Implementación de un Chatbot utilizando una Red Neuronal Seq2Seq

Nathaniel Calderón González[1] and Ernesto Mancebo Tavárez[2]

[1]MULCIA, Universidad de Sevilla, Sevilla, España

Junio 2020

Abstract

A continuación se presenta la memoria de un estudio de la implementación de un chatbot mediante el uso de una Red Neuronal bajo la arquitectura Seq2Seq, la misma es entrenada utilizando el Corpus de mensajes del microblog Twitter.

1 Objeto de Estudio

2 Estado del Arte

Implementación	**Ventajas**	**Desventajas**
K-Nearest Neightbor	Puede determinar anomalías y fácil de implementar.	Alto consumo de recursos.
Redes Neuronales Artificiales	Aprende patrones de transacciones previas y detecta fraudes en tiempo real.	Se debe de estudiar la técnica a utilizar.
Arboles de Desición	Maneja patrones no lineales.	Igual que las RNA, existen varias técnicas por lo que hay que estudiar cuál conviene.
Outlier Detection Method	Consume menos recursos que otras técnicas y puede trabajar con data en línea (online)	No es tan preciso como otros métodos.
Aprendizaje Profundo	Estudia y extrae patrones de grandes conjuntos de datos.	Es utilizado mayormente en procesamiento de imágenes y del lenguaje, por lo que para este campo hay poca información.

Table 1: Estado del arte para implementaciones de Chatbots.

3 Redes Seq2Seq

3.1 Redes Recurrentes - RNN

A medida que se analizan estructuras tales como una fotografía, todo el contenido es capturado a la vez y el mismo no cambia a través del tiempo mas no toda la información es analizada de esta manera; existen estructuras donde se depende de la evolución de la misma para conocer su significancia, es decir, se necesita conocer en un determinado punto sus valores previos, es por ello que necesitamos herramientas que traten la información de manera secuencial [Cita Naranjo].

Para solventar esto se tienen las Redes Neuronales Recurrentes (RNN) las cuales son diseñadas para el análisis de secuencias. Estas redes se caracterizan por lo siguiente:

- Comparten parámetros.

- Capaces de procesar secuencias de distintas longitudes.
- Detectan información relevante que pueden aparecer en distintas posiciones de la secuencia.

A modo de ejemplificar la última característica de estas redes se tiene *'El 23 de Enero salí a pasear'* o *'Salí a pasear el 23 de Enero'*.

Las RNN no conocen el significado de los símbolos que procesan, éstas infieren en el significado a partir de la extructura de la secuencia y la posición de los símbolos. Por otro lado, una RNN es capáz de lograr todo lo mencionado por el hecho de que tales redes tienen un estado al que en ocasiones se le puede escuchar referir como *memoria*.

El estado o memoria de tal red se logra a partir del hecho de que una RNN son redes con bucles dentro de ella.

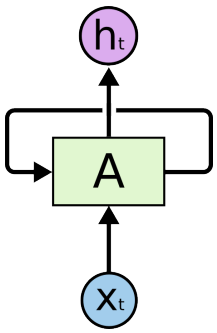

Figure 1: Ilustración de una RNN.

En la representación se tiene un trozo de red A la cual toma como entrada un x_t y emite un valor h_t, de este modo, utilizando el bucle mencionado, la información pasa al siguiente paso dentro de la red. Dicho ya esto, podemos ilustrar una red recurrente como una red con múltiples copias de sí misma[1].

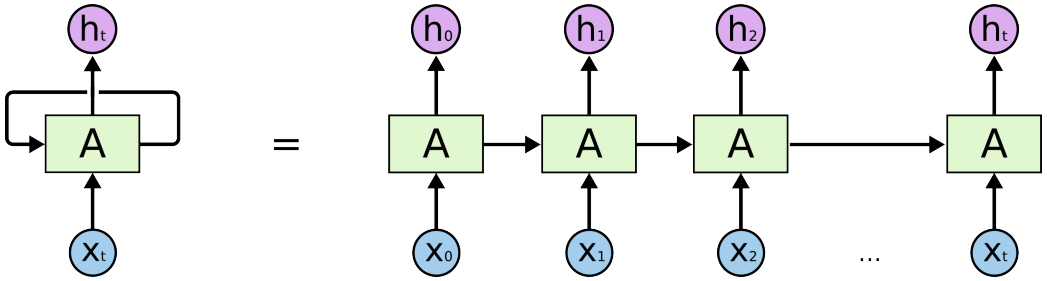

Figure 2: Ilustración de la evolución de una RNN a través del tiempo t.

