Guía de estudio - Redes Neuronales Recurrentes

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

En esta oportunidad nos sumergimos en estudiar las fascinantes y poderosas redes neuronales recurrentes y sus variantes, entre ellas las Long Short-Term Memory (LSTM) y Gated Recurrent Unit (GRU). En la actualidad estas arquitecturas juegan un papel importante en tareas que involucran secuencias de datos, como el procesamiento del lenguaje natural, la predicción de series temporales y todo aquello en que los datos se encuentren auto correlacionados.

Las Redes Neuronales Recurrentes poseen la capacidad de capturar información presente en la dependencia temporal que existe en la secuencia. Los modelos iniciales de estos modelos (RNN) presentan un problema no menor que tiene que ver con el desvanecimiento del gradiente, limitando su eficacia en secuencias largas. Para superar estas limitaciones aparece una arquitectura de RNN llamada LSTM, y posteriormente la arquitectura GRU. Estas arquitecturas introducen celdas de memoria y compuertas que permiten el flujo controlado de información, haciendo posible la retención de información y con esto mitigan el problema de gradientes desvanecientes.

En esta guía descubriremos la anatomía de las RNN y las mejoras que proponen las arquitecturas LSTM y GRU, para terminar implementando estas redes gracias a la librería Keras (con Tensorflow).

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Redes Neuronales Recurrentes (RNN)

Hasta ahora las redes neuronales que hemos visto, tanto las Fully Connected (FC) y Redes Neuronales Convolucionales (CNN), no poseen la capacidad de capturar información referente a la temporalidad en que se encuentren los datos. Por ejemplo, si le pasamos a una red CNN una secuencia de imágenes (Video), esta usará cada imagen como una observación diferente y aprenderá cada ejemplo en forma individual para lograr una predicción, sin capturar la información de la secuencialidad.

Las Redes Recurrentes tienen la capacidad de aprender información respecto de la temporalidad en una secuencia. Para lograr esto, las RNN pueden usar entradas que corresponden a una secuencia. Por ejemplo, en una frase sabemos que cada palabra no puede ir en cualquier lugar para que la frase tenga sentido, por lo que debe establecerse una secuencia. Otros ejemplos de datos secuenciales son los frames o imágenes en instantes de un video, o el audio en una conversación. Es importante señalar que los largos de las secuencias pueden ser diferentes entre ellas, lo que no ocurre con las redes vistas anteriormente.

Secuencias

Las secuencias las podemos representar como un vector en el que cada posición representa un valor en la secuencia.

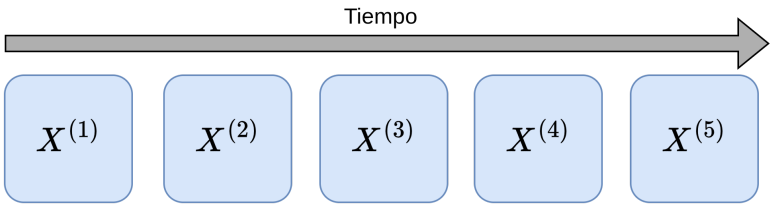

Figura 1. Representación de secuencia
Fuente: Desafío Latam.

Si las entradas fueran texto, entonces tendremos en cada observación una oración o frase de algún largo representando el largo de la secuencia **time_step**. Cada palabra puede ser representada por un vector de características **n_features**. Las dimensiones de un conjunto de entradas de tipo texto se verá como un tensor de **N x time_step x n_features.**

