Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957
*ISSN 2007-1957*

# REDES NEURONALES LSTM PARA LA GENERACIÓN AUTOMÁTICA DE TEXTO A PARTIR DE NOVELAS CODIFICADO EN PYTHON

**Israel Rivera Zárate**
*Instituto Politécnico Nacional-CIDETEC*
*irivera@ipn.mx*

**Miguel Hernández Bolaños**
*Instituto Politécnico Nacional-CIDETEC*
*mbolanos@ipn.mx*

**Patricia Pérez Romero**
*Instituto Politécnico Nacional-CIDETEC*
promerop@ipn.mx

## Resumen

*Long Short-Term Memory (LSTM) is a variation of recurrent network (RNN) able to analyze variable length data sequences and overcomes the problem of vanishing and exploding gradients present in RNN's training stage. It's a powerful class of computational model that correlates information in the short or long-term of chronologically dependent data. LSTMs can use their internal state (memory) and several internal gates to be able to know what to forget and what to remember. So, when a new input comes in, the model first forgets any long-term information it decides it no longer needs. Then it learns which parts of the new input are worth using and saves them into its long-term memory. In the present work it is shown an example of the application of LSTMs in automatic text generation.*

*Palabras clave: Red LSTM, Procesamiento de Secuencias, Lenguaje Python.*

Las redes LSTM *("long short-term memory"*- memoria de largo y corto plazo) son una evolución de las redes recurrentes o RNN, fueron propuestas por primera vez en 1997 por Sepp Hochreiter y Jürgen Schmidhuber. De acuerdo con Christopher Olah (2015). Una red RNN experimenta el desvanecimiento del gradiente, o la explosión del gradiente en su fase de entrenamiento; por lo que interrumpe su aprendizaje, permitiéndole exclusivamente aprender a modelar dependencias a corto plazo (es decir, relaciones cercanas en la serie cronológica). La LSTM puede aprender dependencias largas, por lo que se podría decir

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957
*ISSN 2007-1957*

que tiene una memoria a más largo plazo. Ejemplos de aplicación se tienen en el *"image captioning"* donde la entrada es una imagen y la salida es una secuencia de caracteres; un texto, que describe el contenido de la imagen. Otro ejemplo es el reconocimiento de escritura desarrollado por *Google* y ya disponible en dispositivos con sistema Android que permite al usuario escribir directamente sobre la pantalla táctil de su dispositivo móvil y el sistema es capaz de reconocer y convertir a texto. Por su parte, Microsoft ha desarrollado aplicaciones que reconocen texto y permiten tomar decisiones con base en el contenido y lo ha aplicado en las áreas de automatización y de IOT (Internet de las Cosas). Adicionalmente, existen los convertidores de voz a texto o texto a voz. En las áreas de la medicina y la genética en particular se han usado las redes LSTM para la detección de modificaciones en la secuencia de ADN o bien en la detección de arritmias cardiacas. En la música se introducen algunas notas y el sistema genera una melodía correspondiente.

## I. Estructura de una red LSTM.

Las redes LSTM funcionan de forma similar a como el cerebro analiza las secuencias. Por ejemplo, si se analiza un texto; el cerebro se enfoca solamente en las palabras relevantes y se desecha el resto de la información. Como lo indica Bengio, (Bengio et al., 1994). Las redes LSTM son capaces de añadir o desechar la información que se considere relevante para el procesamiento de la secuencia. Comparada con una celda de red recurrente básica; la celda LSTM tiene una entrada y salida adicional, este elemento adicional se conoce como celda de estado: *"cell state*". Esta celda de estado es la clave del funcionamiento de las redes LSTM. La celda de estado es como una banda transportadora a la que se pueden añadir o donde se pueden remover datos que no se desean preservar en la memoria de la red. Para añadir o remover datos se utilizan varias compuertas: Compuerta de olvido *("forget gate")*: que permite eliminar elementos de la memoria. Compuerta de entrada ("*update gate")*: que permite añadir nuevos elementos de la memoria. Compuerta de salida *("output gate")*: que permite crear el estado oculto actualizado. Estas compuertas son redes neuronales que funcionan como válvulas donde totalmente abiertas permiten el paso de la información y totalmente cerradas lo bloquean por completo. Ver figura.1

