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.

Contents

1 Objeto de Estudio

[PENDIENTE]

2 Marco Teórico

2.1 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.

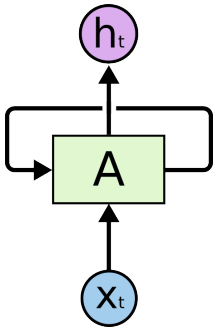

Figure 1: Ilustración de una RNN.

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].

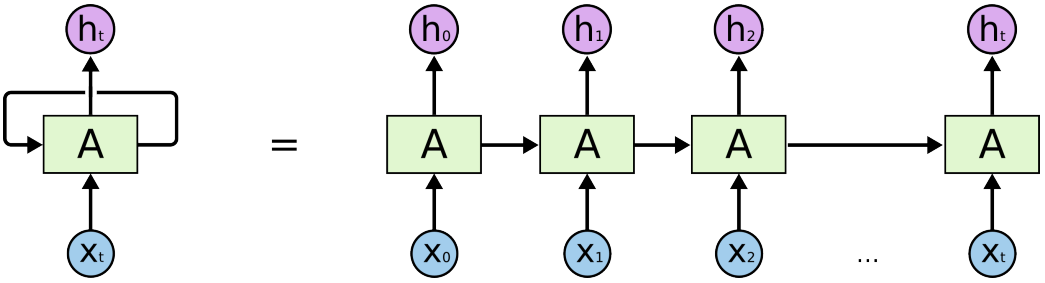

Figure 2: Ilustración de la evolución de una RNN a través del tiempo t.

[REVISAR TEXTO] 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].

2.2 Gated Recurrent Unit (GRU)

Las redes Gated Recurrent Unit, en lo adelante GRU, son un tipo de red recurrente(RNN) más sofisticadas que las RNN tradicionales y no sufren de desvanecimiento del gradiente, un problema típico en las RNN.

Las redes GRU utilizan dos compuertas para mitigar los problemas del gradiente:

- Compuerta de actualización *(Update Gate)*.
- Compuerta de reinicio *(Reset Gate)*.

Estas compuertas representan dos vectores de la red que deciden qué información retener y qué información descartar con respecto al vector de información de salida de la red.

El siguiente diagrama esboza el flujo de la red:

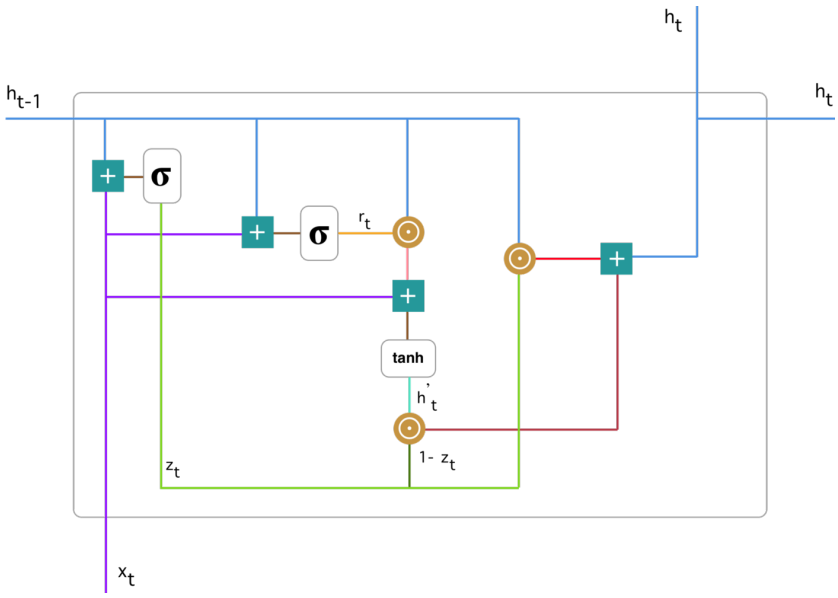

Figure 3: Ilustración de una red GRU.

La notación utilizada corresponde a la terminología común manejada en el área de aprendizaje automático, las cuales se definen como:

- σ: Función Sigmoide.
- tanh: Función Tangente Hiperbólica.
- $\odot$: Operación Producto Hadamard.
- $+$: Operación Adición.

