

Deep Learning Specialization

by DeepLearning.AI

Course #5: Sequence Models



[Course Site](#)

Made By: [Matias Borghi](#)

Table of Contents

Summary	3
Week #1: Recurrent Neural Networks (RNN)	4
¿Por qué modelos secuenciales?	4
Notación	4
¿Por qué no utilizar una red neuronal estándar?	6
Backpropagation a través del tiempo	9
Diferentes tipos de RNNs	10
Modelado del lenguaje y generación secuencial	11
Obteniendo oraciones nuevas (Sampling novel sequences)	13
Modelo de lenguaje a nivel carácter	13
Gradientes desvanecientes (vanishing gradients) con RNNs	14
Gated Recurrent Units (GRU)	15
Long Short Term Memory Unit (LSTM)	17
Deep RNNs	19
Week 2: Natural Language Processing & Word Embeddings	21
Introduction to Word Embeddings	21
Representación de palabras	21
Visualización de word embedding usando t-SNE	22
Usando word embedding	23
Transfer learning y word embeddings	23
Propiedades de los word embeddings	23
¿Qué función de similitud se utiliza?	24
Cosine similarity	24
Embedding Matrix	25
Neural language model	25
Modelo Word2Vec	27
Skip-gram model	27
Negative sampling	28
GloVe (Global vectors for word representation) word vectors	30
Applications using Word Embeddings	30
Clasificación sentimental	30
RNN para clasificación sentimental	31
Eliminando el bias de word embeddings	32
Week 3: Sequence models & Attention mechanism	33
Various sequence to sequence architectures	33
Modelo secuencia a secuencia	33
Algoritmo Beam Search	35
Refinamientos al algoritmo de Beam Search	37
Análisis de errores con Beam Search	37
Bleu Score	38

Modelo de atención	40
Reconocimiento del habla (Speech recognition)	43
¿Cómo construimos un modelo de atención para reconocimiento de audio?	43
Costo CTC para reconocimiento de sonido	43

Summary

Week 1 Recurrent Neural Networks

Learn about recurrent neural networks. This type of model has been proven to perform extremely well on temporal data. It has several variants including LSTMs, GRUs and Bidirectional RNNs, which you are going to learn about in this section.

Week 2 Natural Language Processing & Word Embeddings

Natural language processing with deep learning is an important combination. Using word vector representations and embedding layers you can train recurrent neural networks with outstanding performances in a wide variety of industries. Examples of applications are sentiment analysis, named entity recognition and machine translation.

Week 3 Sequence models & Attention mechanism

Sequence models can be augmented using an attention mechanism. This algorithm will help your model understand where it should focus its attention given a sequence of inputs. This week, you will also learn about speech recognition and how to deal with audio data.

Week #1: Recurrent Neural Networks (RNN)

Estos modelos han transformado el reconocimiento del habla, procesamiento natural del lenguaje y otras áreas.

¿Por qué modelos secuenciales?

Primero veamos unos ejemplos de datos secuenciales a través de un modelo supervisado de aprendizaje.

Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation		→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAAC TAG	→	AGCCCCTGTGAGGAAC TAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

Por ejemplo, reconocimiento de audio transcribe un audio a oraciones. Generación musical puede crear música (partituras), clasificación sentimental puede asignar un rating a una oración que rankea una película por ejemplo, análisis secuencial de ADN se encarga de ver que parte de la cadena corresponde a una proteína, traducción es otra de las posibilidades, también se puede a partir de una secuencia de imágenes determinar la acción que se está realizando o determinar a partir de una oración los sujetos de la misma.

Notación

A modo de motivación, supongamos que tenemos el siguiente ejemplo

x: Harry Potter and Hermione Granger invented a new spell.

Como salida del mismo queríamos saber cuáles de esas palabras son caracteres de Harry Potter por ejemplo entonces la salida esperada sería la siguiente, un uno si la palabra corresponde con el objetivo y un 0 si no.

u: | | 0 | | 0 0 0 0

Dado que la entrada posee 9 palabras, la salida posee nueve números, 1 o 0.

Introduciendo terminología, a las entradas las escribiremos como

$x^{<1>} \quad x^{<2>} \quad x^{<3>} \quad \dots \quad x^{<t>} \quad \dots \quad x^{<9>}$

$$T_x = 9$$

Y a la longitud de este vector como

De la misma manera para la salida

$y^{<1>} \quad y^{<2>} \quad y^{<3>} \quad \dots \quad y^{<9>}$

$$T_y = 9$$

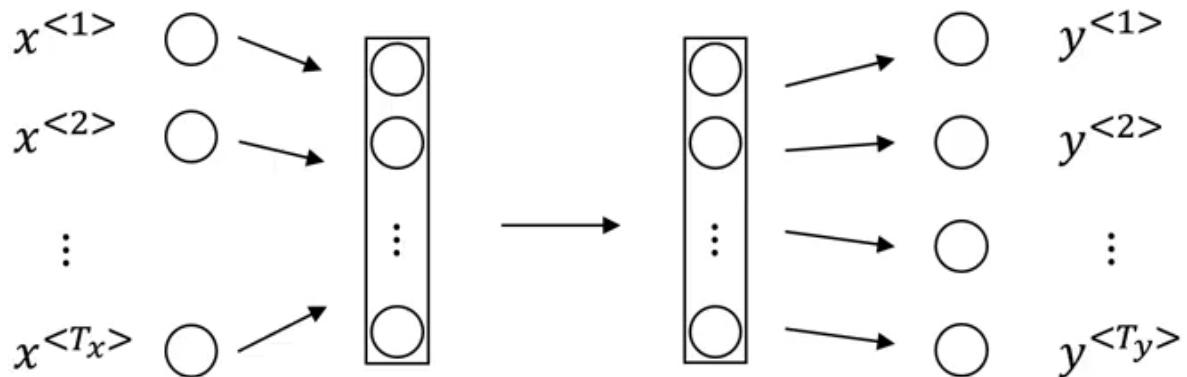
Con sigue . Si hay varias entradas y por ende salidas esto se representa como

$$\begin{array}{l} x^{(i)<t>} \\ y^{(i)<t>} \end{array} \quad T_x^{(i)} = 9 \quad T_y^{(i)}$$

Veamos ahora cómo representar palabras individuales en una oración. Lo primero que debemos hacer es definir un **vocabulario o diccionario**. Esto es una lista de palabras que se utilizaran en las representaciones. Un ejemplo de esto se muestra en la figura de la derecha junto con las posiciones que ocupan en el array. En este caso son 10 mil palabras, pero usualmente se utilizan entre 30 y 50 mil, aunque algunas aplicaciones llegan a tener hasta 100 mil palabras. Dicho esto, una manera de representar palabras en una oración sería a través del método one-hot. El cual es un array de 0 o 1s del tamaño del vocabulario que indica con un uno la posición de la palabra en el vocabulario. Por ejemplo, si la palabra es Harry habrá un 1 en la posición 4075 del vector. Por último, en el caso de que haya una palabra que no esté en el vocabulario, hay un espacio reservado denominado `<unkw>` de unknown o desconocida.

a	1
aaron	2
:	:
and	367
:	:
harry	4075
:	:
potter	6820
:	:
zulu	10,000

¿Por qué no utilizar una red neuronal estándar?

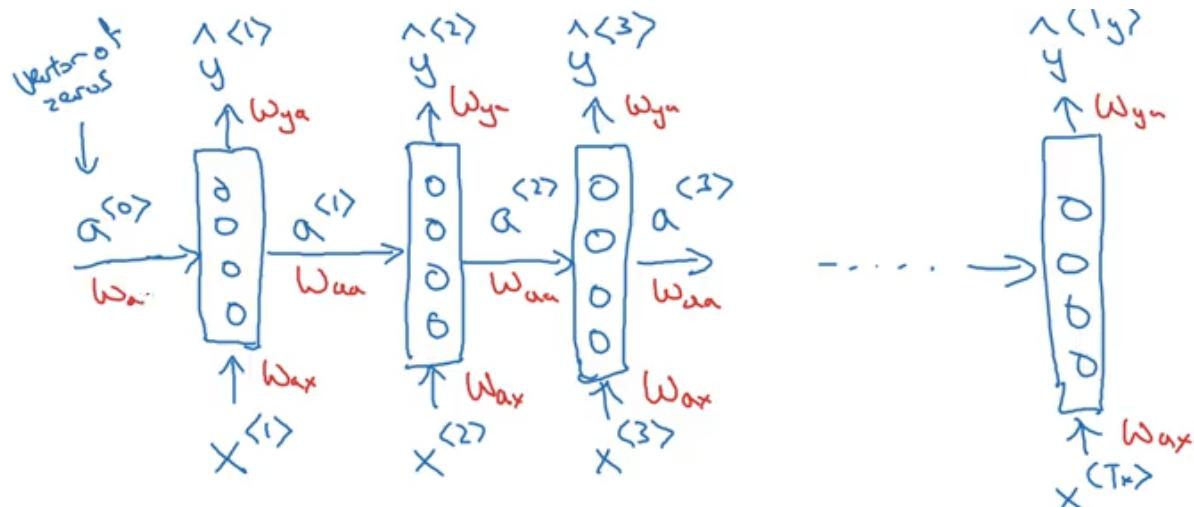


Haciendo uso del problema explicado anteriormente, dada una oración podemos ir viendo palabra por palabra (Cada una de las 9 de la oración de ejemplo; con viendo nos referimos a que sea un input de la red) y analizar la salida que será un 0 o 1 de acuerdo a si la misma representa un nombre de persona. Pero esto posee dos problemas principales:

- Las entradas y salidas pueden ser de distintas longitudes para distintos ejemplos.
- No se comparten características aprendidas a lo largo de las distintas posiciones del texto; Por ejemplo, si Harry aparece en la primera posición de la oración y se aprende que es un nombre de persona y luego aparece en alguna otra parte entonces la red deberá aprender todo de nuevo y no se hizo uso de lo aprendido previamente.

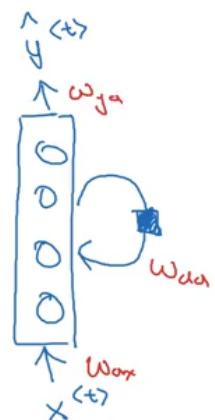
Dado estos problemas utilizaremos **redes neuronales recurrentes (RNN)**. A continuación, veremos que son.

La primera palabra se alimentará con una capa de una red neuronal tratando de predecir si la misma es un nombre de persona o no. Similarmente con la segunda palabra, pero la activación es alimentada del proceso anterior. Lo mismo es hecho para el resto de las palabras que componen la oración. El diagrama de esta red neuronal recurrente se muestra en la figura inferior.



Esta misma red se suele simplificar en términos de la notación como sigue. Habrá, a su vez, un conjunto de parámetros asociados para cada time step que gobiernan las conexiones desde la entrada a la red neuronal oculta.

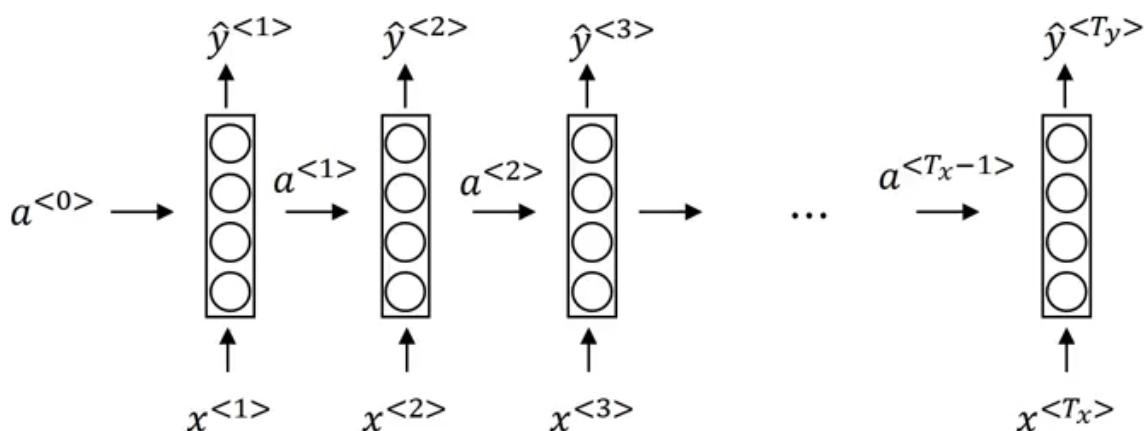
Una de las limitaciones de esta red es que no usa información de la oración posterior. Es decir, en el ejemplo siguiente no es posible determinar si la palabra *Teddy* forma parte de un nombre o no a partir de las primeras tres letras, mientras que si se leyera la oración completa se sabría que en el primer caso si forma parte y en el segundo no. Las redes neuronales que se encargan de este tema se llaman **Bidirectional Recurrent Neural Networks (BRNN)**.



He said, "Teddy Roosevelt was a great President."

He said, "Teddy bears are on sale!"