Configuraciones RNN Input/Output

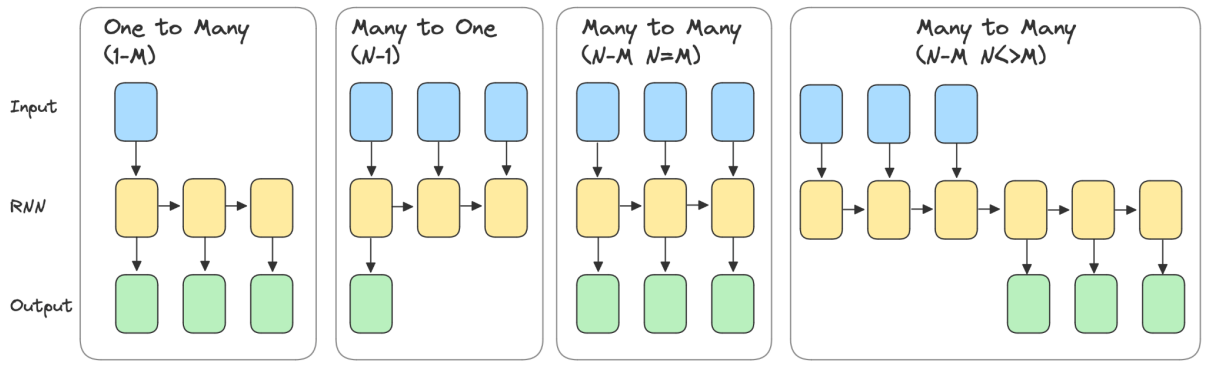

Figura 2. Configuración RNN Input/Output
Fuente: Desafío Latam

One to Many

Este es el caso en el que las entradas son datos que no son secuencias y las salidas son secuencias de largo M. Un ejemplo para este tipo es la descripción de imágenes, es decir, tenemos una imagen y la salida es una frase que describe la imagen.

Many to One

Las entradas son una secuencia de largo N, y la salida es una clase que puede ser un vector. Por ejemplo, el texto de entrada corresponde a opiniones, y la salida a una clasificación del texto en sentimiento positivo, negativo o neutral.

Many to Many (con tamaño de entrada igual al tamaño de salida)

En este caso tendremos secuencias de entrada con largo igual al largo de la secuencia de salida. Por ejemplo, dada una oración asignaremos a cada palabra de esta qué entidad corresponde en la oración, si es un verbo, sustantivo, artículo, etc. De este modo tendremos equivalencia uno a uno.

Many to Many (con tamaño de entrada (N > 1) diferente al de la salida (M > 1)

La entrada es una secuencia de largo N > 1, y la salida es otra secuencia con largo M > 1. Por ejemplo, la entrada de texto en español y salida de texto traducido a Inglés.

Arquitectura RNN

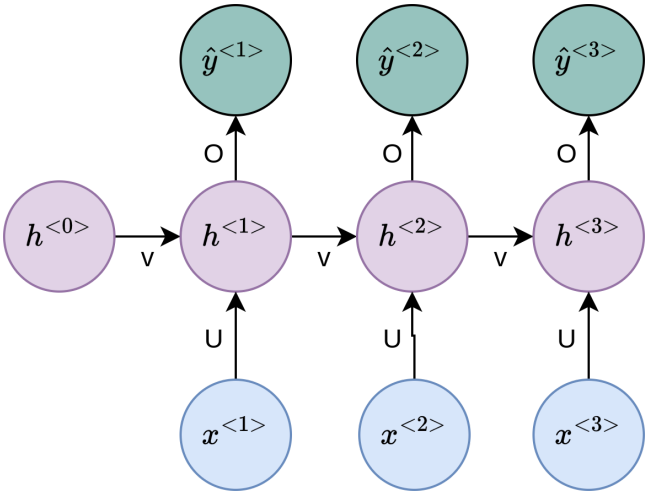

Figura 3. Arquitectura RNN
Fuente: Fuente propia.

La arquitectura de una RNN se puede ver en la Figura 3, donde las X's representan las entradas, $h^{<0>}$ corresponde al estado inicial anterior, $h^{<1>}$ es el nuevo estado que se construye usando información del estado anterior y de la entrada actual, para lo que se van aproximando las matrices compartidas de pesos U y V y luego para la salida se usa una nueva matriz de pesos O. Cuando se quiere obtener múltiples salidas se capturan las salidas de cada instante de tiempo; en caso de modelar un problema con una salida se considera sólo la última.