[LO DE ABAJO ES DE LA MISMA LISTA O NECESITA UNA ORIENTACION COMO LA LISTA DE ARRIBA??]

- z_t representa la compuerta de actualización, la cual se calcula a partir de multiplicar el peso de la compuerta por x_t y h_{t-1}, luego sumar los resultados y aplicar la función sigmoide.

$$z_t = \sigma(W^z x_t + U^z h_{t-q}) \tag{1}$$

- r_t representa la compuerta de reinicio, básicamente se calcula igual que la compuerta de actualización, solo que con sus pesos.

$$r_t = \sigma(W^r x_t + U^r h_{t-1}) \tag{2}$$

- Entonces, la forma en que interviene la compuerta de reinicio para determinar qué información descartar de la celda estado o h'_t es calculada de la siguiente forma:

$$h'_t = \tanh(W x_t + r_t \odot U h_{t-1}) \tag{3}$$

  Se multiplica x_t por el peso W, y lo mismo para h_{t-1} por el peso U donde luego este resultado se usa para calcular el producto *Hadamard* con r_t. Por último se suman ambos resultados y se aplica la tangente hiperbólica. Básicamente cuando r_t tiende a 0 pues significa que se debe descartar tal información.

- Finalmente, para calcular la celda estado h_t se utiliza la fórmula:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t \tag{4}$$

  Donde se calcula el producto de Hadamard para la compuerta de actualización y la celda estado previa, y el producto de *Hadamard* para la misma compuerta de actualización y la celda estado h'_t. En esencia si el vector z_t tiende a uno pues retendrá la mayoría de la información previa y por consiguiente ignorando gran parte de la información actual[3].

2.3 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[4][5].

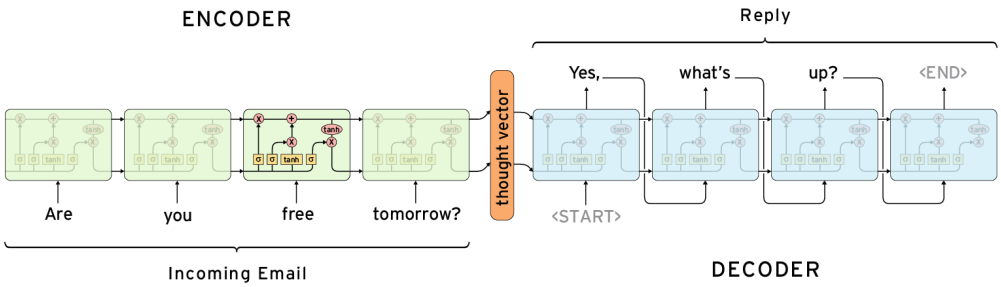

Figure 4: Comportamiendo de una red Seq2Seq.

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.

2.4 Beam Search

Consiste en un algoritmo de búsqueda similar al Greedy Search, donde mientras Greedy Search siempre considera una única alternativa como la mejor, este algoritmo organiza las predicciones en forma de árbol y expande los nodos con mayor probabilidad. Es necesario definir el Beam With o Top N lo cual limita a un número n el número de alternativas que serán evaluadas en paralelo[6].

2.5 Contextualización de Palabra (Word Embedding)

La contextualización de palabras o word embedding, es una estrategia que permite vectorizar las palabras de un vocabulario de forma eficiente y en la que la codificación preserva la relación entre palabras. La contextualización de palabras es realizada usando una matriz de n dimensiones de números reales los cuales representan pesos entrenables que son aprendidos durante el entrenamiento, de la misma forma que son aprendidos los pesos de una red.

