Data Science

www.datascience.pe

Data Science

Procesamiento de Lenguaje Natural

Elaborado por la Semilla 21

- Josué Gastón Távara

La Libertad - 2020

Data Science

NLP - Natural Language Processing

Natural Language

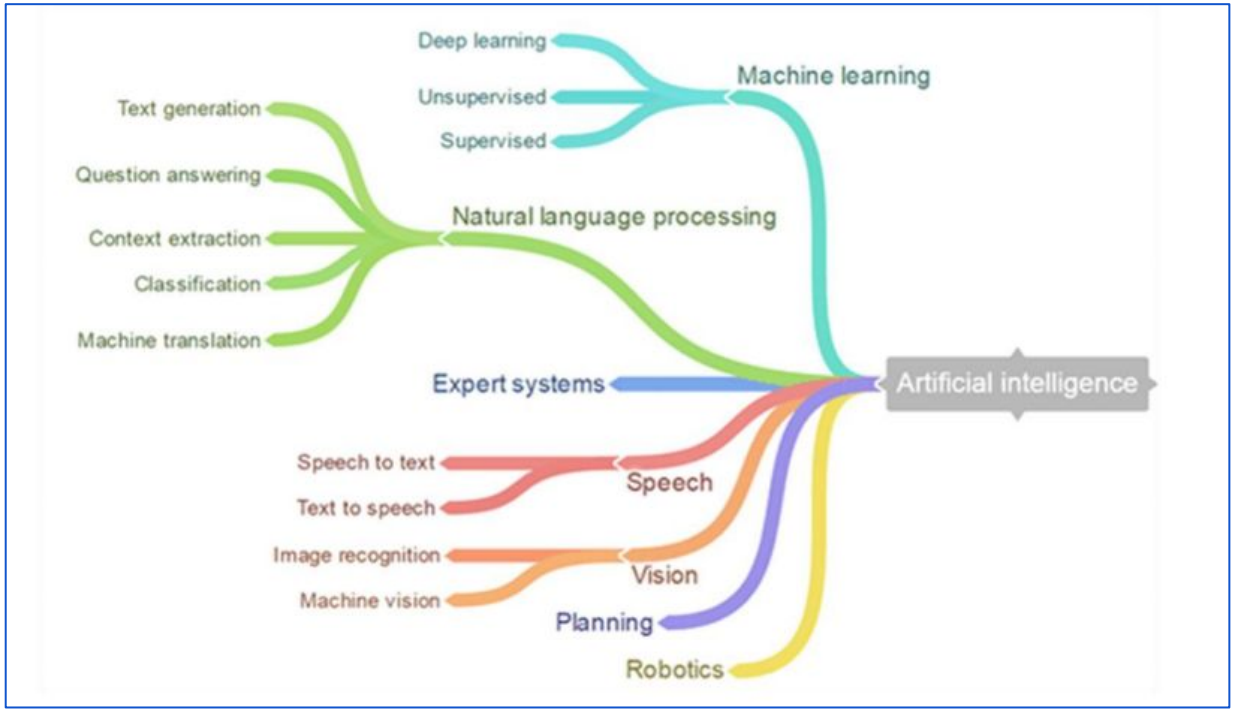
English Language
Spanish Language
~~Python Language~~

Processing

Cómo la computadora es capaz de procesar estos lenguajes naturales

ARTIFICIAL INTELLIGENCE
[Una computadora ejecuta tareas que un humano puede realizar]

NATURAL LANGUAGE PROCESSING
[Como manejar información en texto]



Relación con Procesamiento de Imágenes

A diferencia de las imágenes, donde se evalúan los valores de cada pixel, cuando se trabaja con texto, este tiene que ser codificado para que pueda ser procesado.

Valores ASCII para encodificar

Problema: Limita el significado semántico de las oraciones.