Este comportamiento en secuencia es lo que enlaza las RNN con secuencias y listas, por ellos, tales redes son exitosas para problemas de reconocimiento de voz, modelado del lenguaje, traducción, contextualización de imágenes y otras aplicaciones relacionadas[2].

3.2 Redes Long Short-Term Memory - LSTM

Una variante las RNN capáz de aprender a partir de dependencias a largo plazo es la Long Short Term Memory, muchas veces llamadas tan solo LSTM.

Las LSTM fueron diseñadas con la intención de evitar problemas de dependencia a largo plazo, con la naturaleza de recordar información por un largo período de tiempo.

La estructura interna de una LSTM varía un poco de una RNN regular, donde en vez de tener una única capa, tiene cuatro y cada una interactúa de un modo distinto.

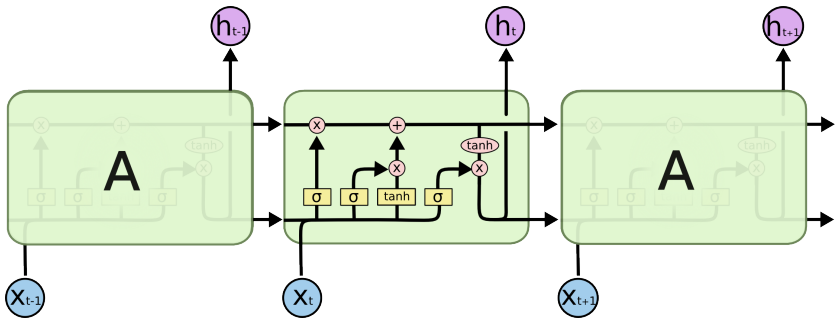

Figure 3: Ilustración de una red LSTM.

A continuación tenemos una radiografía de lo que conforma una red LSTM.

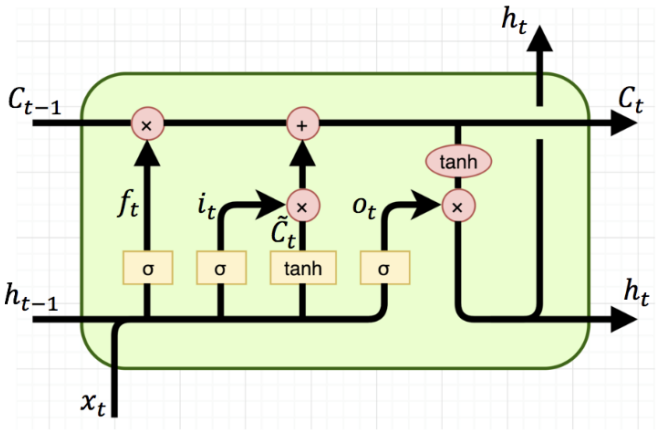

Figure 4: Composición interna de una LSTM.

Como se puede apreciar la red está compuerta por varias compuertas (*gates*) los cuales tienen la tarea de agregar o eliminar data al estádo de la celda. Cada celda es controlada por una capa que se apoya en la función sigmoide (σ) y la operación producto. Es preciso resalta que la función sigmoide retorna un valor entre cero y uno, siendo cero *no pase información* y uno *pase toda la información*. Finalmente, una LSTM contiene tres compuertas σ descritas de la siguiente manera:

- f_t o *"Puerta del Olvido"*, decide qué dato se debe de descartar en el estado de la celda tras observar los valores de h_{t-1} y x_t, retornando un número entre cero y uno para cada valor en el estado C_{t-1}.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1}$$

  En un problema de modelado del lenguaje, f_t trataría de predecir la siguiente palabra a partir de las anteriores. De modo que para un primer sujeto trataría de inferir en su género, sin embargo, para un siguiente sujeto la celda olvidaría el género del primero y tratára de identificar el del segundo.