Las ecuaciones derivadas de esta arquitectura, considerando una salida, son

$$h^{<t>} = \tanh(X^{<t>}U + h^{<t-1>}V + b)$$
$$\hat{y} = \text{softmax}(h^{<N>}O + c)$$
$$\mathcal{L} = \text{CE}(\hat{y}, y)$$

Para realizar el aprendizaje, al igual que en las otras redes neuronales que hemos visto, se usa el algoritmo de Backpropagation que en este caso se conoce como BPTT (Backpropagation Through Time). La función de pérdida se calcula como la suma de las pérdidas en cada instante.

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}^{<t>}$$

Arquitectura RNN Multicapa

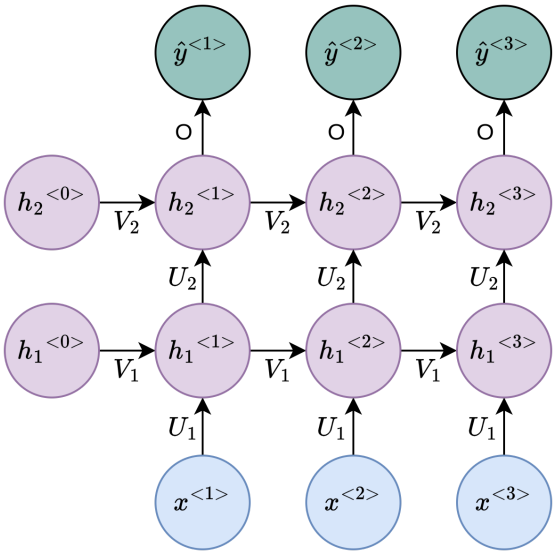

Figura 4. Arquitectura RNN Multicapa (2)
Fuente: Fuente propia.

Las RNN pueden tener secuencias como entradas, y como salidas clases o secuencias. Podrían además tener datos no secuenciales como entradas y salidas secuenciales.

En la Figura 4 se muestra una RNN con dos capas. Estas redes se construyen ubicando la segunda capa sobre la primera y conectando las salidas en cada instante de la capa 1 como entradas de la capa superior. De esta manera podemos apilar tantas capas como se requiera. El usar más capas le otorga a la Red mayor capacidad.

Problemas con las RNN

Las RNN sufren de dos problemas que ya hemos revisado antes, a saber: desvanecimiento del gradiente y gradientes explosivos. Esto significa que a medida que las secuencias son más largas la información que la red puede aprender de las secuencias más antiguas será mínima. Asimismo, si el peso de la arista recurrente es mayor a uno en términos absolutos entonces los pesos anteriores se vuelven muy grandes a medida que nos alejamos.

Para resolver estos problemas requerimos de las redes LSTM.

Arquitectura LSTM (Long Short-Term Memory)

Las LSTM se hacen cargo del problema de los gradientes desvanecientes incorporando ciertas compuertas lógicas y una celda de memoria. Esto le permite a la red decidir qué es

importante recordar y qué no lo es; de esta forma la red podrá enviar gradientes para actualizar a celdas muy antiguas en el caso que la red lo haya determinado importante.

En la Figura 5 se muestra la arquitectura de una red recurrente con bloque LSTM. Esta arquitectura se ve igual que la anterior, sólo que cambiamos la celda recurrente por una LSTM.