A partir de una imagen limpia de una RNN, veamos los cálculos que hace la misma. Empecemos con los cálculos de forward propagation



Como se dijo anteriormente la primera activación es un vector de ceros.

$$a^{<0>} = \vec{0}.$$

Mientras que la siguiente activación y la primera salida se calculan como sigue, donde g es una **función de activación**. Como dice la imagen, para las activaciones las funciones pueden ser tanto Tanh como ReLUs, mientras que, para las salidas, funciones sigmoides.

$$\begin{aligned} a^{<1>} &= g_1(w_{aa} a^{<0>} + w_{ax} x^{<1>} + b_a) \leftarrow \tanh \text{ / ReLU} \\ \hat{y}^{<1>} &= g_2(w_{ya} a^{<1>} + b_y) \leftarrow \text{sigmoid} \end{aligned}$$

En cuanto a la terminología utilizada para los pesos w , con dos subíndices, el primero hace referencia a la variable que se quiere calcular, mientras que el segundo a la variable sobre la cual es multiplicada.

$$a \leftarrow W_a x^{<t>} \quad \begin{matrix} \uparrow & \uparrow \\ w_{ax} & x^{<t>} \end{matrix}$$

De manera general, las funciones de activación y la salida se calculan como sigue

$$a^{<t>} = g(W_{aa}a^{<t>} + W_{ax}x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

Pero antes de continuar simplifiquemos la notación de las mismas. Comprimiendo los primeros dos términos de la primera ecuación.

$$a^{<t>} = g(W_a [a^{<t-1>} ; x^{<t>}] + b_a)$$

En la ecuación superior se introdujeron dos términos. Las dos matrices W se concatenaron en una sola de la siguiente manera.

$$\boxed{\begin{bmatrix} W_{aa} & ; & W_{ax} \end{bmatrix}} = W_a \quad (100, 10000)$$

Poniendo unas dimensiones la matriz, si la activación a tiempo $t-1$ posee dimensión 100 y la entrada a tiempo t 10mil, entonces W_{aa} debe tener dimensión $(100,100)$ y W_{ax} $(100,10000)$. Uniendo las dos matrices se obtendrá una matriz W_a con dimensión $(100,10100)$.

$$g(\boxed{W_{aa}} a^{<t-1>} + \boxed{W_{ax}} x^{<t>} +$$

$\uparrow \quad \uparrow$
 $(100,100) \quad (100,10000)$

Por otra parte, la terminología de corchetes introducida para multiplicar W_a es la siguiente: (donde los términos simplificados se muestran en rojo).

$$[a^{(t-1)}, x^{(t)}] = \begin{pmatrix} a^{(t-1)} \\ x^{(t)} \end{pmatrix}$$

↑ 100
↓ 10000
↑ 10100

$$[w_{aa}; w_{ax}] \begin{pmatrix} a^{(t-1)} \\ x^{(t)} \end{pmatrix} = w_{aa} a^{(t-1)} + w_{ax} x^{(t)}$$

Por otra parte, la ecuación para el cálculo de la salida se computa de la siguiente manera

$$\hat{y}^{(t)} = g(w_y a^{(t)} + b_y)$$

Backpropagation a través del tiempo

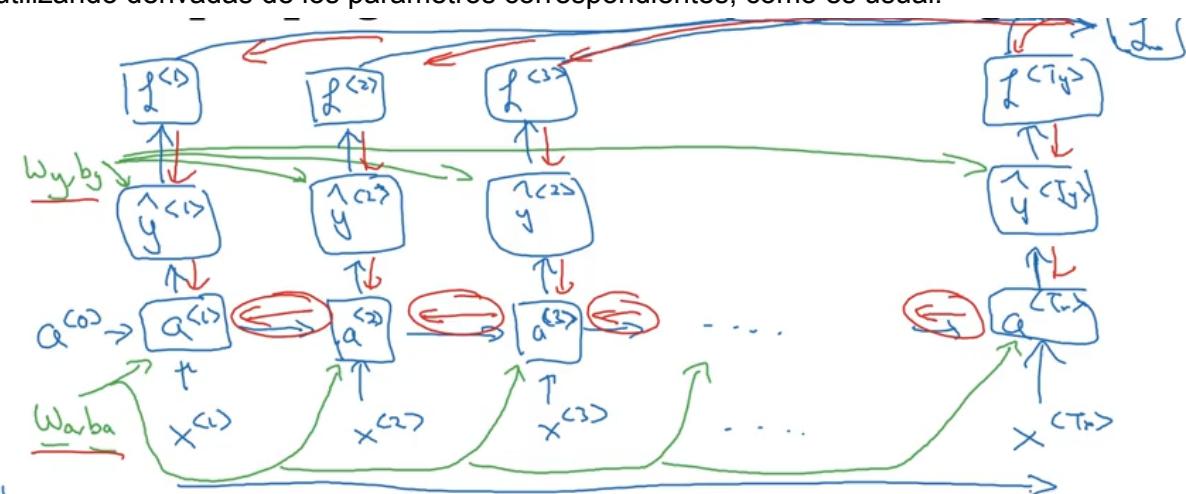
Para calcular el backpropagation, hay que calcular la función de coste. La misma se calcula a tiempo t como la regresión logística usual.

$$\underline{\mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)})} = -y^{(t)} \log \hat{y}^{(t)} - (1-y^{(t)}) \log (1-\hat{y}^{(t)})$$

Mientras que la función de coste total es la suma de estas pérdidas.

$$\underline{\mathcal{L}(\hat{y}, y)} = \sum_{t=1}^T \mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)})$$

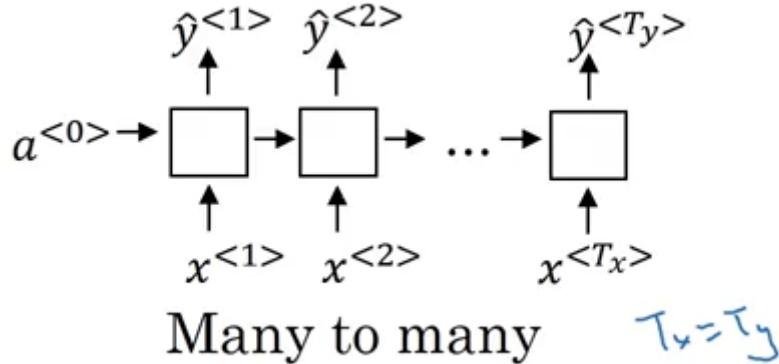
El cálculo de backpropagation (y la función de coste) se muestra en la figura siguiente en rojo. Luego del cálculo de cada salida se calcula la función de coste correspondiente a tiempo t. Calculadas todas estas se modifican los parámetros a través de backpropagation utilizando derivadas de los parámetros correspondientes, como es usual.



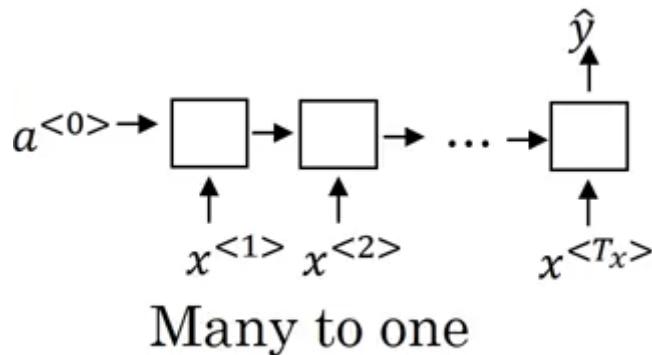
Diferentes tipos de RNNs

A continuación veremos otros tipos de RNNs que incluyen tener un número de entradas distintas al número de salidas (T_x distinto a T_y). Un ejemplo de esto puede ser cuando se mostraron ejemplos de datos secuenciales, al realizar traducciones, para generación musical o clasificación sentimental (ranking de películas a partir del review).

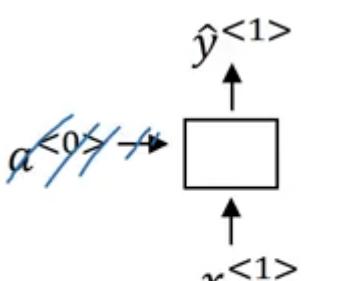
La red neuronal vista previamente donde se cumple que $T_x = T_y$ se la llama **Many-to-many**.



Mientras que en el caso de clasificación sentimental donde se requiere una salida, el tipo de red utilizado es **Many-to-one**.

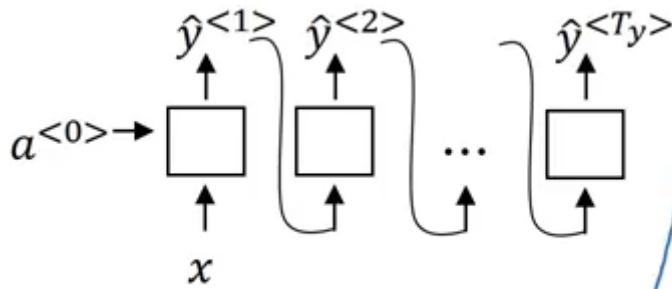


Por una cuestión de completitud también se tiene la red **one-to-one** que sería la red neuronal estándar, explicada en los dos primeros cursos de la especialización.



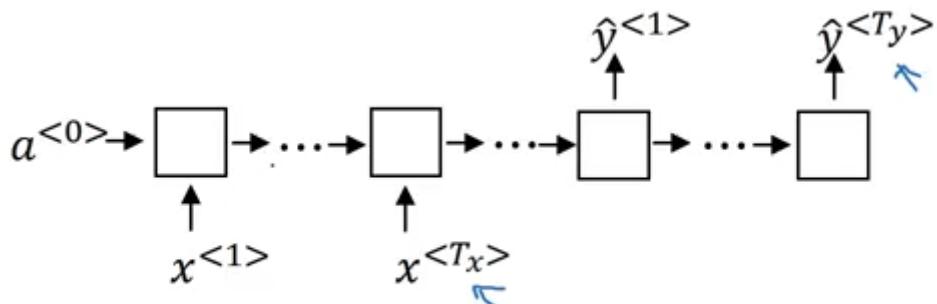
Por otra parte, la *generación musical* se puede realizar con redes del tipo **one-to-many**.

Music generation $x \rightarrow y^{<1>} y^{<2>} \dots y^{<Ty>}$ $x = \emptyset$ One-to-many

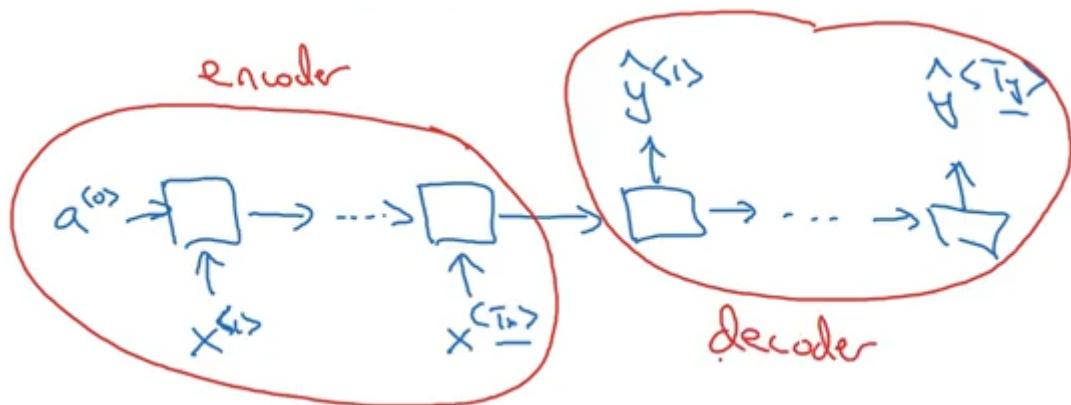


One to many

Todavía queda ver un caso particular de many-to-many en por ejemplo, *traducciones donde la entrada puede tener distinta longitud que la salida.*



Many to many



Modelado del lenguaje y generación secuencial

Dada por ejemplo una oración que suena similar a otra en reconocimiento del habla. Para determinar si se dijo una frase o la otra, se suele utilizar el contexto de la frase teniendo en cuenta que una es más probable que la otra. En el ejemplo siguiente, ¿se dijo *pair* o *pear*? Suenan ambas de manera muy similar.

The apple and pair salad.

The apple and pear salad.

Un modelo de lenguaje interpretará que lo más probable es que se haya dicho la segunda oración dado que es más probable que la primera.

$$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$$

$$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$$

Es decir, lo que el modelado del lenguaje hace es determinar la cantidad

$$P(y^{(1)}, y^{(2)}, \dots, y^{(T)})$$

Pero, ¿cómo se modela un idioma con una RNN para determinar estas probabilidades?

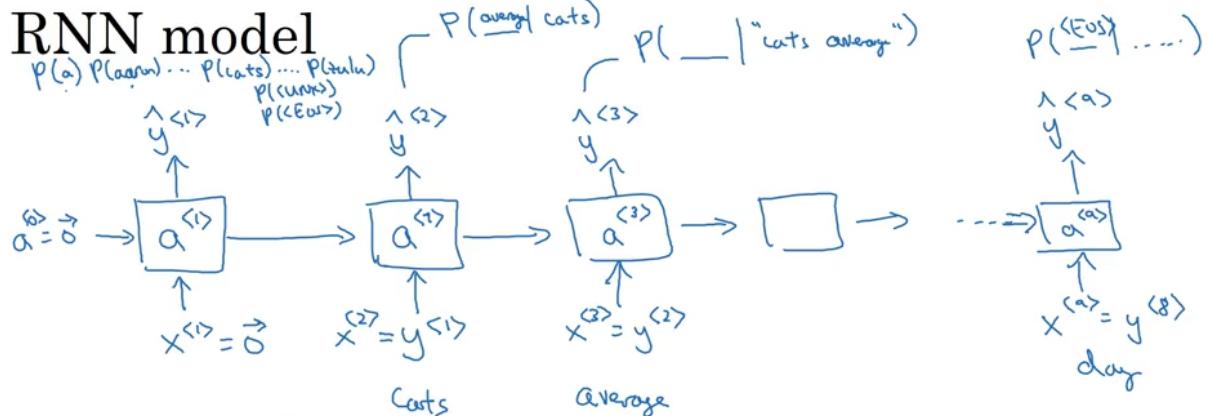
Training set: un texto grande de texto en el idioma.

Dada una oración que se quiere medir la probabilidad

Cats average 15 hours of sleep a day. <EOS>
y⁽¹⁾ y⁽²⁾ y⁽³⁾ ... y⁽ⁿ⁾

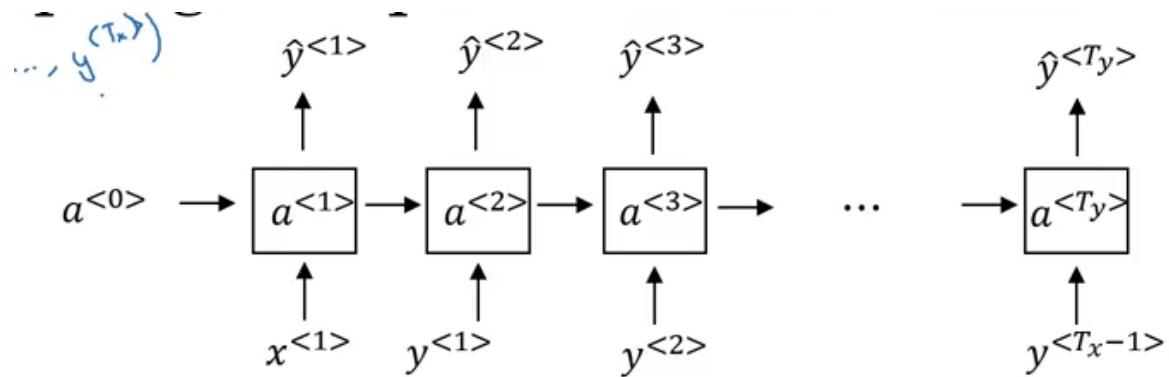
Donde se puede agregar un end-of-sentence al final. Si hay una palabra que no está presente en el vocabulario, se puede reemplazar por el token <unk>.