- En el siguiente paso se decide qué información se debe de persistir en la celda tras dos pasos, primero la capa sigmoide llamada *"Capa Puerta de Entrada"* la cual decide qué valor se actualiza seguido de la creación de nuevos candidatos tras una capa que utiliza la Tangente Hiperbólica crea un vector de nuevos candidatos $\tilde{C}_t$. Estas luego se combinan y actualizan el estado de la celda en cuestión.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_C) \end{aligned} \tag{2}$$

  Siguiendo la línea del modelado del lenguaje, para el segundo sujeto en la secuencia se actualizaría su género en la celda reemplazando así el género del sujero previo.

- En este tercer paso se produce la actualización del estado anterior C_{t-1} en C_t, logrado esto en los pasos previos. Primero se olvidan los estados previos mediante f_t, se computa los nuevos valores mediante $i_t * \tilde{C}_t$ dándonos esto el nuevo valor para el estado C_t.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3}$$

- Finalmente se procede a emitir un valor de salida para la celda, mas éste valor debe de ser filtrado. El primer paso es a través de una sigmoide se decide qué se debe de emitir o_t, luego el estado actual de la celda C_t es evaluado por una Tangente Hiperbólica y finalmente multiplicado por el resultado de o_t, filtrándose así la salida[1].

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{4}$$

3.3 Redes Seq2Seq

Una implementación Seq2Seq consiste en dos redes recurrentes (RNN) enlazadas, una siendo codificadora y la otra decodificadora. La tarea de la codificador es leer una secuencia de entrada y emitir un contexto a partir del último estado oculto h_t a partir de lo que considere importante en la secuencia de entrada. Posteriormente se tiene una red decodificadora la cual utiliza una función softmax sobre el vocabulario a utilizar para así emitir una secuencia de salida[3][4].

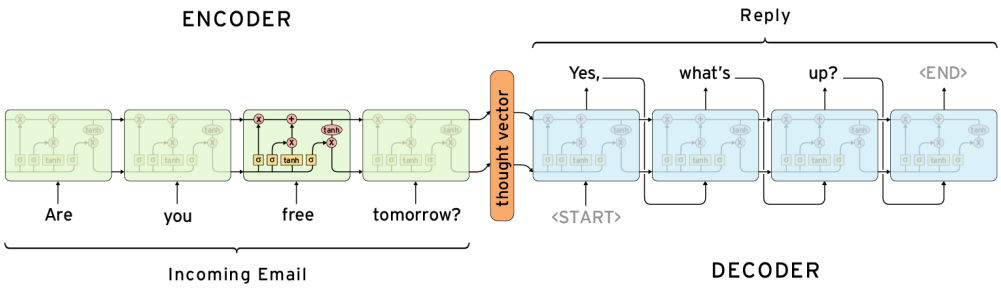

Figure 5: Comportamiendo de una red Seq2Seq.

Se aprecia en la parte central de un modelo Seq2Seq lo que se conoce como *Though Vector* o vector de pensamiento, el cuál representa la intención de la secuencia de entrada y a partir de aquí la sección decodificadora del modelo genera una secuencia a emitir, símbolo tras símbolo, palabra en este caso, donde cada símbolo es influenciado por el anterior.

4 Corpus

El corpus utilizado para entrenar esta red corresponde a un conjunto de más de 700 mil mensajes procedentes de la red social Twitter. Este corpus es provisto por el respositorio chat_corpus[5] donde detalla que para las líneas impares corresponden a *tweets* y las líneas pares corresponden a respuestas al *tweet* predecesor.

5 Implementación

5.1 Retos de la Implementación

Existen varios retos para el modelo Seq2Seq y uno de ellos es la variabilidad de la longitud de la secuencias, pues la longitud de las secuencias de entrada a procesar es indiscutiblemente variable. Otro problema es la dimención del vocabulario a utilizar, pues es ineficiente aplicar una función softmax a cada palabra de nuestro vocabulario[3].