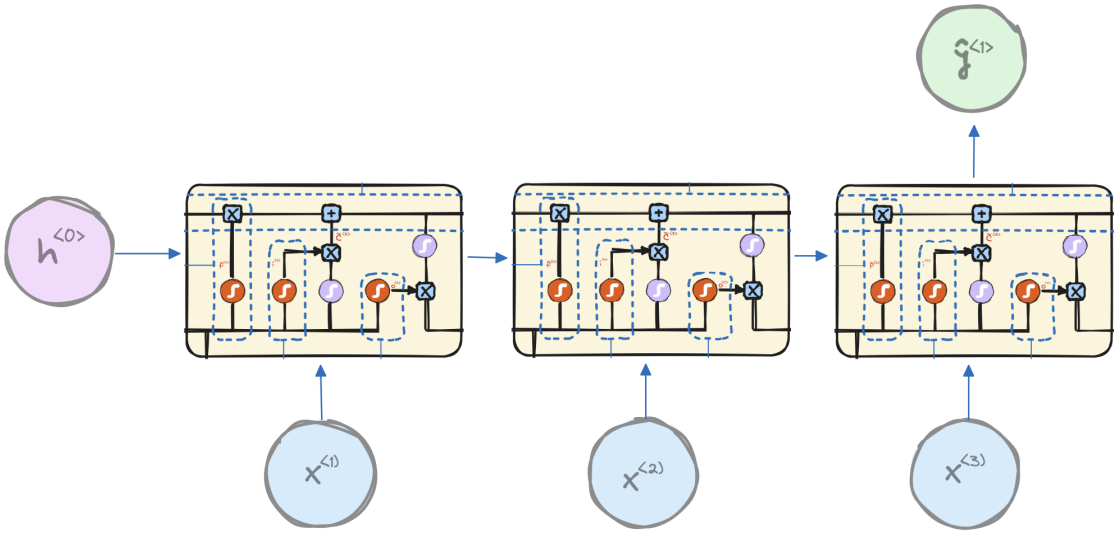

Figura 5. Arquitectura LSTM
Fuente: Desafío Latam

El bloque LSTM se compone de tres compuertas, como se muestra a continuación:

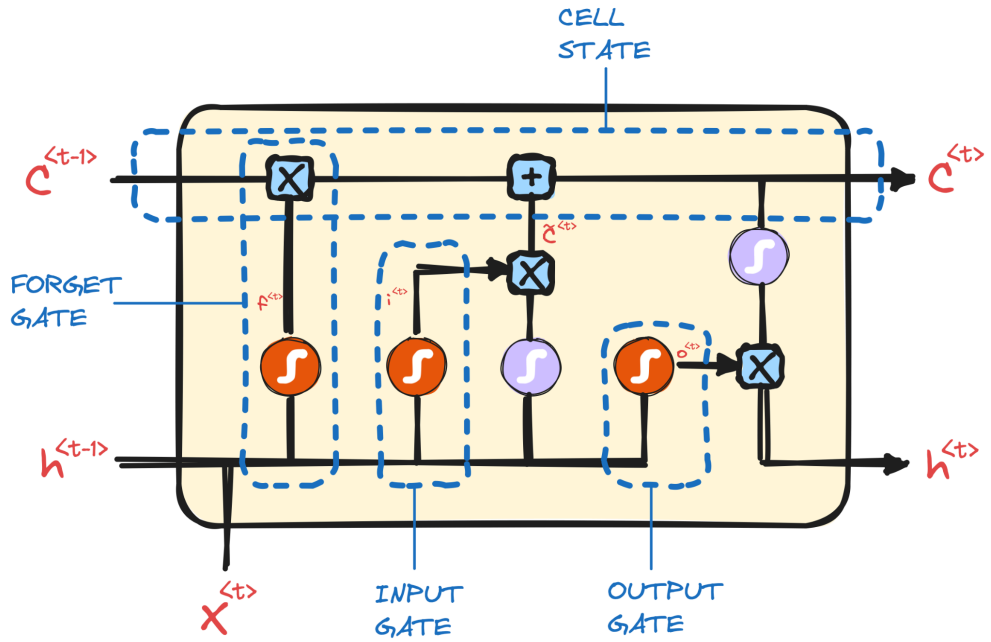

Figura 6. Bloque LSTM
Fuente: Desafío Latam.