The Egyptian Mau is a bread of cat. <EOS>
<UNK>



Obteniendo oraciones nuevas (Sampling novel sequences)

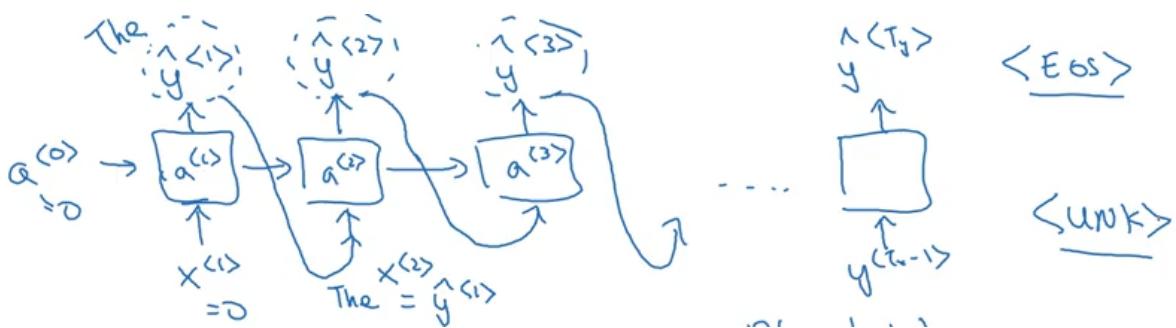
Dada la siguiente RNN, lo que queremos hacer es muestrear oraciones originales a partir de una distribución de probabilidad $P(y^{(1)}, \dots, y^{(T_y)})$



Aunque en realidad, la red neuronal presentada anteriormente se utiliza para entrenar la red. **Para samplear se utiliza una modificación de la misma.** De manera aleatoria se hace como muestra la figura inferior. Seteando a_0 y x_1 a 0, se calcula una palabra aleatoria con la cual empezar la oración o se elige una en particular

$p(a)p(a|a_0)\dots p(zulu)p(<\text{UNK}>)$ n.p. random.choice

Luego esta alimenta el paso siguiente de la red para determinar la siguiente palabra y así sucesivamente. Esta iteración termina hasta que se llegue a un EOS o hasta que uno decida si se quiere un número determinado de palabras. Por otra parte, si se obtiene un UNK, se puede volver a muestrear hasta obtener una palabra distinta.



Modelo de lenguaje a nivel carácter

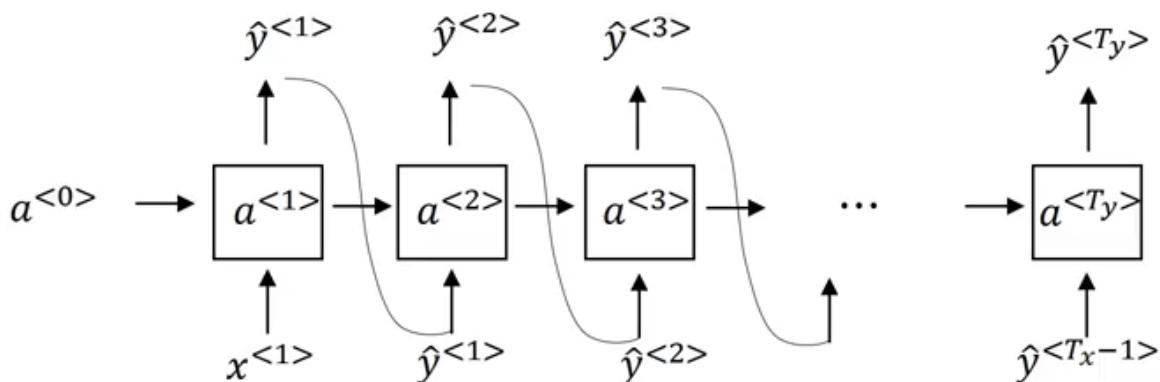
En lugar de construir un vocabulario de palabras como se hizo anteriormente

Vocabulary = [a, aaron, ..., zulu, <UNK>]

Se puede construir un vocabulario de caracteres.

$$\text{Vocabulario} = [a, b, c, \dots, z, \cup, ., ., ;, 0, \dots, 9, A, \dots, Z]$$

De esta manera usando un modelo de este tipo para la oración anterior (“Cats average 15 hours...”) se obtendría que $c=y_1, a=y_2$, etc. Usando modelado de lenguaje a nivel caracteres no deberíamos preocuparnos por UNK. A la palabra Mau se le asignaría una probabilidad distinta de cero. Mientras que si no estuviera esta palabra en el vocabulario en el modelado a nivel de palabras, a esta se le asignaría un UNK. Una de las mayores **desventajas** de este modelo es que se terminan con oraciones mucho más largas. Muchas oraciones en inglés poseen entre 10 y 20 palabras, pero tendrán decenas de caracteres. Entonces el modelo a nivel carácter podría no capturar el rango de dependencias a largo alcance. A su vez, son computacionalmente más caras de entrenar.



Un ejemplo de modelado de lenguaje de nivel carácter se muestra a continuación. La primera columna fue obtenida a partir de artículos de noticias mientras que la segunda de textos de Shakespeare.

News

President enrique peña nieto, announced
sench's sulk former coming football langston
paring.

“I was not at all surprised,” said hich langston.

“Concussion epidemic”, to be examined.

The gray football the told some and this has on
the uefa icon, should money as.

Shakespeare

The mortal moon hath her eclipse in love.
And subject of this thou art another this fold.
When lesser be my love to me see sabl's.
For whose are ruse of mine eyes heaves.

Gradientes desvanecientes (vanishing gradients) con RNNs

Empecemos con un ejemplo para entender las dependencias de largo alcance en oraciones. Si dice gato el verbo es singular, si dice gatos, debería ir plural.

The cat, which already ate ..., was full
 The cats, wh ... were full

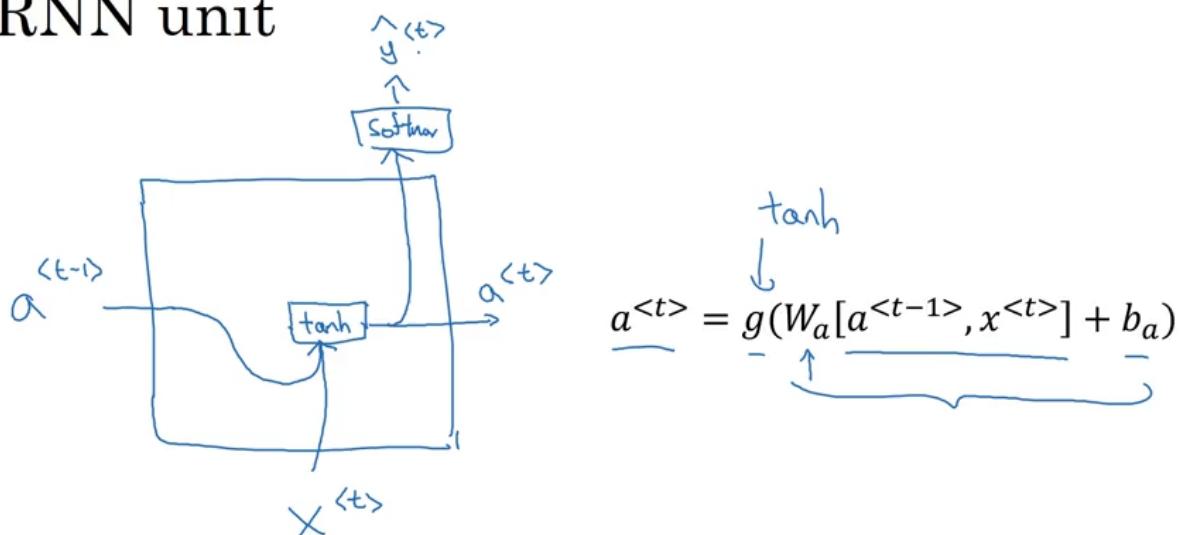
La RNN que vimos hasta ahora tiene una influencia local. Es decir, las palabras se ven afectadas principalmente por las palabras cercanas.

Recordemos que también se puede ver el efecto de gradientes que “explotan” al ver que hay parámetros con NaNs, aplicando gradient clipping, re escalar vectores de gradientes para que no haya overflow. En el caso de vanishing gradient debemos introducir **Gated Recurrent Units (GRU)** para introducir dependencias de largo alcance.

Gated Recurrent Units (GRU)

Recordemos que una unidad de RNN viene definida de la siguiente manera, y usemos la misma para explicar el concepto siguiente.

RNN unit



El siguiente tema está basado en los siguientes trabajos:

- Cho, Kyunghyun, et al. "On the properties of neural machine translation: Encoder-decoder approaches." *arXiv preprint arXiv:1409.1259* (2014)
- Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:1412.3555* (2014).

GRU units van a tener una nueva variable, la célula de memoria, C. a tiempo t. Esta poseerá el valor igual, por ahora, a la activación.

$$\underline{C}^{<t>} = \underline{a}^{<t>}$$

En cada paso temporal, consideraremos reemplazar C con un valor C tilde,

$$\tilde{C}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$$

La idea importante de GRU es que se considerara actualizar o no el memory cell de acuerdo a otra variable gamma.

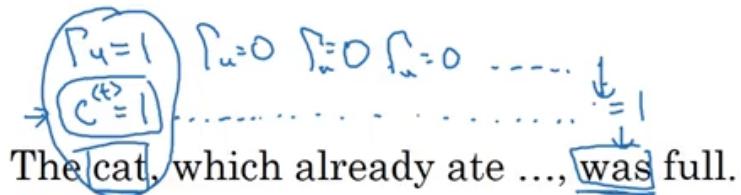
$$\Gamma_u = \sigma(W_u [c^{<t-1>}, x^{<t>}] + b_u)$$

↑ "update"

Por lo cual esta actualización viene dada de la siguiente manera

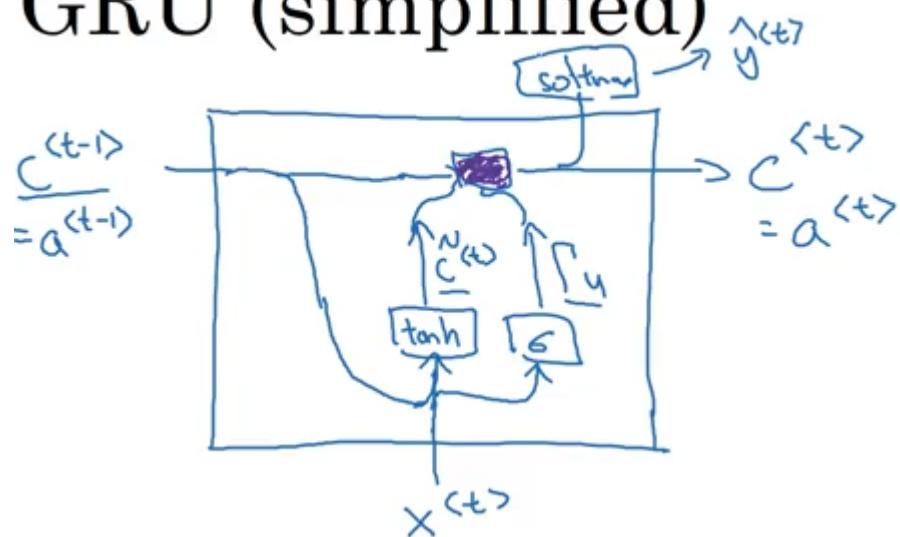
$$\tilde{c}^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

Mediante el ejemplo de la oración que estamos viendo esto se ve de la siguiente manera



Un esquema simplificado de esta compuerta se muestra a continuación

GRU (simplified)



Haciendo un resumen de las ecuaciones y reescribiendo c tilde con Γ_r (relevancia) llegamos a las GRU completas (**Full GRU**)

$$\tilde{c}^{<t>} = \tanh(W_c [\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u [c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r [c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

Long Short Term Memory Unit (LSTM)

GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\underline{\Gamma_u} = \sigma(W_u[\underline{c}^{<t-1>}, x^{<t>}] + b_u)$$

$$\underline{\Gamma_r} = \sigma(W_r[\underline{c}^{<t-1>}, x^{<t>}] + b_r)$$

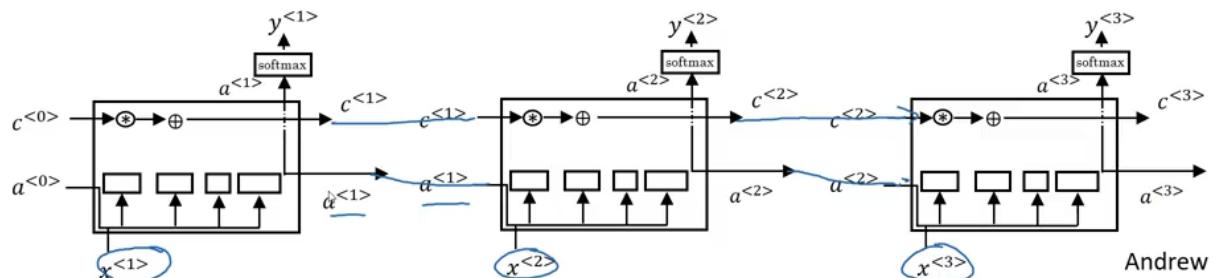
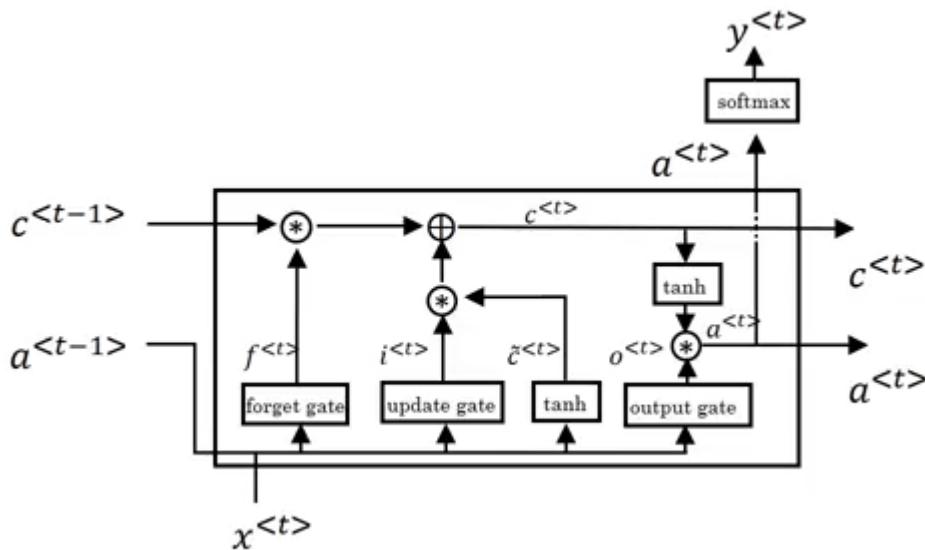
$$\underline{c^{<t>}} = \underline{\Gamma_u} * \underline{\tilde{c}^{<t>}} + \underline{(1 - \Gamma_u)} * \underline{c^{<t-1>}}$$

$$\underline{a^{<t>}} = \underline{c^{<t>}}$$

Este texto está basado en la siguiente investigación

- [Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 \(1997\): 1735-1780.](#)

$$\begin{aligned}\tilde{c}^{<t>} &= \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \\ \Gamma_u &= \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \\ \Gamma_f &= \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \\ \Gamma_o &= \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \\ c^{<t>} &= \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \\ a^{<t>} &= \Gamma_o * \tanh c^{<t>}\end{aligned}$$