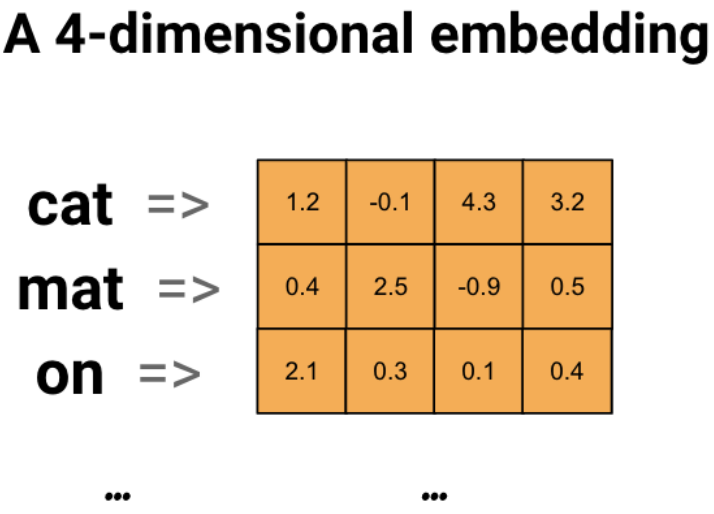

Figure 5: Ilustración de una matríz de Word Embedding.

A cada palabra del vocabulario del dataset se le asigna un índice en la matriz el cual es utilizado para referenciar la contextualización o *"embedding"* a dicha palabra. Por esta razón se suele interpretar esta capa como una tabla de búsqueda, o *"lookup table"*[7].

2.6 Tecnologías Utilizadas

2.6.1 TensorLayer

Es una librería consistente en una extensión o una capa de alto nivel para la librería TensorFlow de Google. Esta librería al igual que TensorFlow permite realizar aprendizaje profundo *(Deep Learning)* y aprendizaje por refuerzo *(Reinforcement Learning)* pero aun nivel de abstracción superior, simplificando muchas de las tareas que comúnmente realizamos con TensorFlow.

TensorLayer proporciona módulos de Aprendizaje Profundo y Aprendizaje por Refuerzo populares listos para ser usados, los cuales pueden ser modificados y adaptados a necesidades específicas de forma simple. Tal es el caso de la arquitectura de red Seq2Seq, la cual viene ya implementada y disponible para ser usada, y es la utilizada en este proyecto con ligeras modificaciones.

También es importante destacar que a pesar de TensorLayer ser una librería de alto nivel, la misma no tiene efectos sobre el rendimiento de TensorFlow. Lo cual además de ser simple y flexible la hace aún más atractiva para la comunidad de científico de datos[8].

2.6.2 NLTK

NLTK o Natural Language Toolkit, es una plataforma para trabajar con procesamiento del lenguaje natural en Python. NLTK cuenta con una coleccion de librerias para las típicas tareas de procesamiento de texto tales como, clasificación de textos, tokenización, análisis gramatical (parsing), razonamiento semántico, entre otros. NLTK también ofrece incorporado soporte a diferentes corpus y modelos de Aprendizaje Automático ya entrenados[9].

Para este proyecto se utilizó la última versión, NLTK 3, compatible con Python 3. Se utilizó NTLK para el preprocesamiento del corpus utilizando la función *FreqDist* para obtener la distribución de frecuencia de las palabras del corpus.

3 Estado del Arte

Nombre	**Plataforma**	**Propósito**	**Características**
@HotelReviewsBot[10]	Python, NLTK	Consultar cualidades de un Hotel registrado en http://www.hostelworld.com/	Capturaba tweets que contenían enlaces a un hotel en el referido portal y extraía comentarios relevantes del hospedaje, ya sea sobre el wi-fi, desayuno, baño, ducha o ruido del lugar.
A Tool of Conversation: Chatbot[11]	Java, Java Applets, Base de Datos	Demostrar la implementación de un flujo conversacional a partir de un esquema de patrones	Sugiere el uso de patrones para encontrar preguntas ya registradas en la base de datos y de hallarla, dar una respuesta al usuario.
Chatbots Enlatados[12]	Computación en la Nube	Proveer a empresas soluciones listas para su uso	Tienden a estar asociadas a ciertas industrias de manera predefinidas, tales como restaurantes y tiendas en línea.
Practical Sec2Sec[5]	Python, NLTK, Tensorflow	Chatbot capaz de dar respuestas coloquiales, a partir de lo aprendido de corpus como Twitter y Cornell Movies	Implementa una red neuronal utilizando Tensorflow la cual es entrenada a partir de un corpus de conversaciones.

Table 1: Estado del arte para implementaciones de Chatbots.