1. **Forget Gate**: Esta compuerta permite a la celda de memoria (C) borrar o no recordar lo que se tenía en memoria. Se utiliza una función sigmoidal para la decisión.

$$f^{<t>} = \sigma(W_{xf}x^{<t>} + W_{hf}h^{<t-1>} + b_f)$$

2. **Input Gate:** Se encarga de evaluar qué información es importante para el estado de la celda. Este resultado, junto con el valor candidato, son los encargados de actualizar el estado de la célula.

$$i^{<t>} = \sigma(W_{xi}x^{<t>} + W_{hi}h^{<t-1>} + b_i)$$
$$\tilde{c}^{<t>} = \tanh(W_{xc}x^{<t>} + W_{hc}h)$$

Para calcular el estado de la célula usamos la siguiente función

$$c^{<t>} = (c^{<t-1>} \otimes f^{<t>}) \oplus (i^{<t>} \otimes \tilde{c}^{t})$$

3. **Output Gate:** Su misión es decidir cómo actualizar los valores de las unidades ocultas.

$$o^{<t>} = \sigma(W_{xo}x^{<t>} + W_{ho}h^{<t-1} + b_0)$$
$$h^{<t>} = o_t \otimes \tanh(c^{<t>})$$

Arquitectura GRU (Gate Recurrent Unit)

Este bloque nace en respuesta a la complejidad de los cálculos que se realizan en el bloque LSTM, reduciendo las compuertas a sólo dos. En la Figura 7 se muestra este bloque.

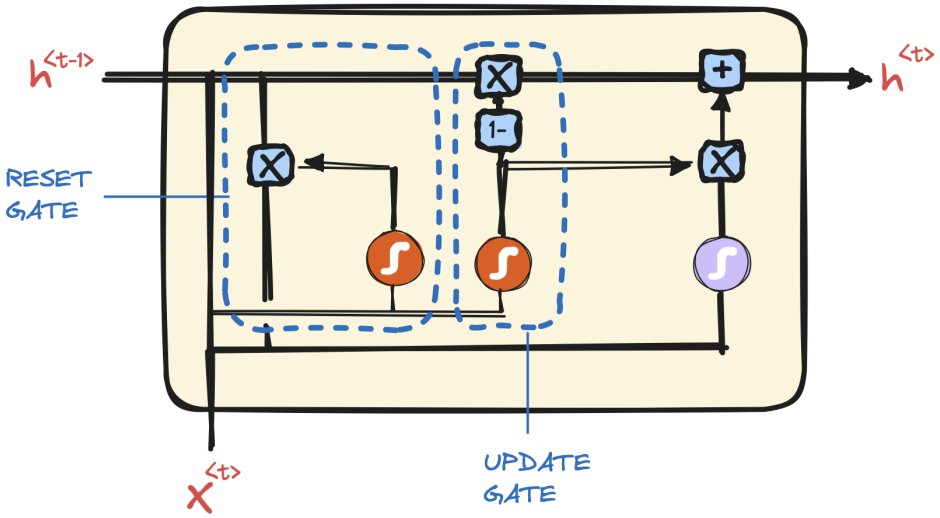

Figura 7. Bloque GRU
Fuente: Desafío Latam.

1. **Reset Gate:** Decide cuánta de la información del pasado debe olvidarse.
2. **Update Gate:** Se encarga de decidir cuanto de la información actual se debe incorporar en la memoria de corto plazo

Este bloque registra mejores desempeños en cuanto a velocidad computacional que los bloques LSTM.

Representación de Texto en vector

Habitualmente, para redes neuronales recurrentes trabajamos con textos, los que deben ser representados como vectores numéricos. Para realizar esto existen distintas técnicas, y revisaremos algunas de ellas.