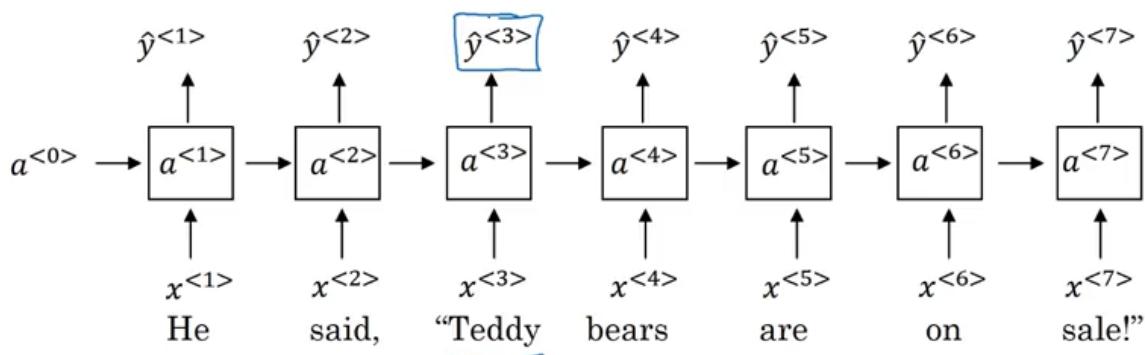


Para construir modelos más poderosos faltan introducir dos conceptos. Uno es **RNNs bidireccionales** que permite en un punto del tiempo tomar información de antes y después en la secuencia. Por otra parte, **Deep RNNs** se verán luego.

A modo de motivación, no hay manera de saber si Teddy es un nombre de persona o no conociendo las primeras tres palabras, independientemente de si las unidades de la RNN son GRU o LSTM.

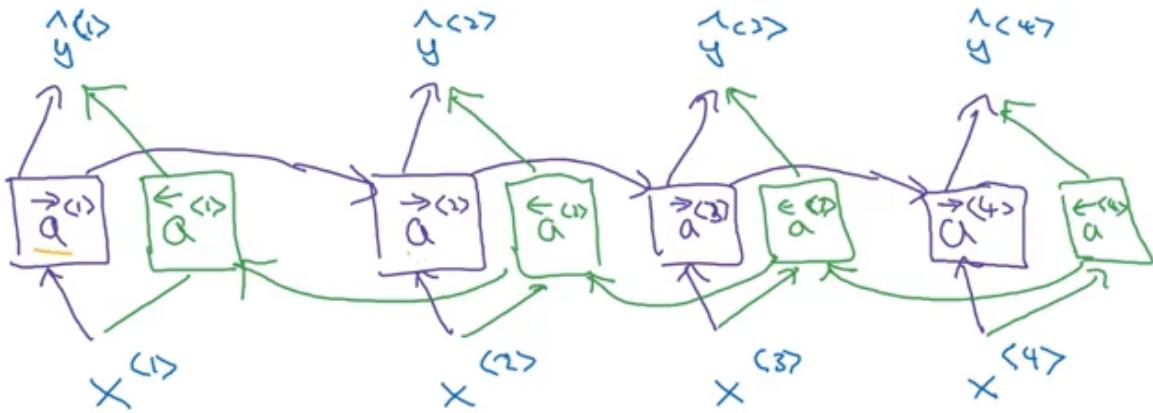
He said, "Teddy bears are on sale!"

He said, "Teddy Roosevelt was a great President!"



Para construir una red bidireccional, se insertan activaciones que mueven el flujo de información en sentido derecha-izquierda, que a modo de distinguirlas con las previas se les

inserta una flecha horizontal indicando el sentido de flujo de la información. Entonces el gráfico resultante es uno acíclico. La salida queda determinada por ambas activaciones, una que tiene en cuenta la información pasada y otra que tiene en cuenta la futura. Notemos que **todos los cálculos son realizados en forward propagation**.



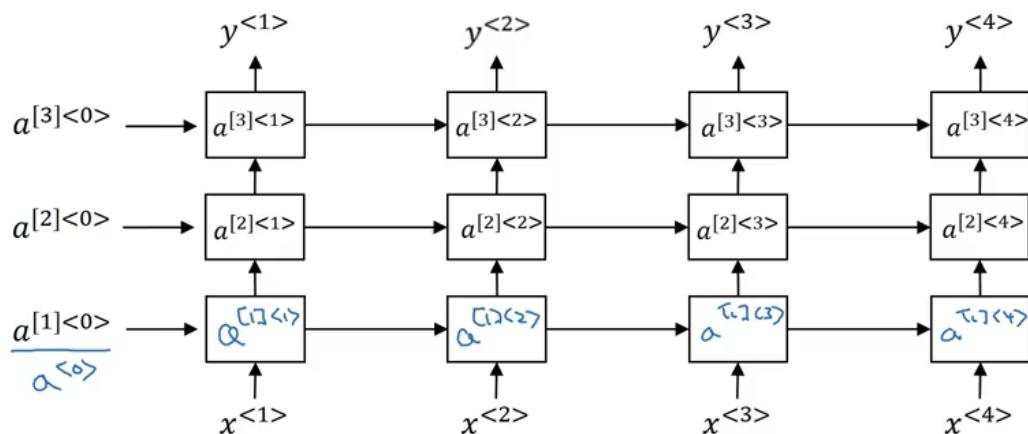
El cálculo de la salida puede expresarse entonces como

$$\hat{y}^{<1>} = g(W_y[\vec{a}^{<1>}, \vec{a}^{<2>}] + b_y)$$

Como desventaja, este método necesita tener toda la secuencia determinada para poder procesar la información. Por ejemplo, en speech recognition habría que esperar a que la persona termine de hablar.

Deep RNNs

Para lograr procesamientos más completos de datos secuenciales a veces es necesario acumular varias unidades de RNNs para determinar una salida en particular.



Donde se introdujo una nueva notación a las activaciones donde $[l]$ significa en número de capas y $<t>$ el tiempo. En el caso de la imagen superior se muestra una RNN profunda con tres capas.

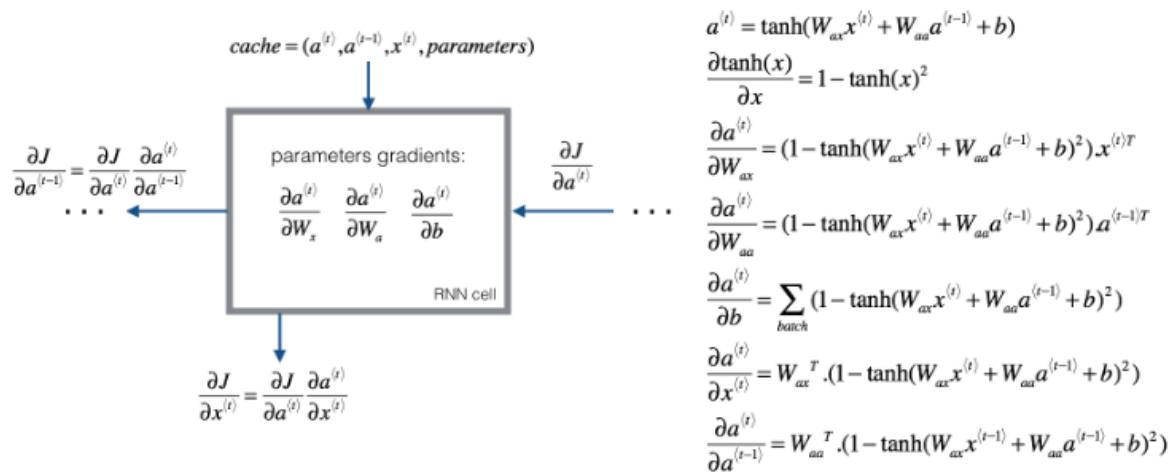
$a^{[l]} < t >$

A modo de ejemplo, $a_{23}^{(2)} \leftarrow g(W_a^{(2)} [a_{12}^{(2)}, a_{23}^{(2)}] + b_a^{(2)})$

$$a_{23}^{(2)} \leftarrow g(W_a^{(2)} [a_{12}^{(2)}, a_{23}^{(2)}] + b_a^{(2)})$$

A parte de este método se pueden agregar redes ocultas que no estén conectadas entre sí horizontalmente.

A modo de completitud, las ecuaciones correspondientes a **backpropagation** son las siguientes



Week 2: Natural Language Processing & Word Embeddings

Introduction to Word Embeddings

Es una manera de entender relaciones entre palabras, como hombre-mujer, rey-reina. Esto nos permitirá desarrollar aplicaciones de NLP.

Representación de palabras

Palabras representadas por un vocabulario y el método de one-hot vector. Las desventajas de este método es que las palabras son tratadas por separado y no existe una correlación entre las mismas.

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$

Para esto se les asignan **características** a las palabras, como género, royal, edad, etc. con números entre -1 y 1, donde cercanos a uno indican una correlación fuerte y un 0 no relacionado entre sí.

Por ejemplo, manzana y naranja comparten prácticamente valores similares para las características. Esto da a pensar que, si queremos completar las oraciones mostradas abajo, entonces si terminan con dos palabras que son similares, es muy probable que se completen con la misma, en este caso juice (jugo).

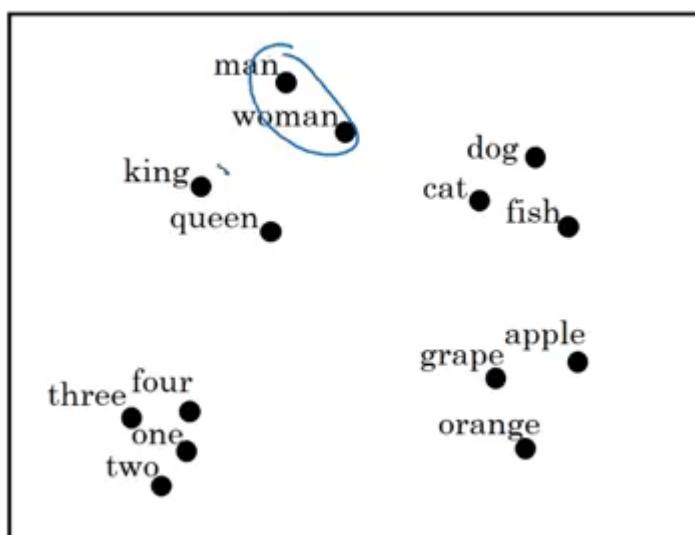
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.62	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
size cost alive verb	⋮ e_{5391}	⋮ e_{9853}			I want a glass of orange ____.	
					I want a glass of apple ____.	Andrew Ng

Visualización de word embedding usando t-SNE

Esta sección está basada en el siguiente trabajo:

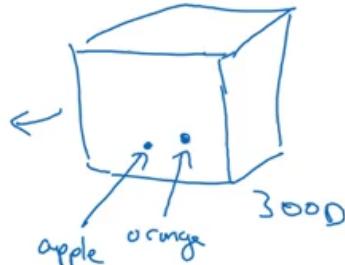
- [Maaten, Laurens van der, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9.Nov \(2008\): 2579-2605.](#)

La técnica llamada **t-SNE** permite visualizar datos en un número alto de dimensiones al asignarle a cada punto un punto del espacio 2D o 3D.

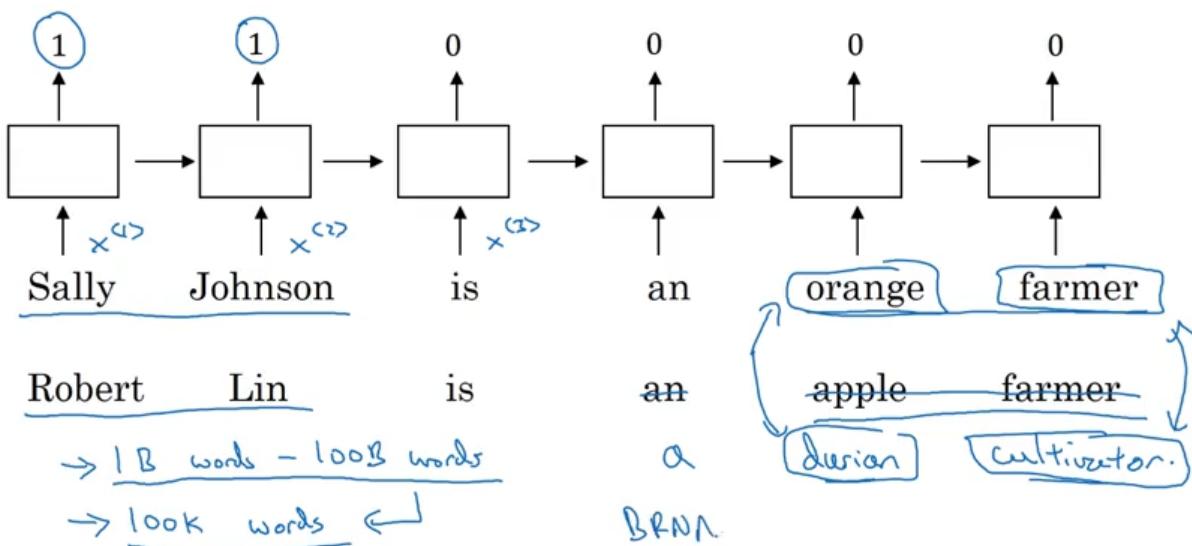


Vemos en la figura de la izquierda que hombre-mujer tienden a estar agrupados juntos. Lo mismo sucede para rey-reina o animales. Esto mismo sucede para números o fruta. A su vez los seres vivientes suelen estar agrupados más cerca entre sí que aquellos que no lo son.

Usando word embedding



Veamos cómo usar los word embeddings para aplicaciones en NLP. Por ejemplo, si sabemos que Sally Johnson es el nombre de un orange farmer, entonces se podría determinar que Robert Lin es el nombre de un apple farmer debido a los features representation. Pero si en lugar de ser apple farmer es durian cultivator, identificar el nombre ya es más complicado porque es probable que no se pueda utilizar feature representation para esto. Una manera de solucionar este inconveniente podría ser el de utilizar una base de datos ya entrenada de 1B-100B de palabras en lugar de la que uno podría disponer más pequeña de 100k. Esto sería Transfer Learning usando word embeddings.