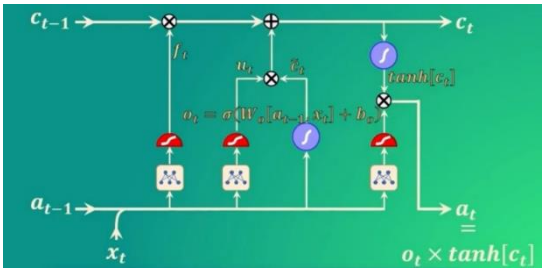


*Figura 1. Estructura de una celda LSTM. Tomado de: http:// colah.github.io/posts/2015-08-Understanding-LSTMs*

De acuerdo con Alex Graves (Graves, 2012). Cada una de las compuertas están conformadas de tres elementos: una red neuronal, una función sigmoidal y un elemento multiplicador. La función sigmoidal es precisamente la que le da el comportamiento de válvula ya que alcanza valores entre 0 (totalmente cerrada) y 1 (totalmente abierta).

A. *Forget gate*: Toma el estado oculto anterior y la entrada actual los transforma y los lleva a la función de activación

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957
ISSN 2007-1957

sigmoidal. Los coeficientes $W_f$ se aprenden durante el entrenamiento y como salida genera el vector $f_{(t)}$. Si uno de los valores de este vector es 0 o cercano a 0, entonces la LSTM eliminará esa porción de información. Mientras que si alcanza valores iguales o cercanos a 1; esta información se mantendrá y llegará a la celda de estado. Ver ecuación 1.

$$f_{(t)} = \sigma\left(W_f \left[h_{(t-1)}, X_{(t)}\right] + b_f\right) \ldots \boldsymbol{1}$$

B. *Update gate*: Se toma el estado oculto anterior y la entrada actual, se transforman y se llevan a una función de activación sigmoidal. También este caso los coeficientes $W_i$ se aprenden durante el entrenamiento y como salida esta compuerta genera el vector $u_{(t)}$. En este caso los valores que se desean preservar serán los cercanos a 1. Ver ecuación 2.

$$i_{(t)} = \sigma\left(W_i \left[h_{(t-1)}, X_{(t)}\right] + b_i\right) \ldots \boldsymbol{2}$$

Teniendo ya los datos generados por las compuertas *forget* y *update*, es posible ya actualizar la celda de estado; es decir, la memoria de la red LSTM. En primer lugar, se elimina la información irrelevante de la celda de estado: multiplicando el valor anterior de esta celda por el vector generado por la compuerta *forget*. A continuación, se crea un vector de valores "candidatos" a formar parte de la nueva memoria $\hat{c}_{(t)}$. Nuevamente los coeficientes $W_c$ se aprenden durante el entrenamiento. Posteriormente, se deben filtrar éstos últimos valores: multiplicando punto a punto el vector obtenido por el generado por la compuerta *update*; el resultado es sumado a los valores anteriores a la celda de estado, generando así la memoria actualizada. Finalmente, se debe calcular el nuevo estado oculto; para lo cual se utilizará la compuerta de salida. Ver ecuación 3.

$$\hat{c}_{(t)} = \tanh\left(W_c \left[h_{(t-1)}, X_{(t)}\right] + b_c\right) \ldots \boldsymbol{3}$$