Bolsa de palabras (Bag of Words BoW)

En este enfoque se ignora el orden y la estructura gramatical del texto, y se enfoca únicamente en la frecuencia de las palabras presentes en el documento. Para realizar esta representación aplicamos los siguiente pasos:

1. **Tokenización**: Divide el texto en unidades más pequeñas llamadas tokens. Estos token suelen ser palabras individuales, sin embargo, pueden ser combinación de dos o más palabras conocidas como n-gramas.
2. **Construcción del vocabulario**: Se construye un vocabulario único que contenga todas las palabras o n-grama diferentes en el conjunto de documentos.
3. **Creación del vector de características:** En esta etapa crearemos para cada documento un vector numérico, de modo que cada posición corresponde a una palabra en el vocabulario y el valor será la frecuencia de esa palabra en el documento.

Algunos de los problemas de este modelo radican en que no considera el orden de las palabras, el vector de características tendrá muchos ceros (sparse) y se otorga mayor importancia a las palabras que ocurren con mayor frecuencia.

TF-IDF (Term Frequency-Inverse Document Frequency)

Este modelo evalúa la importancia relativa de una palabra en un documento respecto a un conjunto más amplio de documentos. Para esto se combinan dos métricas: frecuencia del término (TF) y la frecuencia inversa del documento (IDF).

$$\mathrm{TF}(a, d) = \frac{\text{Número de veces que } a \text{ se encuentra en } d}{\text{Cantidad total de palabras en } d}$$

$$\text{IDF}(a, D) = \log\left(\frac{\text{Número total de documentos en } D}{(\text{Total de documentos que contienen a } a) + 1}\right)$$

$$\text{TF-IDF}(a, d, D) = \text{TF}(a, d) \times \text{IDF}(a, D)$$

Por medio de este modelo, las palabras en un documento y que son menos comunes en el conjunto completo de documentos reciben puntajes más altos.

Word Embedding

Es una técnica que asocia a cada palabra un vector de características, el cual incorpora información semántica y sintáctica entre las palabras. El propósito de estos vectores es capturar el significado y la similitud semántica entre palabras en un espacio vectorial más denso y continuo.

Word2Vec es una técnica particular de Word Embedding desarrollada por Mikolov et. al. en 2013. Esta técnica usa dos arquitecturas, llamadas **Continuous Bags of Words (CBOW)** y **Skip-gram**. En CBOW, su entrada es el contexto para predecir la palabra objetivo, en cambio en Skip-gram su entrada es la palabra objetivo para predecir el contexto.

Es común obtener embedding de palabras usando modelos pre entrenados como Word2Vec o GloVe.

Actividad guiada: Análisis de sentimiento en críticas de películas

En esta actividad se entrenará una red neuronal recurrente multicapa usando arquitectura LSTM. Además entrenaremos nuestro propio Word Embedding usando Word2Vec. El objetivo será predecir el sentimiento positivo o negativo asociado a cada crítica de película, en este caso cada crítica está etiquetada con su sentimiento.

En el desarrollo de esta guía aprenderemos a Tokenizar, realizar Padding a las secuencias, entrenar Word Embedding, construir un modelo red neuronal recurrente evaluando su desempeño y realizando predicciones.

Revisa nuevamente el archivo 01 - procesamiento de texto y críticas de películas que vimos en clase, para repasar detalladamente estos procesos.

Preguntas de proceso

Reflexiona:

- ¿Qué entendemos por Padding en el contexto de RNN?
- ¿Cómo se representan las palabras en un modelo de RNN?
- ¿Cual es la diferencia entre una arquitectura LSTM y una GRU?
- ¿Por qué un modelo RNN es mejor para secuencias que una red neuronal Fully Connected?
- ¿Mencione un ejemplo para arquitecturas RNN one-many?

Referencias bibliográficas

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press. ISBN: 9780262035613

Bishop, C. M., Bishop, H. & Cham, S. (ed.) (2023). *Deep Learning - Foundations and Concepts*. ISBN: 978-3-031-45468-4