Transfer learning y word embeddings

- Aprender word embeddings desde un texto largo de billones de palabras o descargar un word embedding online pre-entrenado.
- También se pueden transferir word embeddings a nuevas tareas donde se tienen sets de entrenamiento mucho más chicos. En lugar de utilizar un one-hot vector de 10 mil posiciones se puede utilizar un Dense vector de 300D.
- Opcional: continuar ajustando word embeddings con nueva información.

Word embedding ha sido útil para tareas donde se tienen base de datos pequeñas.

Propiedades de los word embeddings

Si sabemos que hombre-mujer están relacionados, ¿entonces se puede determinar con qué palabra se corresponde al rey, por ejemplo? Una manera de determinar esto es restando las columnas de embeddings y ver que son similares.

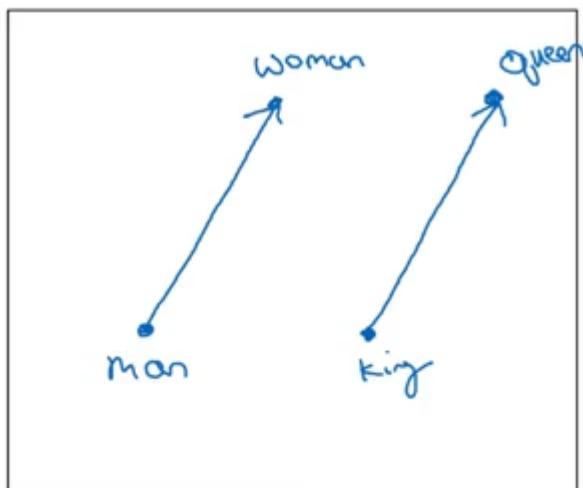
Esto fue mencionado por primera vez en el trabajo:

- [Mikolov, Tomas, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2013.](#)

$$\begin{array}{c} e_{\text{man}} \\ e_{\text{woman}} \end{array} \xrightarrow{\text{Man} \rightarrow \text{Woman}} \begin{array}{c} e_{\text{man}} - e_{\text{woman}} \\ e_{\text{king}} - ? \end{array}$$

$$e_{\text{man}} - e_{\text{woman}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$e_{\text{king}} - e_{\text{queen}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



300 D

¿Qué función de similitud se utiliza?

$$\text{Sim}(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$

Cosine similarity

Correr un word embedding sobre un texto de gran longitud puede provocar que se obtengan relaciones entre las mismas como las siguientes

Man:Woman as Boy:Girl

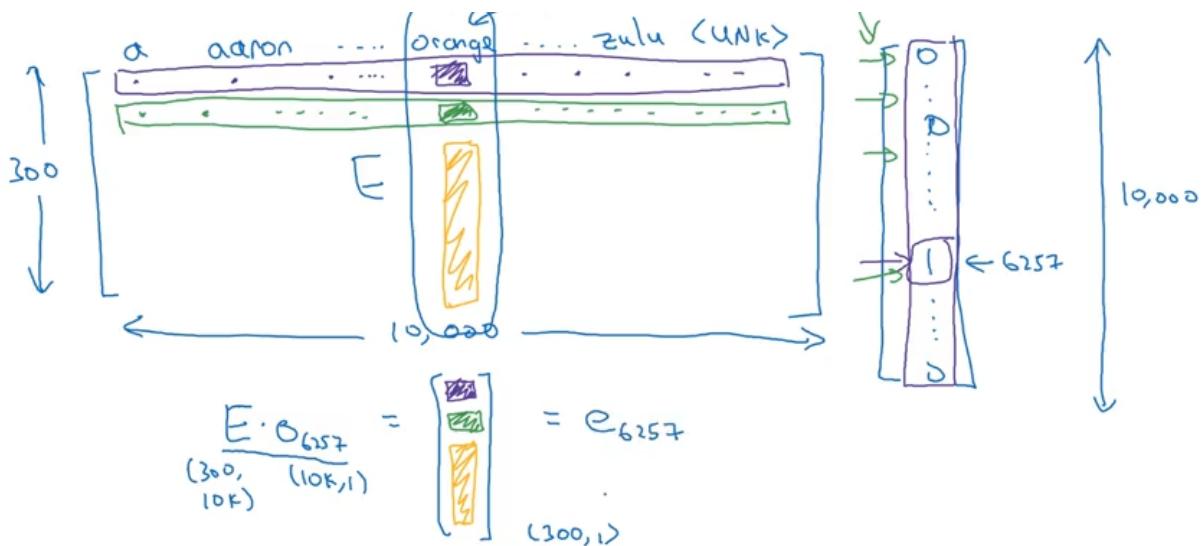
Ottawa:Canada as Nairobi:Kenya

Big:Bigger as Tall:Taller

Yen:Japan as Ruble:Russia

Embedding Matrix

Cuando entrenamos una red para obtener un word embedding lo que se hace es obtener una matriz embebida. Esta matriz posee dimensión igual a *#palabras en el vocabulario * # de features*. Al multiplicarla por cualquier one-hot vector se obtiene un vector de features correspondiente a la palabra a la cual el one-hot se corresponde.



De manera general, se obtiene entonces

$$E \cdot \mathbf{o}_j = e_j$$

↓
= embedding for word j

Neural language model

Este modelo fue desarrollado a partir del trabajo:

- [Bengio, Yoshua, et al. "A neural probabilistic language model." Journal of machine learning research 3.Feb \(2003\): 1137-1155.](#)

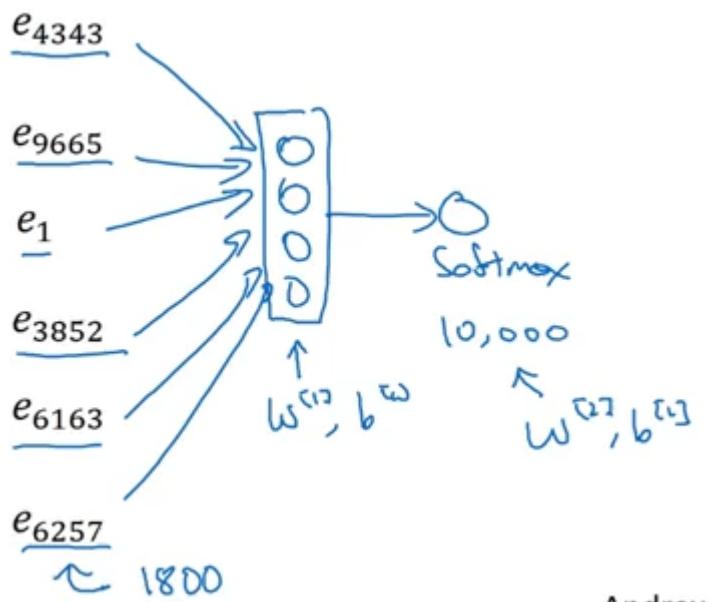
Modelo para predecir la siguiente palabra en la secuencia. Por ejemplo, dada la siguiente oración, queremos saber cómo continuarla.

I	want	a	glass	of	orange	_____
4343	9665	1	3852	6163	6257	

Primero creamos los vectores de embedding

I	o_{4343}	\longrightarrow	E	\longrightarrow	$e_{\underline{4343}}$
want	o_{9665}	\longrightarrow	E	\longrightarrow	$e_{\underline{9665}}$
a	o_1	\longrightarrow	E	\longrightarrow	$e_{\underline{1}}$
glass	o_{3852}	\longrightarrow	E	\longrightarrow	$e_{\underline{3852}}$
of	o_{6163}	\longrightarrow	E	\longrightarrow	$e_{\underline{6163}}$
orange	o_{6257}	\longrightarrow	E	\longrightarrow	$e_{\underline{6257}}$

Estos vectores son la entrada de una red neuronal conectada a un Softmax del tamaño del vocabulario prediciendo la palabra necesaria. Si cada vector embobido posee un tamaño de 300, entonces $300 \times 6 = 1800$, será el tamaño de la entrada. El esquema de esta red se representa en la red siguiente



También podemos tratar de predecir la palabra utilizando únicamente una parte de la oración, lo cual disminuiría el número de entradas y parámetros de la red.

Veamos otro contexto. Definiendo contexto de acuerdo a las palabras que anteceden/prosiguen a la palabra a determinar. Por ejemplo, podría ser las 4 palabras previas, las 4 a la izquierda y las 4 a la derecha, la palabra anterior o, algo que funciona bastante bien, 1 palabra cercana, llamado esto **skip-gram**.

4 words on left & right

a glass of orange to go along with

Last 1 word

orange ?

Nearby 1 word

glass ?

Modelo Word2Vec

Este modelo fue desarrollado en el siguiente paper:

- [Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 \(2013\).](#)

Skip-gram model

Este modelo crea un par de datos supervisados de la siguiente manera: una palabra que marca el contexto se determina aleatoriamente. Por otra parte, la palabra de target/objetivo se crea al azar también hasta un cierto número de palabras al azar hacia adelante o hacia atrás del contexto. Dada la oración "*I want a glass of orange juice to go along with my cereal*", unos ejemplos. De pares de embeddings son:

Context

orange

orange

orange



Target

juice

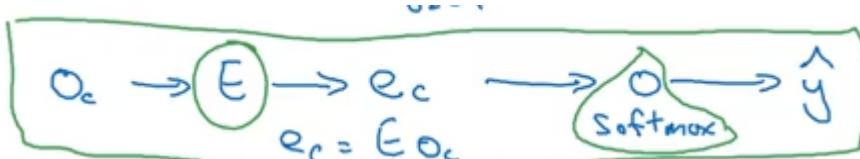
glass

my



La idea del paper es aprender buenos word embeddings, no solucionar un problema de aprendizaje supervisado correctamente.

Resumiendo, dado un vocabulario de 10k palabras, la red neuronal será de la siguiente manera



Donde Softmax viene dada por

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

Donde theta se corresponde con los parámetros asociados con el target t. El costo se calcula como

$$L(\hat{y}, y) = - \sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

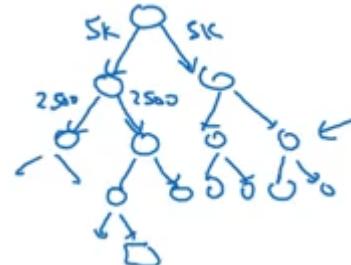
Donde y se corresponde con el one-hot vector del target.

$$y = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow 4834$$

Andrew N
("juia")
4834

Content c ("orange") \rightarrow Target t

Aunque, este modelo presenta un inconveniente al calcular Softmax. El denominador requiere hacer una suma sobre el vocabulario completo. Con 10 mil palabras no hay problema, pero como un vocabulario más grande la sumatoria se vuelve costosa computacionalmente. Una solución a esto podría ser utilizar un clasificador **Softmax Hierarchical** que nos dice primero si el target está en las primeras 5 mil palabras o en las últimas 5. En la práctica las palabras más comunes están al principio del árbol, mientras que las menos comunes se encuentran en la parte más profunda del mismo.



¿Cómo se muestrea el contexto c? Dado esto el target simplemente se muestrea dentro de una ventana de esta palabra. Si muestreamos de manera uniforme lo más probable es que aparezcan palabras frecuentes todo el tiempo como the,of,a, and,to,... En lugar de esto hay algunas técnicas heurísticas que se pueden utilizar.

Negative sampling

Este trabajo fue presentado en el siguiente paper:

- [Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.](#)

El gran problema de skip-gram es lo que tarda Softmax. Este algoritmo trata de que este proceso sea más eficiente.

<u>Context</u>	<u>word</u>
orange	juice
orange	king
orange	book
orange	the
orange	of

target?

1
0
0
0
0

La idea es crear un método supervisado más eficiente.

La idea es muestrear un contexto y un target y determinar si las mismas están asociadas o no, es decir si son un par valido de contexto-target.

Dado un par de palabras, ¿estas poseen target 1 o 0? Así se genera el training set.

Si consideramos k como el número de repeticiones por palabras (filas) en las que repetimos cada contexto. Se suele utilizar k entre 5 y 20 para base de datos pequeñas y k entre 2 y 5 para base de datos grandes.

Describamos el modelo de aprendizaje supervisado para aprender a mapear desde x a y.

El training set vendrá dado por

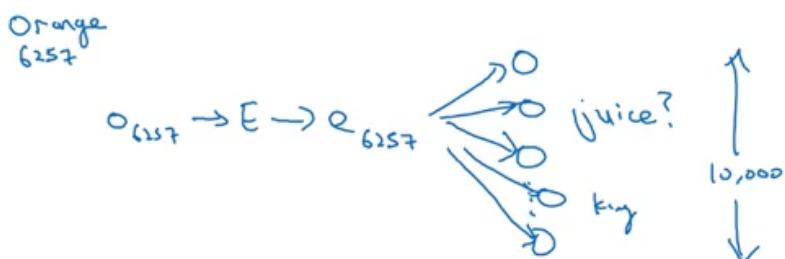
<u>context</u>	<u>word</u>	<u>target?</u>
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

Para estimar la probabilidad de que $y = 1$, definimos

$$P(y=1 | c, t) = \sigma(\theta_t^T e_c)$$

En lugar de entrenar el Softmax con 10 mil palabras vamos a entrenarlas con 5 ($k+1$, k número de negativos)

Pero, ¿cómo seleccionamos los ejemplos negativos? Se toma un valor heurístico. Un valor inversamente proporcional (con una potencia) al número de veces que aparece.



$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10,000} f(w_j)^{3/4}}$$

GloVe (Global vectors for word representation) word vectors

Este trabajo fue obtenido del siguiente trabajo:

- [Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing \(EMNLP\). 2014.](#)

Antes: pares de palabras cercanas. En cambio, GloVe mide el número de veces que la palabra j aparece en el contexto de i, X_{ij} .

$\hat{J} = \sum_{i,j} f(X_{ij})(w_i^T \tilde{w}_j - \log X_{ij})^2$. El modelo GloVe pretende minimizar la siguiente función de coste donde la terminología parece confusa pero w transpuesta por w tilde es lo que nosotros definimos previamente como θ transpuesta por el vector de embedding e. Por otra parte, $f(x_{ij})$ se define por el hecho de que si X_{ij} es cero entonces el log del mismo diverge y, para evitar esto, f se define igual a cero en esa situación. Para mayor información respecto a formas funcionales de f, dirigirse al paper.