C. *Output gate:* Este estado oculto de salida es simplemente una versión filtrada del *cell state* que se acaba de generar. Para garantizar que se encuentre en los rangos de -1 a 1 (rango del estado oculto) se emplea la función $tanh$. Ahora se usará la compuerta de salida, para determinar qué porciones del *cell state* entrarán a formar parte del nuevo estado oculto. Al igual que en los casos anteriores los coeficientes $W_o$ se aprenden durante el entrenamiento. Como último, se filtran los valores del *cell state* con el vector generado por la compuerta de salida. Ver ecuaciones 4, 5 y 6. En particular las ecuaciones 5 y 6 utilizan el producto de *Hadamard* que es una operación binaria que toma dos matrices de las mismas dimensiones y produce otra matriz de iguales dimensiones que éstas. También se conoce como producto punto a punto, ya que cada elemento *i, j* en la matriz generada, es el producto de los elementos *i, j* de las dos matrices iniciales.

$$o_{(t)} = \sigma\left(W_o \left[h_{(t-1)}, X_{(t)}\right] + b_o\right) \ldots \boldsymbol{4}$$

$$c_{(t)} = f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes \hat{c}_{(t)} \ldots \boldsymbol{5}$$

$$h_{(t)} = o_{(t)} \otimes \tanh c_{(t)} \ldots \boldsymbol{6}$$

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957
ISSN 2007-1957

Cuando se analiza una secuencia, realmente se tienen réplicas de celdas LSTM; cada una de ellas correspondientes a un instante de tiempo diferente dentro de la secuencia. De este modo, se evidencia el concepto de *cell state* como banda transportadora; la información puede ser removida o añadida de la memoria, basta entrenar adecuadamente las compuertas *forget* y *update.* Se observa que, con el entrenamiento adecuado, que la información almacenada en el estado $c_{(0)}$ se propague fácilmente hasta el estado $c_{(3)}$ o estados posteriores; y que la información irrelevante sea eliminada de la memoria en los momentos adecuados. Ver figura 2.

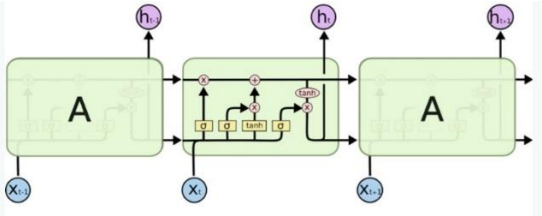


*Figura 2. Red de celdas LSM.*
*Tomado de: http:// colah.github.io/posts/2015-08-Understanding-LSTMs*

## II Entrenamiento de la red LSTM

El descenso del gradiente es el algoritmo de optimización utilizado para el entrenamiento; donde se busca minimizar la función de error de los valores de salida obtenidos en relación con los esperados respecto a los parámetros de la red (producto de derivadas parciales empleando la regla de la cadena). Según indica Fei-Fei (Fei-Fei et al., 2017). El resultado total del error es suma de las derivadas parciales calculadas en cada intervalo *time-step*. El proceso termina cuando se logra ajustar los pesos de las neuronas (coeficientes $W_{hh}$, $W_{xh}$, y $W_{hy}$) respecto a la función de coste o pérdida $L$, de ahí que se llama *"backpropagation through time"* (BTT). Ver figura 3.

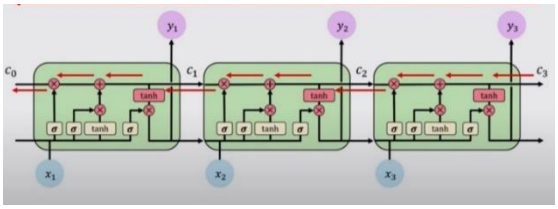


*Figura 3. Entrenamiento de la red LSTM mediante Backpropagation through time (BTT). Tomado de: http:// colah.github.io/posts/2015-08-Understanding-LSTMs*

Las ecuaciones 7 a 10 muestran el parámetro ɳ, que es la tasa de aprendizaje que permite ajustar los pesos en pasos reducidos; el parámetro *f*, es la función de activación y la *diag(x)* es la matriz diagonal de estados previos con las $h_{(i-1)}$ en su diagonal principal.