4 Corpus y Modelado de la Data

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 repositorio chat_corpus[13] donde detalla que para las líneas impares corresponden a *tweets* o mensajes emitidos y las líneas pares corresponden a respuestas al *tweet* predecesor.

Para modelar este vasto conjunto de datos es preciso realizar una serie de pasos con la intención de reducir la complejidad de aprendizaje para la red a entrenar, pues el modelo Seq2Seq trae consigo varios retos y uno de ellos es la variabilidad de la longitud de la secuencias, por el hecho de que la longitud de las secuencias de entrada a procesar es indiscutiblemente variable. Otro problema destacable es la dimención del vocabulario a utilizar ya que sin

duda alguna es ineficiente aplicar una función softmax para cada palabra del vocabulario con el cual trabajaremos[4].

4.1 Conversión a minúscula

La primera tarea de preprocesado del corpus, todas las palabras en el mismo son llevadas a minúsculas con el objetivo de que palabras lexicográficamente equivalentes sean en un posterior paso consideradas al momento de evaluar su distribución.

4.2 Filtrado de caracteres

Cada carácter dentro para cada palabra es filtrado mediante una expresión regular de forma [a-z0-9\s], permitiéndonos tan solo procesar texto alfanumérico, evitando que la red considere escenarios complejos donde haya presencia de símbolos.

4.3 Ternas (Bucketing)

Es preciso para la entrenar una red Seq2Seq el que las secuencias de entrada sean de una misma longitud, siendo esto un inconveniente pues las secuencias en el corpus son variables en cuanto a longitud se trata. Para solventar esta situación existe lo conocido como ternas o "bucketing", donde se elige un subconjunto tanto de preguntas como de respuestas las cuales entren dentro del rango establecido.

Para el problema a tratar se tuvo el siguiente criterio, una terna o bucket $((0, 20), (3, 20))$ siendo el primer binomio el rango para las preguntas y el segundo el de respuestas, leído de forma: Una pregunta o secuencia de entrada debe de corresponder a una longitud no mayor a 20 palabras, mientras que una respuesta debe de constar entre tres y veinte palabras.

4.4 Indexado de Palabras

Este paso es uno de los más importantes y críticos de la implementación, pues se trata de crear una tabla de búsqueda en donde se tenga una distribución de frecuencia para cada palabra obtenida del paso anterior, siendo la primera en la lista la palabra más utilizada y lo opuesto para la última palabra, dígase la menos frecuente. Asimismo, en la implementación estudiada la tabla obtenida es delimitada a tan solo las primeras seis mil palabras, por lo que se restringe el dominio del vocabulario a tal cantidad.

La implementación de ésta técnica es lograda apoyándose en la ya expuesta librería NLTK y su función *FreqDist*, y se ha de destacar que una vez obtenida la distribución de frecuencia para cada palabra en el corpus y restringido el dominio del vocabulario, se añaden en la cabecera de la lista los caracteres '_' que representan rellenado o "padding" y el caracter "unk" o desconocido para palabras fueras del vocabulario obtenido; estos caracteres sirven de apoyo a la red pues como se explicará en el apartado siguiente, es preciso tener tales caracteres para situaciones excepcionales a la cual nos enfrentaremos.

[VER SI VALE LA PENA ESTA SUBSECCION]

4.5 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.

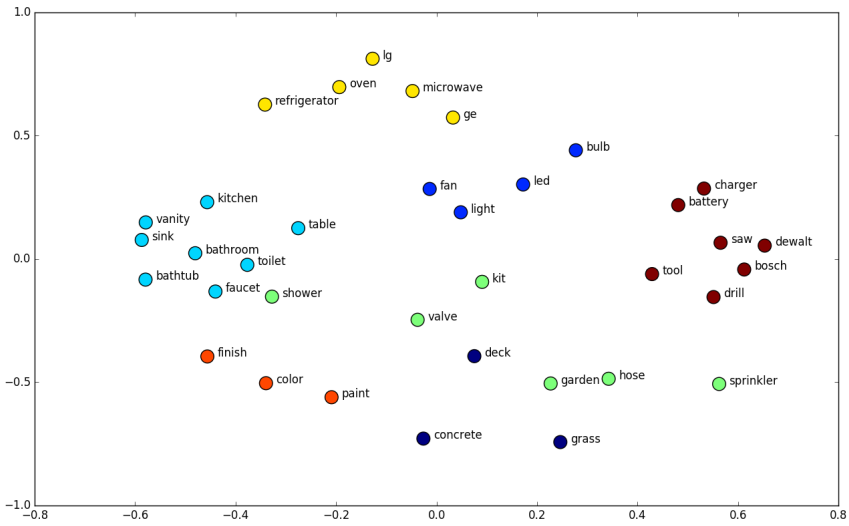

Figure 6: Word Embedding.