Applications using Word Embeddings

Clasificación sentimental

Veamos cómo aplicar lo que vimos hasta ahora. Un ejemplo de clasificación sentimental podría ser como sigue

x	y
The dessert is excellent.	★★★★★☆
Service was quite slow.	★★☆☆☆
Good for a quick meal, but nothing special.	★★★☆☆
Completely lacking in good taste, good service, and good ambience.	★☆☆☆☆

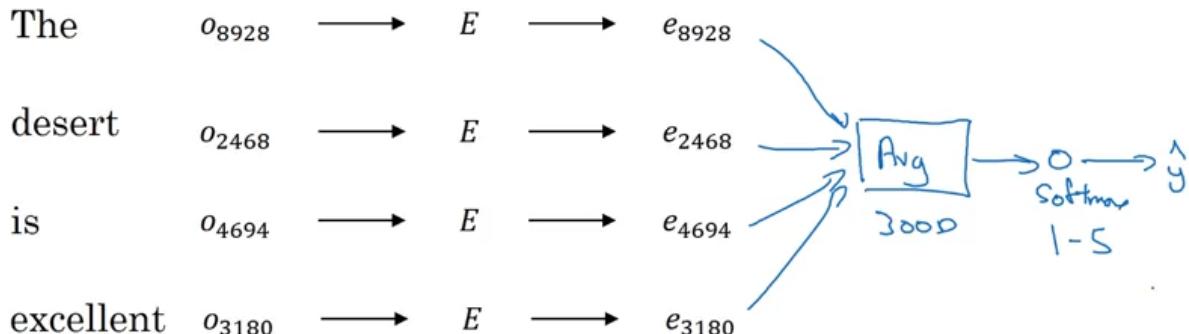
El problema con la clasificación sentimental es que no se cuenta con un dataset tan grande. Los trainings set para este tipo de tareas son de entre 10 mil a tal vez 100 mil palabras.

Un modelo de clasificación sentimental simple sería como sigue. Dado el siguiente review

The dessert is excellent
8928 2468 4694 3180



Se puede agarrar cada one-hot vector de la palabra, la matriz E de embedding entrenada sobre 100B de palabras, y así obtener el vector de embedding, el cual se alimenta a una red neuronal que promedia los valores de rating y se obtiene la salida de Softmax entre 1 y 5 que corresponden al rating.



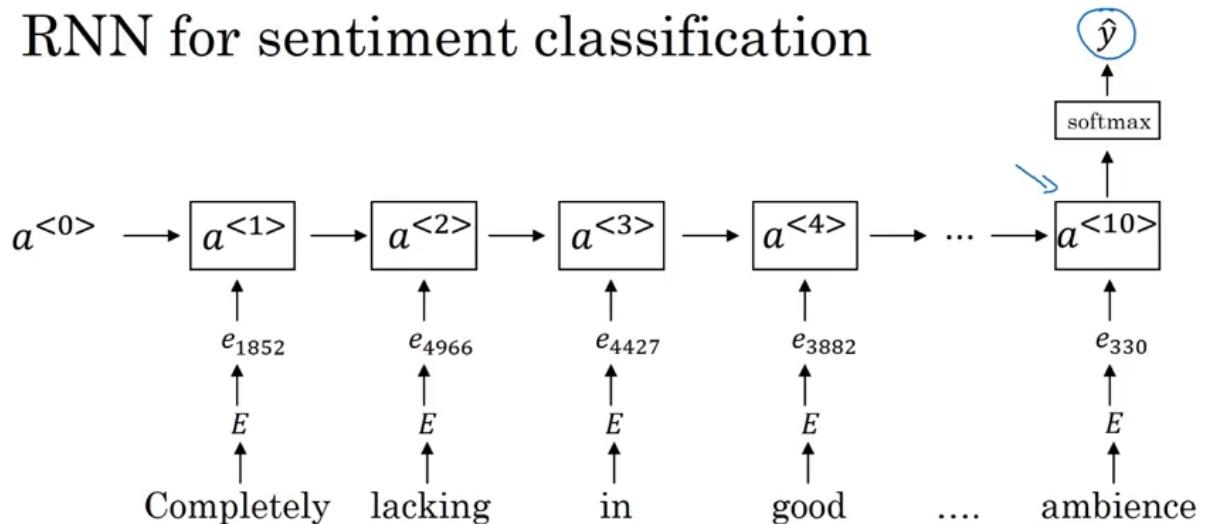
Desventaja: este algoritmo ignora el orden de las palabras entonces podemos tener un review negativo con muchas palabras positivas y se obtendrán resultados erróneos.

“Completely lacking in good taste, ~~good~~ service, and ~~good~~ ambience.”

RNN para clasificación sentimental

Un ejemplo de many-to-one RNN se muestra a continuación, el cual hará un review mejor que el método presentado anteriormente.

RNN for sentiment classification



Eliminando el bias de word embeddings

A continuación se mostrarán técnicas para eliminar el bias de word embeddings. **Con bias no nos estamos refiriendo a bias-variance, sino al bias de género, etnicidad, etc.** Para entender esto basta con mirar el título del paper que trató este tema: "Man is to Computer Programmer as Woman is to Homemaker?".

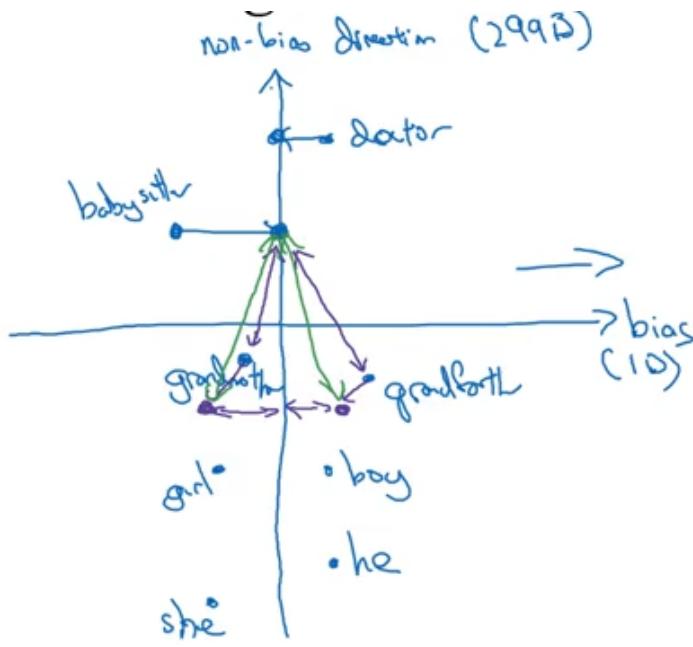
Man:Computer_Programmer as Woman:Homemaker

Father:Doctor as Mother:Nurse

Este tema fue tratado en el paper que se muestra a continuación:

- [Bolukbasi, Tolga, et al. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." Advances in Neural Information Processing Systems. 2016.](#)

Veamos cómo eliminar el bias con un caso particular de bias de género. Esto necesita 3 pasos.



El primero sería **identificar la dirección del bias**. Por ejemplo, restar los pares de palabras de aquellas que están relacionadas, por ejemplo, he-she, male-female y promediar.

Segundo, cada palabra que no esté definida en los pares anteriores se debe proyectar para eliminar el bias. Esto se conoce como **neutralizar**. Por ejemplo, como sucede en el caso de babysitter y doctor.

Tercero y último, **equilibrar los pares**. Por ejemplo, teniendo en cuenta, grandmother y grandfather o girl-boy, para el caso de babysitter está más cerca del lado femenino que masculino, entonces para eliminar este bias lo que se suele utilizar es hacer que los pares estén equidistantes la dimensión sobre la cual no debería haber bias.

Week 3: Sequence models & Attention mechanism

Various sequence to sequence architectures

Modelo secuencia a secuencia

Las ideas que se van a presentar fueron obtenidas del paper

- [Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.](#) Y [Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 \(2014\).](#)

Supongamos que queremos traducir una oración de francés a inglés.

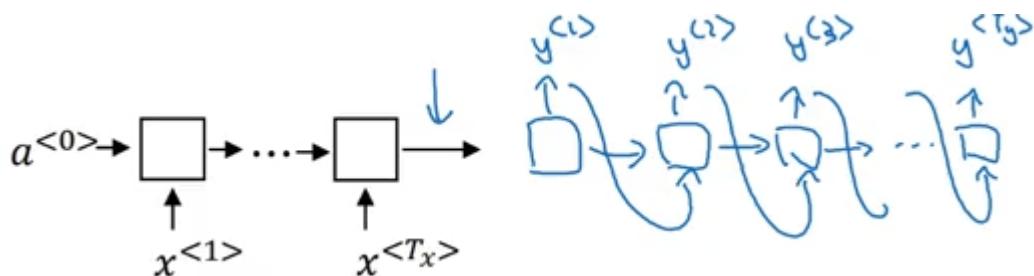
$$x^{<1>} \quad x^{<2>} \quad x^{<3>} \quad x^{<4>} \quad x^{<5>}$$

Jane visite l'Afrique en septembre

→ Jane is visiting Africa in September.

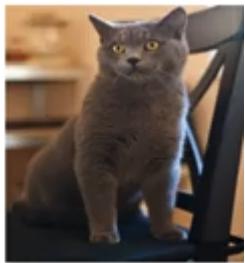
$$y^{<1>} \quad y^{<2>} \quad y^{<3>} \quad y^{<4>} \quad y^{<5>} \quad y^{<6>}$$

Consideremos primero una parte de una red, llamada **encoder** con entradas dadas por la oración en francés conectada la salida a otra red llamada **decoder** cuya salida es la oración en inglés. Este método ha demostrado funcionar relativamente bien.



A su vez, se puede utilizar **Image Captioning** usando RNN. Esto es, dada una imagen describirla. Esto fue obtenido de los siguientes trabajos

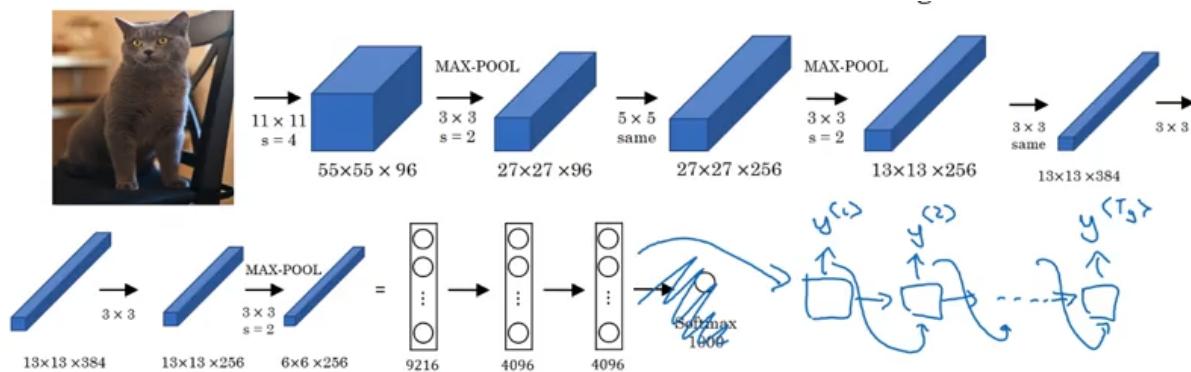
- [Mao, Junhua, et al. "Deep captioning with multimodal recurrent neural networks \(m-rnn\)." arXiv preprint arXiv:1412.6632 \(2014\).](#)
- [Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.](#)
- [Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.](#)



Por ejemplo, dada la siguiente imagen queríamos obtener una oración como la mostrada.

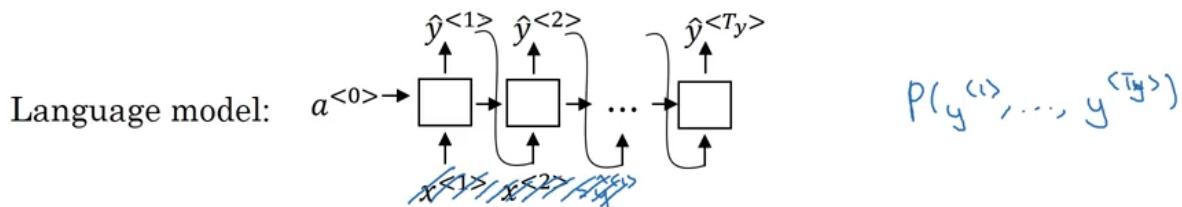
$y^{<1>} y^{<2>} y^{<3>} y^{<4>} y^{<5>} y^{<6>}$
A cat sitting on a chair

Para ello alimentamos la imagen de un gato a una arquitectura del tipo AlexNet, pero en lugar de la función Softmax final, lo que hacemos es conectarle un decoder como vimos anteriormente.

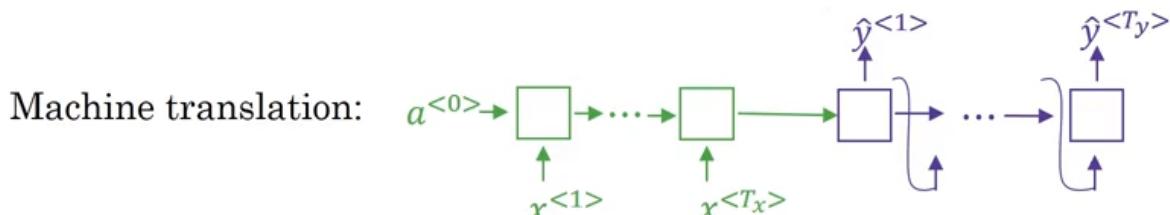


Veamos ahora cómo poder determinar en el caso de realizar una traducción, hay una diferencia entre generar texto como vimos antes, que es una oración traducida aleatoriamente elegido, sino que queremos la mejor traducción.

Básicamente, una traducción se puede pensar como construir un modelo de lenguaje condicional. Esto es, un language model como vimos se generaba como sigue dada una distribución de probabilidad.



Por otra parte, la traducción viene dada por la estructura siguiente



Donde se puede mirar que la parte púrpura coincide con la del modelo de lenguaje, entonces ambos son similares, pero en lugar de empezar con un vector de ceros, se empieza con un decorder. Es por ello que se la llama a la traducción modelo de lenguaje condicional, porque la distribución de probabilidad que genera está basada en una oración de entrada en otro idioma, en este caso, francés.

$$P(y<1>, \dots, y<Ty> | x<1>, \dots, x<Tx>)$$

Entonces, ¿Cómo encontramos la traducción más adecuada? No queremos muestrear oraciones al azar. Podemos encontrarnos con oraciones que estén relativamente bien pero otras que no.

Jane is visiting Africa in September.

Jane is going to be visiting Africa in September.

In September, Jane will visit Africa.