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \quad \ldots \mathbf{7}$$

$$\frac{\partial L_t}{\partial W} = \sum_{K=1}^{t} \frac{\partial L_t}{\partial Y_t} \frac{\partial Y_t}{\partial h_t} \frac{\partial h_t}{\partial h_K} \frac{\partial h_K}{\partial W} \ldots \mathbf{8}$$

$$\frac{\partial h_t}{\partial h_K} = \prod_{i=K+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=K+1}^{t} W^T \; diag\,[f'\,(h_{i-1})] \ldots \mathbf{9}$$

$$W_{new} = W - \eta \nabla_W L \ldots \mathbf{10}$$

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957
*ISSN 2007-1957*

Se debe destacar que los pesos de las neuronas (coeficientes $W_{hh}$, $W_{xh}$, y $W_{hy}$) son en realidad matrices, donde $W_{xh}$ tiene dimensiones: el número de elementos del vector de salida por el número de elementos del vector de entrada. Y donde $W_{hh}$ como $W_{hy}$ tienen dimensiones: el número de elementos del vector de salida por el número de elementos del vector de salida.

## III. Caso de Estudio

Apoyados de los trabajos de **Andrej Karpathy.** (2015). Se ha propuesto un ejemplo en *Python* donde se busca obtener una red que produzca texto a partir de novelas. Se proporciona un archivo con varias páginas del libro: el señor de los anillos de J. R. R. *Tolkien*. La red lee una secuencia de caracteres y predice cuál será el siguiente carácter del *stream* de texto. Se puede observar que las páginas tienen algo menos de 3000 caracteres y que cuando se convierten a minúsculas sólo hay 38 caracteres distintos en el vocabulario para que la red aprenda. El texto es dividido en porciones de 60 caracteres, una longitud arbitraria. Se podría dividir los datos en frases y rellenar las secuencias más cortas y truncar las más largas. Cada patrón de entrenamiento se compone de 60 time-step de un carácter de entrada o $X_{(t)}$ seguido de la salida de un carácter o $Y_{(t)}$. Al crear estas secuencias, se desliza esta ventana a lo largo de las páginas, lo que permite aprender de los 60 caracteres que precedieron. Por ejemplo, si la longitud de la secuencia es 5 (por simplicidad), entonces un patrón de entrenamiento sería: h, o, b, b, i que produciría la secuencia: h, o, b, b, i, t; entre otras palabras. Se pretende que el texto producido sea parecido al estilo de Tolkien. Ver figura 4.

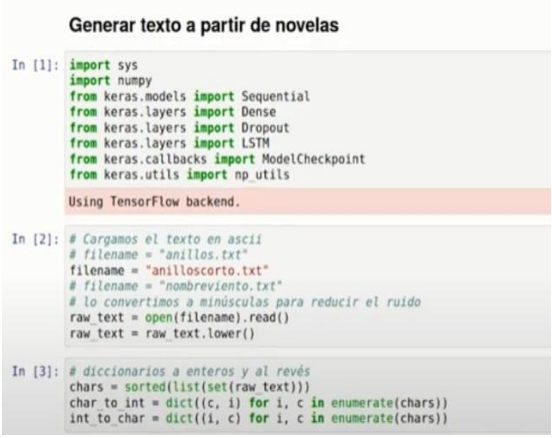

*Figura 4. Caso de estudio: generación de texto a partir de novelas.*

A. *Importación de librerías y lectura de los datos*. Se alimenta el programa con un texto de aproximadamente 5000 líneas en un archivo txt. Se convierte a minúsculas para limitar los tipos de caracteres. El vocabulario incluye además de las letras, signos de puntuación. El número de intervalos temporales o time-step es de 60. Se transforma la lista de secuencias de entrada en la forma: muestras, time-step, características, esperada por la una red LSTM. Posteriormente, se escalan los números enteros al rango 0 a 1 para hacer que los patrones sean más fáciles de aprender ya que se utiliza la función de activación sigmoide por defecto. Los diccionarios transforman los caracteres a números; permiten convertir caracteres a vectores "*one-hot*" y viceversa. Cada valor es un vector con una longitud de 38 elementos, llenos de ceros excepto con un 1 en la columna para la letra particular que el patrón representa. Ver figura 5.