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[4].

4.6 Rellenado de Oraciones (Padding)

Como bien fue mencionado en el apartado "ternas", es necesario al momento de entrenar que las secuencias utilizadas en la red ocupen la misma dimensión, considerando el hecho de que existen secuencias de longitudes inferiores a veinte, siendo veinte el límite superior tanto para pregunta como respuesta. Es por esto que cada vector de secuencia el cual no alcance la dimensión mencionada para cada conjunto es acotado a tal dimensión, dígase veinte unidades, donde para las secuencias con menos de veinte unidades son rellenadas hasta alcanzar la dimensión en cuestión.

5 Arquitectura

```
seq2seq(
  (rnn_4): RNN(cell=GRUCell, n_units=256, name='rnn_4')
  (rnn_5): RNN(cell=GRUCell, n_units=256, name='rnn_5')
  (rnn_6): RNN(cell=GRUCell, n_units=256, name='rnn_6')
  (dense_1): Dense(n_units=8004, No Activation, in_channels='256',
 name='dense_1')
  (embedding_1): Embedding(vocabulary_size=8004, embedding_size=
1024)
  (rnn_1): RNN(cell=GRUCell, n_units=256, name='rnn_1')
  (rnn_2): RNN(cell=GRUCell, n_units=256, name='rnn_2')
  (rnn_3): RNN(cell=GRUCell, n_units=256, name='rnn_3')
  (reshape_1): Reshape(shape=[-1, 256],name='reshape_1')
  (reshape_2): Reshape(shape=[-1, 20, 8004],name='reshape_2')
  (reshape_3): Reshape(shape=[-1, 1, 8004],name='reshape_3')
)
```

Figure 7: Arquitectura modelo Seq2Seq.

- Embedding Layer(vocabulary_size=8004, embedding_size=1024)
- Codificador:
  - 3 - GRU Layer de 256 unidades
  - Embedding Layer(vocabulary_size=8004, embedding_size=1024)
- Decodificador
  - 3 - GRU Layer de 256 unidades
- Reshape Layer (shape(-1,256))
- Dense Layer de 8004 unidades, recibe una entrada de un tensor de 256 canales(shape=(1, 256)).
- Reshape Layer (shape(-1, 1, 8004))
- Capa de contextualización de palabras o Word Embedding, se utiliza previo a pasar el vector de ID de las palabras de la secuencia de entrada al codificador. También se usa previo a pasar el token especial "start_id" al decodificador.

- Codificador
  - Esta conformado por tres capas GRU(GRUCell), de 256 unidades cada una.
  - Convierte las palabras de la secuencia de entrada en vectores de estados ocultos. Cada vector de estado oculto representa una palabra y el contexto de esta.
- Decodificador
  - Tambien esta conformado por tres capas GRU(GRUCell), de 256 unidades cada una.
  - La capa inicial recibe de entrada un vector con el token especial "start_id" y se le pasan los estados ocultos del codificador como los estados iniciales del stack capas.
  - El codificador predice palabra por palabra la secuencia de salida, recibiendo en cada paso en el tiempo, o instante T:
    1. Un contexto de todas las palabras hasta ese instante.
    2. Los estados ocultos producidos hasta ese instante.
  - Para maximizar la probabilidad de la palabra producida se utiliza el algoritmo de búsqueda Beam Search con número de 3 alternativas definido por defecto, también conocido como "top n" o "beam width".
  - La computación termina cuando se produce el número de palabras definido como threshold. Para este caso fue una secuencia de máximo 20 palabras.

La salida del codificador es un vector de 20 palabras con él se produce la salida final concatenando palabra por palabra hasta encontrar el token especial *"end_id"*[8][14].