Her African friend welcomed Jane in September.

$$\arg \max_{y^{<1>} \dots y^{<T_y>}} P(y^{<1>} \dots y^{<T_y>} | x)$$

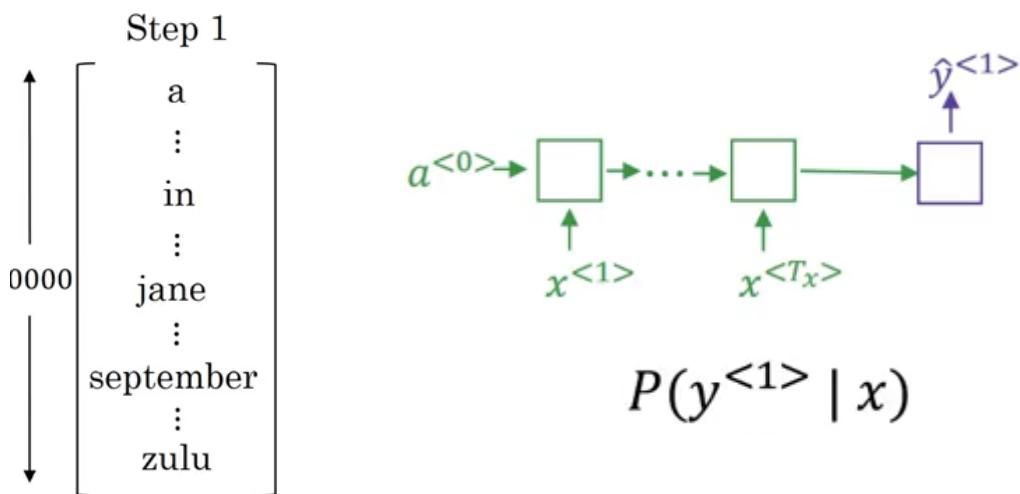
Lo que queremos hacer es encontrar la oración que maximice la probabilidad condicional. El algoritmo que hace esto se llama **BeamSearch**.

Pero, ¿Por qué no utilizar **greedy search**? Este algoritmo es similar al anterior. Lo que hace es *tratar de encontrar la mejor primera palabra, luego la mejor segunda y así sucesivamente*. Considerando que el algoritmo eligió 'Jane is' como las primeras dos palabras de la traducción lo más probable es que para la tercera elija *going* en lugar de *visiting* dado que es más usual en el inglés, y acabaría eligiendo una con probabilidad mayor que es como queremos pero que no tiene mucho sentido dado que la primera en este caso es más adecuada.

$$P(\text{Jane is going} | x) > P(\text{Jane is visiting} | x)$$

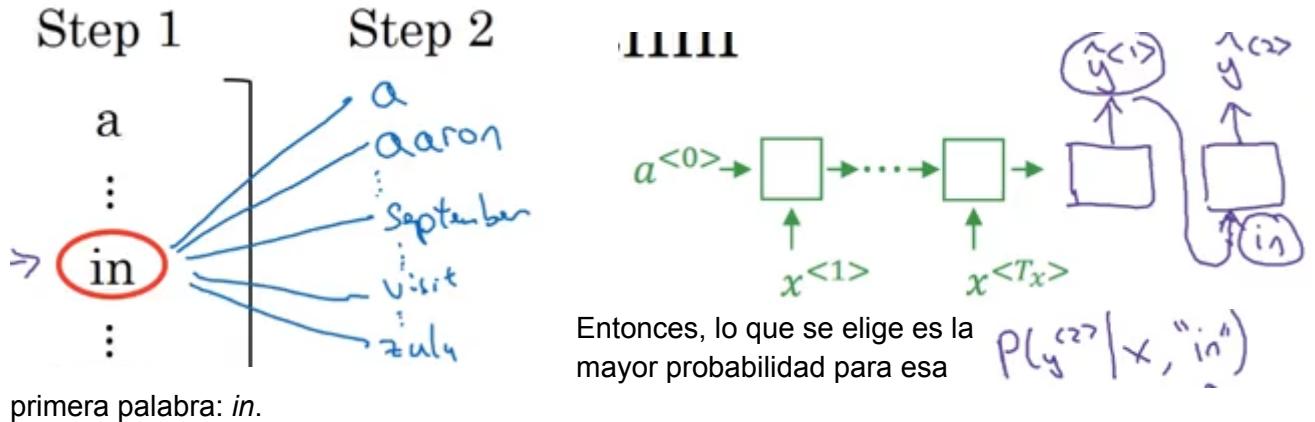
Otra desventaja de esto es que dado un vocabulario de 10 mil palabras, si se quiere formar una oración de 10 letras acabaríamos sampleando $(10 \text{ mil})^{10}$ muestras diferentes lo cual es un número bastante grande. Es por ello que un algoritmo de búsqueda aproximada es recomendado en estos casos.

Algoritmo Beam Search



Este algoritmo lo que hace es determinar primero, dado como input la oración a traducir, las mejores B palabras, esto es las B palabras con mayor probabilidad como primera palabra de la traducción. Este parámetro B se lo llama **Beamwidth**, que puede valer por ejemplo B=3. Un caso, podría ser que haya elegido *In*, *Jane* y *September* como mejores tres palabras.

Como segundo paso, hay que elegir la segunda palabra, para cada una de las anteriores.

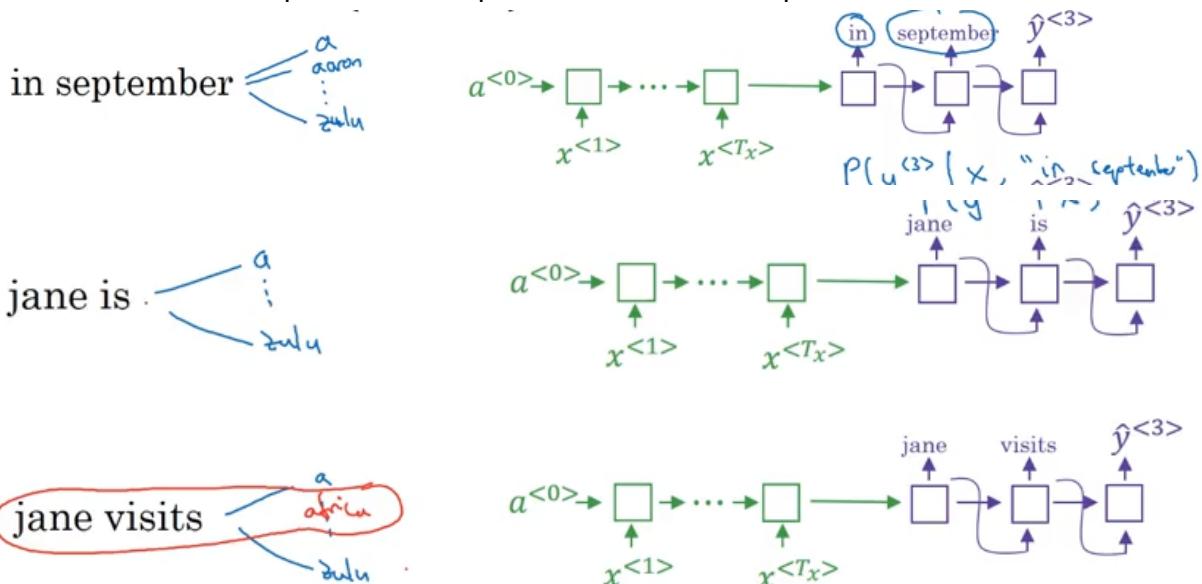


Pero el objetivo final del segundo paso sería calcular la probabilidad del par entre la primera y segunda palabras sea mayor

$$P(y^{<1>} | x) = P(y^{<1>} | x) P(y^{<2>} | x, y^{<1>})$$

En total deberíamos testear en este paso, $30\text{mil} = 3 \times 10 \text{ mil}$ palabras. Si continuamos con B=3, podríamos tener por ejemplo *in-september*, *jane-is* y *jane-visits*, como los pares más probables, esto descartaría *september* como primera palabra.

Similarmente, esto se puede realizar para obtener la tercera palabra.



Afortunadamente, esto terminaría hasta que lo más probable sea un EOS.

jane visits africa in september. <EOS>

Refinamientos al algoritmo de Beam Search

- **Normalización de longitud:** como vimos anteriormente la oración final elegida va a ser la que maximiza las probabilidades condicionales

$$\arg \max_y \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

Pero dado que estas son un número pequeño, terminamos teniendo un *underflow*. Para evitar esto se *maximiza en su lugar el logaritmo de la productoria* lo que resulta en calcular el máximo de

$$\boxed{\arg \max_y \sum_{y=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})}$$

Resulta que hay que hacer una pequeña modificación al realizar la traducción dado que el algoritmo suele preferir oraciones cortas, dado que *mientras más palabras se agreguen más chica será la probabilidad*. Para ello se multiplica por un factor

$$\frac{1}{T_y^\alpha}$$

Dónde $\alpha = 0.7$ elegido heurísticamente. Si α es 1 tenemos una normalización total y si es uno no hay normalización.

- Otras cuestiones a discutir: el **tamaño del ancho del beam (B)**. si es más grande se obtendrán mejores resultados, pero será más lento. Por el contrario, si es pequeño se obtendrán peores resultados, pero será más rápido. Los valores típicos de B son 10, 100, mil y 3 mil. Como **desventaja**, este modelo de Beam Search *no está garantizado que encuentre el máximo a diferencia de los algoritmos no aproximados como BFS o DFS*.

Análisis de errores con Beam Search

Ahora veremos cómo detectar si el beam search algorithm o el RNN el que posee errores. Para esto consideremos el siguiente ejemplo, dada una entrada consideremos una salida y^* que escribiría una persona e \hat{y} la obtenida por el algoritmo.

Jane visite l'Afrique en septembre.

Human: Jane visits Africa in September. (y^*)

Algorithm: Jane visited Africa last September. (\hat{y})

beam Entonces si queremos mirar si el error proviene del RNN (encoder-decoder) o del Beam Search, podríamos empezar aumentando el parámetro de búsqueda B. Pero aumentar este

valor, como así también obtener más training data pueden no llegar a ser una solución. ¿Cómo sabemos entonces si el error proviene del algoritmo de beam search?

Lo que se puede hacer es considerar las probabilidades de error $P(y^*|x)$ y $P(\hat{y}|x)$.

Comparemos ambas, pueden suceder dos cosas

- Primer caso: que $P(y^*|x) > P(\hat{y}|x)$. En este caso beam search eligió \hat{y} , pero y^* posee una probabilidad más alta. Entonces beam search posee el problema.
- Segundo caso: $P(y^*|x) \leq P(\hat{y}|x)$. y^* es una mejor traducción que \hat{y} . Pero el RNN predijo que $P(y^*|x) < P(\hat{y}|x)$. La conclusión es que el modelo de RNN es fallido.

Este proceso se puede repetir seguidamente con diferentes ejemplos del dev set y tratar de definir en qué proporción los errores son más probables por beam search o RNN model. En el caso de que el problema venga del beam search se puede tratar de aumentar el parámetro B, si proviene del RNN podríamos tratar de agregar regularización, agregar más training data o probar una arquitectura distinta, etc.

Bleu Score

Donde **Bleu** significa *Bilingual evaluation understudy*.

Esta sección está basada en el siguiente trabajo:

- [Papineni, Kishore, et al. "BLEU: a method for automatic evaluation of machine translation." Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, 2002.](#)

Veamos ahora el caso en el que una oración en francés tiene varias traducciones en inglés posibles. Dada una traducción hecha por el algoritmo lo que realiza es un score de cuán buena es la traducción.

French: Le chat est sur le tapis.

Reference 1: The cat is on the mat. ↩

Reference 2: There is a cat on the mat.

Esto comienza fijándose si cada una de las palabras generadas por el algoritmo figuran en la traducción humana a través de definir un score llamado **precisión**. Consideraremos el siguiente caso extremo en el que la traducción del algoritmo está dado únicamente por una palabra repetida

MT output: the the the the the the the.

Dada que hay 7 palabras en la oración y cada una de estas palabras aparece en ambas referencias (1 o 2) entonces la precisión daría Precisión 7/7. Por lo cual no es un buen método. La **precisión modificada** que es la que se utiliza *tiene en cuenta no cada palabra sino las veces en las que aparece en la oración de referencia como máximo*. Entonces esto da como resultado 2/7.

Continuando con otro ejemplo un poco mejor, analicemos el ejemplo del bleu score en bigramas, esto es pares de palabras.

Example: Reference 1: The cat is on the mat. ↪

Reference 2: There is a cat on the mat.

MT output: The cat the cat on the mat.

	Count	Count_clip	
the cat	2 ↪	1 ↪	Para analizar el score podemos seleccionar pares de palabras. Entonces los bigramas del MT se listan a continuación. (de manera similar se pueden tomar triadas de palabras llamadas trigramas), cuántas veces aparecen cada uno, y cuántas veces aparecen en cada referencia como máximo (count_clip).
cat the	1 ↪	0	
cat on	1 ↪	1 ↪	
on the	1 ↪	1 ↪	
the mat	1 ↪	1 ↪	

De esta manera nuestra la precisión modificada equivale al

$$\# \text{count_clip} / \# \text{count} = (1+0+1+1+1) / (2+1+1+1+1) = 4/6.$$

Ahora formalicemos esto un poco más, primero sobre unigramas. Si definimos a la traducción de la computadora como \hat{y} . De manera general para n-gramas la definición de la precisión viene dada por la figura de la derecha.

$$P_1 = \frac{\sum_{\text{Unigrams} \in \hat{y}} \text{Count}_{\text{clip}}(\text{unigram})}{\sum_{\text{Unigrams} \in \hat{y}} \text{Count}(\text{unigram})}$$

$$P_n = \frac{\sum_{n\text{-grams} \in \hat{y}} \text{Count}_{\text{clip}}(n\text{-gram})}{\sum_{n\text{-grams} \in \hat{y}} \text{Count}(n\text{-gram})}$$

A su vez se puede definir el **Bleu Score combinado** midiendo p_1, p_2, p_3 y p_4 , donde p_n es el bleu score para n-gramas como el promedio de las mismas exponenciada y multiplicada por un *factor de penalización* llamado **Brevity penalty (BP)**. Donde esta penalidad posee una forma funcional particular.

$$\text{BP} \exp\left(\frac{1}{4} \sum_{n=1}^4 P_n\right)$$

$$BP = \begin{cases} 1 & \text{if } MT_output_length > reference_output_length \\ \exp(1 - MT_output_length/reference_output_length) & \text{otherwise} \\ \exp(1 - reference_output_length/MT_output_length) \end{cases}$$

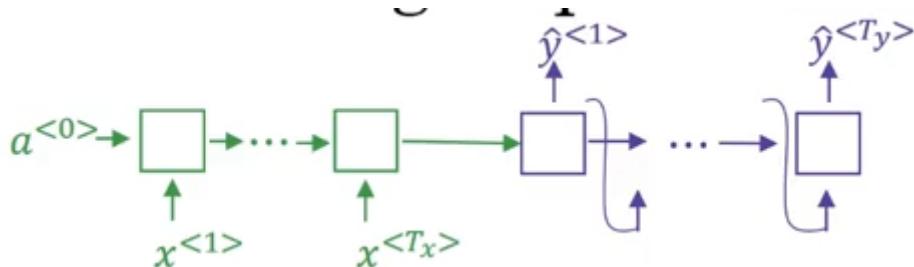
En resumen, **Bleu Score** es una métrica de un único número que permite decidir la calidad de la traducción de un texto.