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957
ISSN 2007-1957

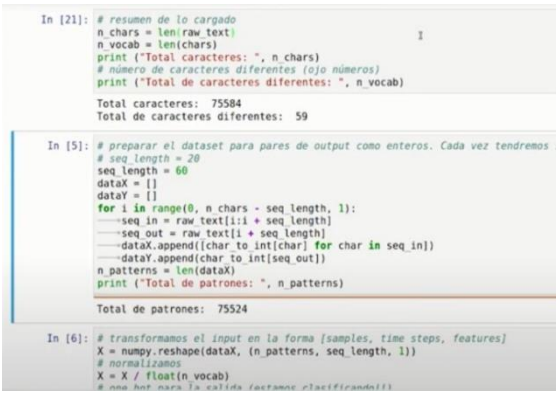


*Figura 5. Fragmento de código: lectura de datos.*

B. *Implementación*. El modelo consta de dos capas: una capa LSTM de 256 neuronas para la entrada que entrega las secuencias y otra capa de 256 neuronas de salida a una función *softmax* que considera las probabilidades e indicará el carácter más probable a generar como predicción. La red utiliza Dropout con probabilidad del 20%. Este caso de estudio es un problema de clasificación de un solo carácter con 38 clases y, como tal, se emplea "ADAM" como algoritmo de optimización y como función de pérdida entropía cruzada. Ver figura 6. Adicionalmente se calculan las expresiones indicadas en las ecuaciones 1 a la 6.

C. *Entrenamiento*. Se lleva acabo el optimizador gradiente descendente calculando las derivadas parciales y aplicando la regla de la cadena. Ver figura 7. Adicionalmente se actualizan los pesos y los gradientes de cada compuerta (*forget, input, output y cell state*), como lo señalan las ecuaciones 7 a la 10. Observar que el algoritmo considera la acumulación total del error luego de calcular el error local en cada elemento de la celda LSTM y de cada time-step.

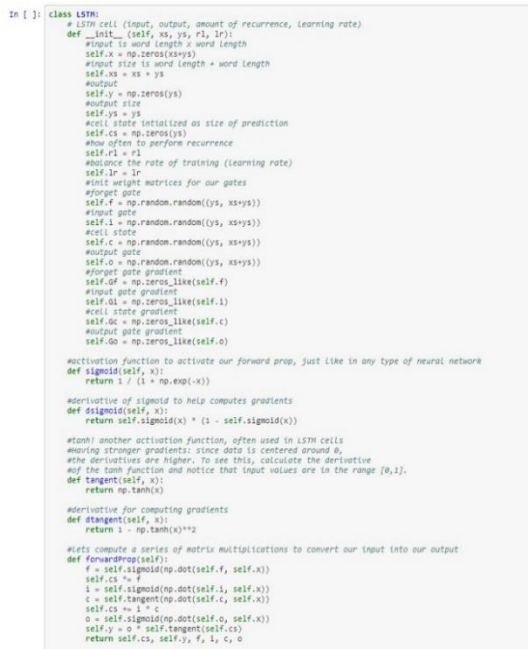


*Figura 6. Fragmento de código: Implementación.*


*Figura 7. Fragmento de código: Entrenamiento.*

**D.** *Generación de palabras*. La forma más sencilla de usar el modelo LSTM de Python para hacer predicciones es comenzar primero con una secuencia de semillas como

