbf{u}_o - \sum_{j \in \mathcal{V}} P(w_j \mid w_c)\mathbf{u}_j. \tag{22}$$

Los gradientes para los otros vectores de palabras pueden obtenerse de modo similar.

Por último, en una nota técnica, notemos que como no utilizamos datos anotados en este proceso de aprendizaje, este algoritmo se conoce como de aprendizaje auto–supervisado.

El resultado del entrenamiento de este modelo —los vectores de palabra central y de contexto de todas las palabras— puede ser incorporado a un modelo de redes neuronales con una capa de *embedding* que realiza una búsqueda y devuelve las representaciones vectoriales obtenidas para los tokens que componen las secuencias de nuestro corpus. Dichos vectores pueden ser pasados como entrada a otras unidades de una red neuronal —como por ejemplo, a una unidad *encoder*—.

2.2.2. *GloVe*

Global Vectors (Pennington et al., 2014) es un modelo de *word embeddings* que ha tenido muy buenos resultados en tareas de aprendizaje de secuencias y que trata de resolver algunos problemas computacionales presentes en `word2vec`. Para entender mejor estos problemas revisitemos el modelo *skip–gram* para introducir notación esencial.

Primero denotemos por q_{ij} a la probabilidad condicional de tener la palabra contextual w_j dada la palabra central w_i en el modelo *skip–gram*; esto es:

$$q_{ij} = \frac{\exp(\mathbf{u}_j^\top \mathbf{v}_i)}{\sum_{k \in \mathcal{V}} \exp(\mathbf{u}_k^\top \mathbf{v}_i)}, \tag{23}$$

donde para cualquier índice i, $\mathbf{v}_i$ y $\mathbf{u}_i$ son los vectores de la palabra w_i como palabra central y contextual, respectivamente, y $\mathcal{V} = \{0, 1, \ldots, |\mathcal{V}| - 1\}$ es el conjunto de índices en el vocabulario.

Consideremos una palabra w_i que puede aparecer varias veces en el corpus. Denotemos por C_i al multiconjunto —es decir, que sus elementos pueden estar contenidos varias veces— de los índices de todas las palabras contextuales en el corpus que tienen a w_i como palabra central dada una ventana de contexto fija.

Al número de veces que ocurre la palabra con índice j en el multiconjunto C_i lo denotamos por x_{ij}. Utilizando esta estadística global del corpus, la función de pérdida del modelo *skip–gram* —ecuación (18)— puede escribirse como:

$$-\sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{V}} x_{ij} \log q_{ij}. \tag{24}$$

Ahora, si denotamos por x_i al conteo de todas las palabras contextuales que tienen a w_i como palabra central —es decir, $x_i = |C_i|$—, y si denotamos por p_{ij} al estimador de la probabilidad condicional de obtener la palabra contextual w_j dada la palabra central w_i —es decir, $p_{ij} = x_{ij}/x_i$, la ecuación (24) puede reescribirse como:

$$-\sum_{i \in \mathcal{V}} x_i \sum_{j \in \mathcal{V}} p_{ij} \log q_{ij}. \tag{25}$$

En la anterior expresión, observemos que el término $-\sum_{j\in\mathcal{V}} p_{ij}\log q_{ij}$ calcula la entropía cruzada de la distribución condicional de las estadísticas globales del corpus p_{ij} y la distribución condicional q_{ij} de predicciones del modelo. Desde este punto de vista es fácil ver que la función de pérdida en la ecuación (18) es una suma ponderada de la entropía cruzada entre las distribuciones p_{ij} y q_{ij}.

Los autores de GloVe argumentan que utilizar las co–ocurrencias de palabra a palabra, que definimos como p_{ik}/p_{jk}, en lugar de las probabilidades condicionales por sí solas, nos guía a una representación vectorial que aprovecha mejor estas estadísticas globales. Esta representación es el modelo GloVe, que además resuelve el problema de que normalizar apropiadamente q_{ij} resulta en la suma sobre todo el vocabulario, que puede ser computacionalmente muy costoso, y además no sufre del problema de que el modelado por entropía cruzada lleva, en corpus grandes, a que co–ocurrencias raras tengan demasiado peso.