6 Entrenamiento y Creación del modelo

```
model = Seq2seq(
    decoder_seq_length=decoder_seq_length,
    cell_enc=tf.keras.layers.GRUCell,
    cell_dec=tf.keras.layers.GRUCell,
    n_layer=3,
    n_units=256,
    embedding_layer=tl.layers.Embedding(vocabulary_size=vocabulary_size, embedding_size=emb_dim),)

```

Figure 8: Creación del modeloSeq2Seq.

Para llevar a cabo el entrenamiento de la red neuronal ya descrita, es utilizado el corpus preprocesado resultante de apartados anteriores, donde tal corpus es segmentado en las siguientes proporciones: un 70% del conjunto de datos es utilizado para entrenamiento, un 15% para prueba y el 15% restante para validación[15].

Asimismo, el entrenamiento de la red es realizado durante una serie de 50 epochs y en cada epoch le es suministrado como ejemplo un subconjunto o batch de 32 instancias.

[VER SI SAMPLING VA] Se utilizan dos tokens especiales como prefijo y sufijo de cada secuencia, "start_id" y "end_id" respectivamente. Donde a la hora de realizar sampling, "start_id" es el token que representa el inicio de la secuencia, y "end_id", representa el final, lo cual permite saber cuando terminar la operación de sampling.

Posteriormente se evalúa las predicciones del modelo a partir de la salida del mismo y se calcula la pérdida mediante el cálculo de entropía cruzada utilizando la función cross_entropy_seq_with_mask provista por tensorflow, el valor computado como pérdida es utilizado para calcular el ajuste del gradiente para los pesos de la red y posteriormente es aplicado a tales pesos, esto ocurre durante cada iteración o epoch.

7 Diálogo con el Modelo (Inference)

Tras el oportuno entrenamiento del modelo el mismo es evaluado proveyendo al mismo un texto o secuencia de entrada, respondiendo el modelo una secuencia de salida. Se ha de destacar que todo esto es apoyado en la tabla de indexado que se construye en la fase de modelado del corpus, donde la entrada digitada es llevada a su representación numérica, y la salida de la red es llevada a texto por el proceso inverso, buscando la equivalencia de cada índice con una palabra, y la unión de este vector es la secuencia producto legible por el usuario.

A continuación se muestra una tabla donde se ilustran las preguntas y respuestas obtenidas por quienes elaboraron este proyecto. Es importante resaltar que el dominio de las preguntas y respuestas se circunscribe al contenido observado en el corpus.

[tabla]

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] Simeon Kostadinov. *Understanding GRU Networks*. Oct. 2019. URL: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

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

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

[6] Renu Khandelwal. *An intuitive explanation of Beam search*. Jan. 2020. URL: https://towardsdatascience.com/an-intuitive-explanation-of-beam-search-9b1d744e7a0f.

[7] *Word embeddings — TensorFlow Core*. URL: https://www.tensorflow.org/tutorials/text/word_embeddings.

[8] Hao Dong et al. "TensorLayer: A Versatile Library for Efficient Deep Learning Development". In: *ACM Multimedia* (2017). URL: http://tensorlayer.org.

[9] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. 1st. O'Reilly Media, Inc., 2009. ISBN: 0596516495.

[10] *Practical Natural Language Processing with Hostel Reviews 2015*. Nov. 2015. URL: https://www.racketracer.com/2015/11/18/practical-natural-language-processing-for-determing-wifi-quality-in-hostels/.

[11] Menal Dahiya. "A Tool of Conversation: Chatbot". In: *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING* 5 (May 2017), pp. 158–161.

[12] Maruti Techlabs. *How To Develop a Chatbot From Scratch*. Dec. 2018. URL: https://chatbotsmagazine.com/how-to-develop-a-chatbot-from-scratch-62bed1adab8c.

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

[14] *seq2seq model in Machine Learning*$_2$018. Nov. 2018. URL: https://www.geeksforgeeks.org/seq2seq-model-in-machine-learning/.

[15] Denny Britz. *Recurrent Neural Network Tutorial, Part 4 – Implementing a GRU/LSTM RNN with Python and Theano*. Sept. 2015. URL: http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/.