Humanidades, Tecnología y Ciencia, del Instituto Politécnico Nacional
Certificado de Reserva de Derechos al Uso Exclusivo del Título: 04-2010-0326124-14000-203 ISSN: 2007-1957
*ISSN 2007-1957*

entrada, generar el siguiente carácter y luego actualizar la secuencia de semillas para añadir el carácter generado al final y recortar el primer carácter. Este proceso se repite mientras se busca predecir nuevos caracteres (por ejemplo, una secuencia de 1.000 caracteres de longitud). Se puede elegir un patrón de entrada aleatorio como semilla y luego imprimir los caracteres generados a medida que se generan. Posteriormente se lleva la activación resultante a la capa *softmax* para así generar la predicción que será un vector que representa la distribución de probabilidad para escoger un elemento aleatoriamente. Finalmente se busca el carácter según el vocabulario y se actualizan las entradas; la predicción generada se convertirá en la entrada a la celda recurrente para el siguiente *time-step*, mientras la activación generada en esta iteración corresponderá con el nuevo estado oculto a usar como entrada en la siguiente iteración. El proceso se repite de forma iterativa produciendo uno a uno los caracteres de la secuencia. Ver figura 8.

```
Seed:
" e los
regalos.
en esta ocasión los regalos fueron desacostum "
bradamente buenos. los
niños hobbits es un enigma; todas las
leyendas e historias familiares lo dan por sabido; durante que meria de
consón cunán cartado curorió el los días de barqos de tipada gesta

start = numpy.random.randint(0, len(dataX)-1)
pattern = dataX[start]
print ("Seed:")
print ("\"", ''.join([int_to_char[value] for value in pattern]), "\"")
# generamos texto
for i in range(150):
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(n_vocab)
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    sys.stdout.write(result)
    pattern.append(index)
    pattern = pattern[1:len(pattern)]

Seed:
"  edad muy respetable para un hobbit
(el viejo tuk había alca "
nzado sólo los ciento treinta; y frodo cumpliría treinta y tres, un número importante:
.
las lenguas empezaron a moverse en h
```

*Figura 8. Fragmento de código: Generación de palabras.*

## IV. Conclusiones

Las celdas LSTM procesan secuencias de longitud variable, son muy útiles en problemas donde existe dependencia de datos entre etapas distantes, permitiendo que la red aprenda patrones contemplando largos periodos de tiempo. La estructura de una red LSTM se caracteriza por contener una serie de compuertas, que operan sobre el flujo principal de la información, de modo que permite seleccionar los datos que son desechables de aquellos que requieren conservarse. El resultado obtenido en el presente caso de estudio mostró un texto compuesto de palabras semejantes a los escritos de Tolkien con una falta de integración en la redacción. Algunas de las palabras en secuencia tienen sentido, pero muchas no lo tienen. Lo cual se explica por el empleo de una red que sólo consta de dos niveles de 256 neuronas. Se logró mejorar los resultados añadiendo más unidades de memoria y más capas. Adicionalmente, se obtuvo una mejor operación al variar los parámetros de entrenamiento, como una mejor regularización (*Dropout*), con lo que se buscó una red más adaptativa y la reducción del *overfit.* Así como el ajuste del tamaño del lote o *batch* y el aumento del número de épocas de entrenamiento.

## 5. Referencias

Bengio, Y., Simard, P., & Fransconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Network*, *5*(2), 157-166. https://doi.org/10.1109/72.279181

Fei-Fei, L., Johnson, J., & Yeung, S. (2017, 20 abril). *Convolutional Neural Networks (CNNs / ConvNets).* Convolutional Neural Networks for

Visual Recognition. Recuperado el 20-02-2020 de: https://cs231n.github.io/convolutional-networks/

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer Publishing.

Karpathy, A. (2015, 21 mayo). *The Unreasonable Effectiveness of Recurrent Neural Networks*. Andrej Karpathy Blog. Recuperado el 08-02-2020 de: http://karpathy.github.io/2015/05/21/rnn-effectiveness

Olah, C. (2015, 27 agosto). *Understanding LSTM Networks*. Colah´s blog. Recuperado el 12-05-2020 de: https://colah.github.io/posts/2015-08-Understanding-LSTMs/