Modelo de atención

Este modelo se desarrolló en el trabajo:

- [Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 \(2014\).](#)

Básicamente hasta ahora habíamos visto las redes neuronales consistiendo en una parte del tipo decoder y otra encoder, lo que vamos a ver ahora es un modelo mejorado de esto.



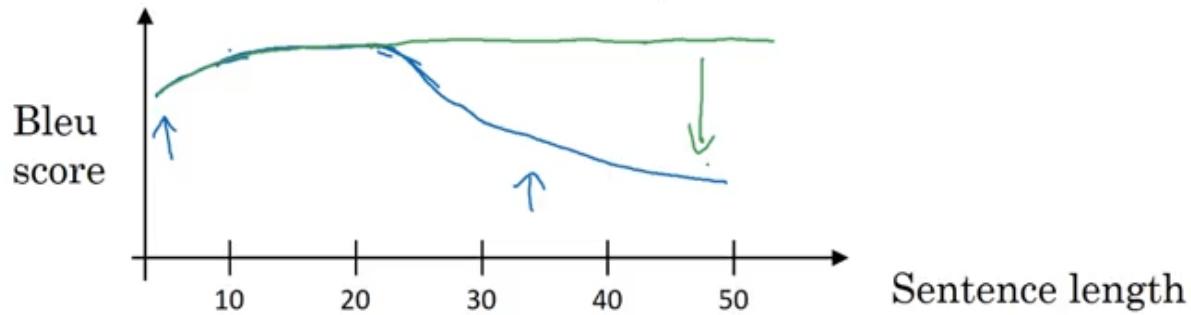
Veamos primero a que se debió este cambio, considerando largas secuencias de texto.

Jane s'est rendue en Afrique en septembre dernier, a apprécier la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.

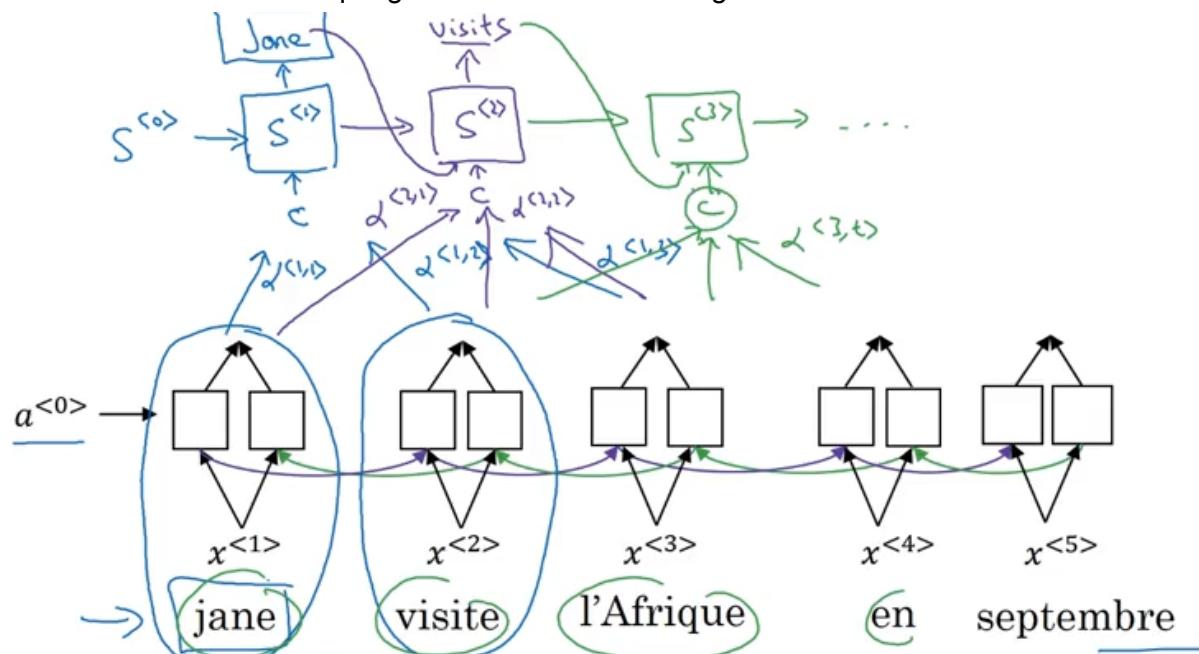
Principalmente el modelo de encoder-decoder toma la secuencia completa de texto y la traduce, mientras que un humano suele hacer esto por tramos de oraciones.

A su vez, se puede notar que el bleu score no funciona correctamente para secuencias mayores a 30 palabras (línea azul de la figura inferior), mientras que el modelo de atención al ir mirando oraciones más cortas posee un score mayor para oraciones más largas.



Ilustremos las ideas del modelo con una oración corta aunque fue pensado para oraciones más largas.

Consideremos un BRNN (RNN bidireccional), el primer paso para generar una salida es considerar que si se traduce la palabra Jane, ¿cuántas palabras se tuvieron en cuenta para generarla? No es necesario que se mire por completo toda la oración sino tal vez las más cercanas nomas. Es por eso que se definen unos parámetros de peso α que serán de entrada a la red neuronal que generará la oración en inglés.

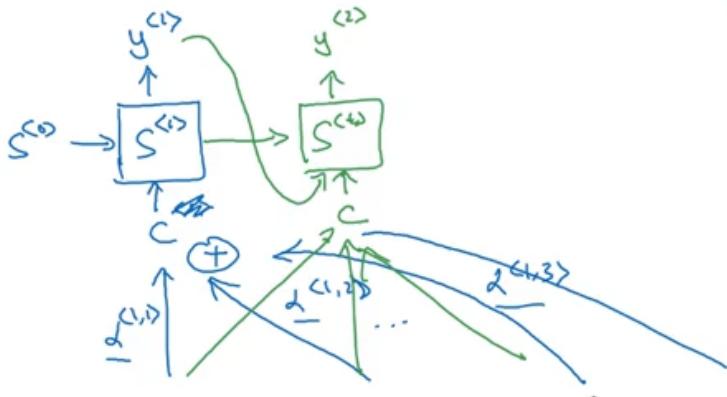


De manera similar para generar la segunda palabra se generarán una serie de parámetros de peso y así sucesivamente con el resto de las palabras.

Veamos cómo funciona este modelo formalmente. Primero definamos a las activaciones forward y backward como una única activación evaluada a tiempo t .

$$\underline{\alpha}^{\leftrightarrow} = (\overrightarrow{\alpha}^{\leftrightarrow}, \overleftarrow{\alpha}^{\leftrightarrow})$$

Por otra parte, para generar la traducción, usamos una RNN que NO es bidireccional.



La nueva palabra generada dependerá del estado inicial S_0 y de los vectores de contexto C que son una suma pesada de los pesos alfa con las activaciones.

$$\sum_{t'} \alpha^{<1, t'} = 1$$

$$C^{<1>} = \sum_{t'} \alpha^{<1, t'} a^{<t'>}$$

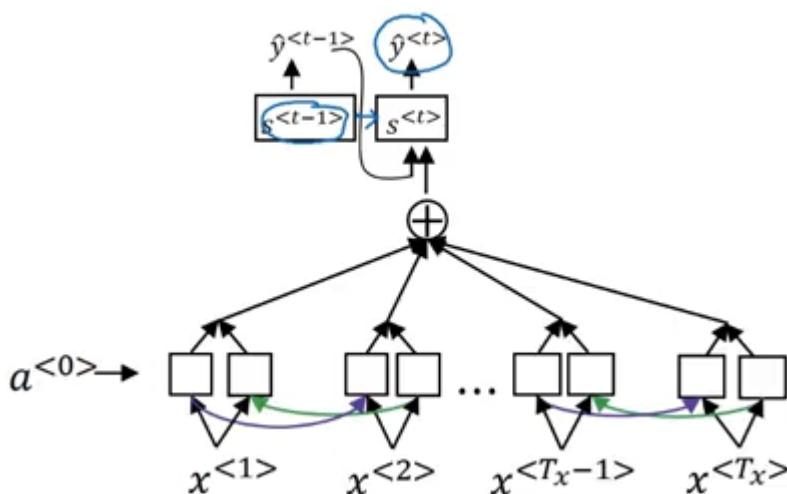
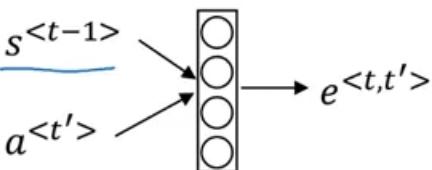
Lo que queda por ver es cómo calcular los pesos de atención α .

Esto fue obtenido del siguiente trabajo:

- [Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." International conference on machine learning. 2015.](#)

$\alpha^{<t, t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

$$\alpha^{<t, t'>} = \frac{\exp(e^{<t, t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t, t'>})}$$



Una manera de determinar cuánto tienen que valer los parámetros α es entrenando una red neuronal.

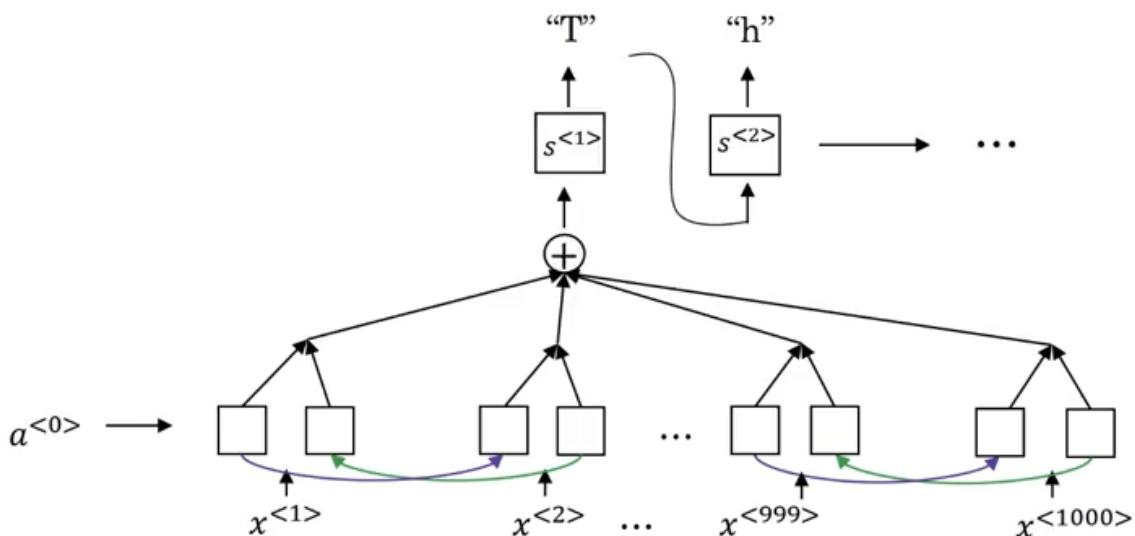
Una desventaja de este algoritmo es que el tiempo de cómputo va como $T_x * T_y$, donde T_x y T_y son el número de palabras de entrada y salida, respectivamente.

Reconocimiento del habla (Speech recognition)

Un ejemplo de esto es transcribir un audio a texto. El oído humano no procesa ondas sin procesar, pero posee estructuras que miden la cantidad de intensidad a diferentes frecuencias. Esto puede verse como un espectrograma, que es un gráfico de frecuencias en función del tiempo donde los colores indican la cantidad de energía.



¿Cómo construimos un modelo de atención para reconocimiento de audio?



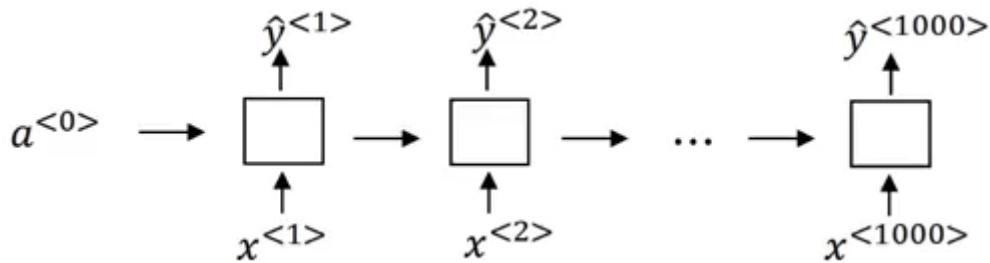
Costo CTC para reconocimiento de sonido

Donde **CTC** significa *Connectionist temporal classification*.

Esta idea fue obtenida del trabajo:

- [Graves, Alex, et al. "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks." Proceedings of the 23rd international conference on Machine learning. ACM, 2006.](#)

Supongamos que el audio clip decía “the quick brown fox”. Si el audio dura 10 segundos con un sampleo de 10 Hz entonces el audio posee mil caracteres. A su vez, se puede considerar el tamaño de la entrada igual que el de la salida.



Pero es probable que la entrada no necesite ni tenga ese mismo número de caracteres que la entrada. Un ejemplo podría ser el que sigue

ttt_h_eee _ _ _ qqq _ _

Que sería el comienzo de la oración “*the quick..*”. Una regla básica que se podría aplicar a continuación sería la de colapsar caracteres repetidos no separados por un espacio en blanco. Notemos que tenemos un espacio como un carácter adicional en este caso.

A día de hoy construir sistema de reconocimiento de audio a nivel de producción requiere una base de datos muy grande. Ahora veremos **cómo realizar un sistema de detección de palabras clave que requiere una cantidad razonable menor de datos**.

Ejemplos de sistemas que funcionan con una palabra para activar la detección de sonido son los siguiente.



Amazon Echo
(Alexa)



Baidu DuerOS
(xiaodunihao)



Apple Siri
(Hey Siri)



Google Home
(Okay Google)

Veamos un ejemplo de algoritmo que podemos utilizar para word trigger detection. En este caso se posee una detección de trigger que es cero en el caso en el que se diga una palabra clave como ‘*Okay Google*’ y unos luego. Como desventaja de esto es el número de ceros que posee.