5.2 Relleno de Oraciones (Padding)

Previo al entrenamiento del modelo, el corpus es manipulado con el propósito de estandarizar la longitud de las secuencias mediante el rellenado o *padding*, lográndose tras el uso de los siguientes comodines:

- **EOS**: Denotándo el Fin de la secuencia (*End of sentence*).
- **PAD**: Relleno.
- **GO**: Bandera de inicio de la decodificación.
- **UNK**: Palabra desconocida no pertinente al vocabulario (*Unknown*).

A modo de ilustración, y haciéndole saber al lector que este y los siguientes ejemplos son en el idioma inglés, pues el corpus utilizado durante el experimento está en dicho idioma, se tiene la siguiente iteracción:

Pregunta: How are you?
Respuesta: I am fine

Asumiendo que estandarizamos nuestro corpus o conjunto de datos para una longitud de diéz caracteres, tendríamos lo siguiente:

Pregunta: [PAD, PAD, PAD, PAD, PAD, PAD, '?', 'you', 'are', 'How']
Respuesta: [GO, 'I', 'am', 'fine', '.', EOS, PAD, PAD, PAD, PAD]

5.3 Ternas (Bucketing)

En el apartado anterior se resolvió un primer problema, pero digamos, si tenemos que nuestra secuencia de entrada más larga en nuestro conjunto de datos o corpus es de cien palabras, para el caso 'how are you?' necesitaríamos 96 rellenos. Esto nos estaría distorcionando la información para el caso referido.

Es por ello que se propuso en la implementación de estudo el uso de ternas de distintos tamaños tales como: $[(5, 10), (10, 15), (20, 25), (40, 50)]$. Las ternas fueron utilizadas del siguiente modo: si una pregunta consta de 4 elementos y así su respuesta como lo fue en el ejemplo previo, éste par fue puesto en la terna $(5, 10)$, rellenándose la pregunta a una longitud 5 y la respuesta a una longitud 10.

Ilustrando ésta implementación para el ejemplo de la sección previa, tendríamos:

Pregunta: [PAD, '?', 'you', 'are', 'How']
Respuesta: [GO, 'I', 'am', 'fine', '.', EOS, PAD, PAD, PAD, PAD]

5.4 Contextualización de Palabras (Word Embedding)

La contextualización de palabras es una técnica para el aprendizaje de representaciones densas de palabras dentro de un vector de reducida dimención. Se puede valorizar cada palabra como un punto en un espacio y tal espacio es representado por el vector ya referido. Relaciones semanticas entre palabras son obtenidas mediante ésta técnica pues tienen propiedades aproximadas.

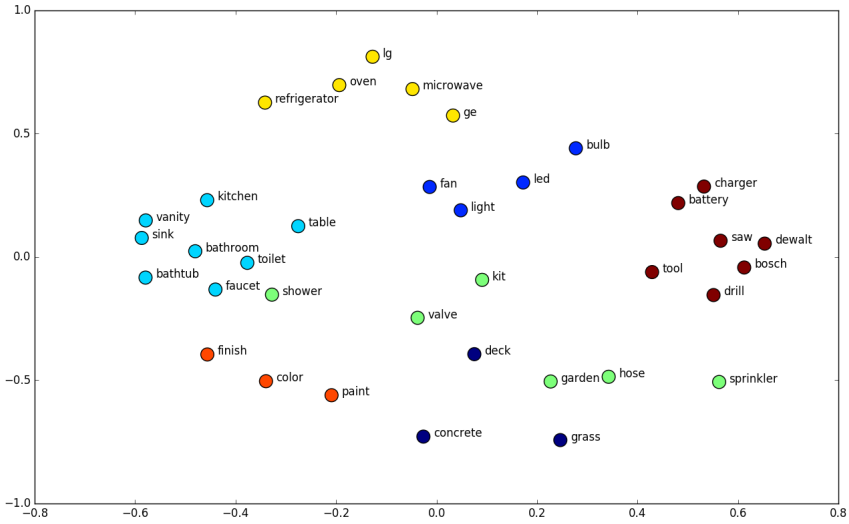

Figure 6: Word Embedding.