Los autores de GloVe realizan tres cambios al modelo *skip–gram* utilizando un modelo de regresión log–cuadrática:

1. Usamos las variables $p'_{ij} = x_{ij}$ y $q'_{ij} = \exp(\mathbf{u}_j^\top \mathbf{v}_i)$ que no son distribuciones de probabilidad, y toman el logaritmo de de ambas, así que el término de pérdida cuadrático es $\left(\log p'_{ij} - \log q'_{ij}\right)^2 = \left(\mathbf{u}_j^\top \mathbf{v}_i - \log x_{ij}\right)^2$.

2. Agregan dos parámetros escalares al modelo para cada palabra w_i: el sesgo de palabra central b_i y el sesgo de palabra contextual c_i.

3. Remplazan el peso de cada término en la función de pérdida con la función de peso $h(x_{ij})$, donde h es creciente en el intervalo $[0, 1]$. Una forma que puede adquirir esta función es $h(x) = (x/c)^\alpha$ si $x < c$, y $h(x) = 1$ en otro caso, para alguna $\alpha > 0$.

La función de pérdida resultante del modelo GloVe está dada por:

$$\sum_{i\in\mathcal{V}}\sum_{j\in\mathcal{V}} h(x_{ij})\left(\mathbf{u}_j^\top \mathbf{v}_i + b_i + c_i - \log x_{ij}\right)^2. \tag{26}$$

Una vez que los vectores contextuales y centrales de todas las palabras han sido determinados a través del proceso de entrenamiento por descenso estocástico de gradiente, los vectores resultantes se suman para obtener la representación final.

3. Arquitectura final

La arquitectura final que utilizamos en este proyecto utiliza todos los elementos que hemos incluido en el marco teórico.

Recordemos que la tarea que intentamos resolver en este proyecto es la clasificación de sentimiento de tuits en español de México, utilizando redes neuronales recurrentes y *word embeddings*. Para una referencia de la arquitectura final observe la figura 8.

Utilizamos vectores pre–entrenados en español siguiendo el modelo GloVe, y usamos dichas representaciones utilizando una capa de *embedding* que pasa las representaciones vectoriales de las secuencias de entrada al *encoder*.

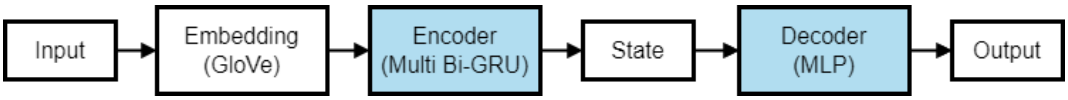

Figura 8: Arquitectura final para clasificación de sentimiento.

El *encoder* es una red GRU bidireccional de varias capas que sirve para codificar la información de cada tuit (en tanto que una sucesión completa de tokens) en un espacio latente que se pasa como entrada al *decoder*.

Finalmente, el *decoder* es una red de perceptrón multicapa completamente conectada. Esta red tendrá como objetivo traducir las representaciones de las secuencias en el estado latente a una representación vectorial de *one–hot encoding* de las tres categorías de interés: «positivo», «negativo» y «neutro». La función de pérdida que utilizamos es de entropía cruzada.

Referencias y bibliografía

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

Etcheverry, M., & Wonsever, D. (2016). Spanish Word Vectors from Wikipedia. *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, 3681-3685. https://aclanthology.org/L16-1584

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning* [http://www.deeplearningbook.org]. MIT Press.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short–Term Memory. *Neural Computation*, *9*(8), 1735-1780. https://doi.org/10.1162/neco.1997.9.8.1735

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

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. https://doi.org/10.3115/v1/d14-1162

Schuster, M., & Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673-2681. https://doi.org/10.1109/78.650093

TASS: Workshop on Semantic Analysis. (2020). Task 1 – train and dev sets. *SEPLN*. http://tass.sepln.org/2020/?wpdmpro=task-1-train-and-dev-sets

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into Deep Learning. *arXiv preprint arXiv:2106.11342*.