Ésta tarea es típicamente aplicada en la primera fase de la red y esto se conoce como *Embeding Layer*, donde se asocia una palabra (index to word in vocabulary) del vocabulario para un vector del tamaño dado. Para la arquitectura Seq2Seq los pesos de la capa *Embeding Layer* son entrenados en conjunto con otros parámetros del modelo[3].

5.5 Mecanismo de Atención (Attention Mechanism)

Una limitante de la arquitectura Seq2Seq es que la secuencia de entrada debe de ser llevada a una longitud determinada, y se ha de tomar en cuenta que a medida que incrementa la secuencia ésta es más propensa a la pérdida de información, por ello una red Seq2Seq tiende a dar mejores resultados para secuencias cortas.

Es por lo planteado anteriormente que surge lo conocido como Mecanismo de Atención (*Attention Mechanism*)[6], lo cual consiste en que la sección decodificadora de la red observa de manera selectiva a la secuencia de entrada

mientras decodifica, tomando la carga del codificador para codificar piezas útiles de información.

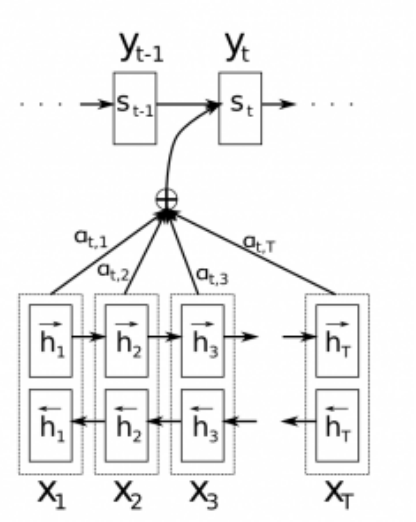

Figure 7: Mecanismo de Atención.

Para cada instante de tiempo el decodificador en vez de tomar un contexto de tamaño fijo (el último estado del codificador), un vector contexto distinto c_i es utilizado para generar la letra y_i, donde para el vector contexto c_i es la suma de los pesos de los estados ocultos en el codificador $c_i = \sum_{j=1}^{n} \alpha_{ij} h_j$, siendo n la longitud de la secuencia de entrada, h_j el estado oculto en el instante de tiempo j; $\alpha_{ij} = \exp(e_{ij}) / \sum_{k=1}^{n} \exp(e_{ik})$, y finalmente e_{ij} representa el modelo que decodifica el estado oculto en s_{i-1} y el j-ésimo estado oculto del codificador. Éste modelo es categorizado como una Red Neuronal hacia delante (*feedforward*) la cual entrena en conjunto con el resto del modelo.

Cada estado oculto en el codificador encapsula información acerca del contexto local en la oración. A medida que se genera data desde la palabra 0 hasta la palabra n, la información asociada al contexto mencionado se desvanece. Es por ello la necesidad de que el decodificador sepa acerca de cada contexto local, pues distintas partes de la secuencia de entrada dan pie a las piezas a utilizar en la secuencia de salida.

El modelo mencionado nos da una medida de qué tan bien una salida en

la posición i encaja con la entrada en la posición j, partiendo de ello se toma la suma de los contextos de entrada o estados ocultos a fin de generar una secuencia de salida.

6 Implementación del Diseño

7 Resultados

8 Conclusiones

Bibliografía

[1] *Understanding LSTM Networks – colah's blog*. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[2] *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[3] Suriyadeepan Ramamoorthy. *Chatbots with Seq2Seq*. URL: http://suriyadeepan.github.io/2016-06-28-easy-seq2seq//.

[4] Suriyadeepan Ramamoorthy. *Practical seq2seq*. URL: http://suriyadeepan.github.io/2016-12-31-practical-seq2seq//.

[5] Marsan Ma. *marsan-ma/chat_corpus*. May 2020. URL: https://github.com/marsan-ma/chat_corpus.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. arXiv: 1409.0473 [cs.CL].