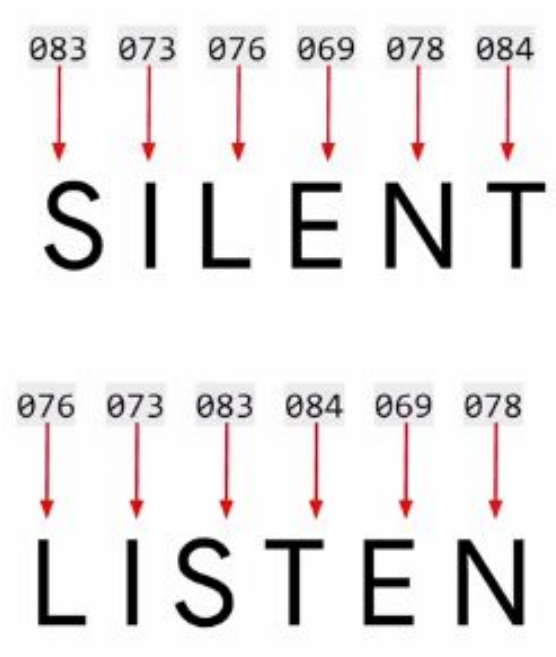


Figura 1. Teniendo los mismo valores ASCII , cada palabra tiene un significado totalmente diferente
Fuente: Moroney(2020)

Etiquetar cada palabra con un número

Podemos comenzar a obtener similaridades en sentencias.

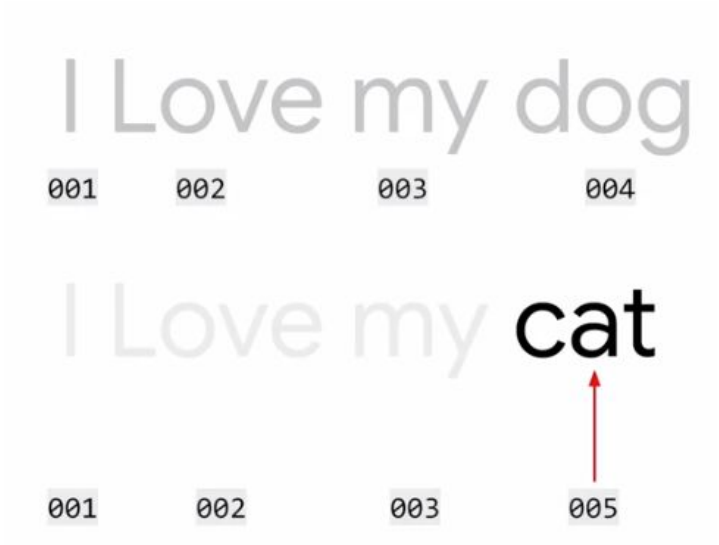


Figura 2. Etiquetando palabras con números, comenzamos a observar patrones
Fuente: Moroney(2020)

NLP - Conceptos: TOKEN

Palabras en el texto. Se puede considerar un token genérico (FDV) para palabras no mapeadas.

```python
import tensorflow as TF
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer

oraciones = [
    'Yo quiero a mi madre.',
    'Yo quiero a mi padre!',
    'Yo quiero helado'
]

tokenizador = Tokenizer(num_words = 20, oov_token="<FDV>")   # Fuera del vocabulario.
tokenizador.fit_on_texts(oraciones)       #
indices_palabras = tokenizador.word_index # Retorna un diccionario compuesto por clave-valor.

oraciones_nuevas = [
    'Madre yo quiero a mi padre',
    'Yo te quiero mucho'
]
#Convierte las oraciones a cada valor numérico basada en cada palabra.
texto_mapeado = tokenizador.texts_to_sequences(oraciones_nuevas)

print("Palabras mapeadas: ",indices_palabras, "\n")
print("Valor numerico: ",texto_mapeado)
```

[Tokenizer]

[fit_on_texts]

[texts_to_sequence]

```
(tensor35_gpu) D:\NLP>python example.py
Palabras mapeadas:  {'quiero': 3, 'mi': 5, 'yo': 2, 'helado': 8,
 'padre': 7, '<FDV>': 1, 'a': 4, 'madre': 6}

Valor numerico:  [[6, 2, 3, 4, 5, 7], [2, 1, 3, 1]]
```

NLP - Conceptos: Pad sequences

Al entrenar datos, se debe mantener una uniformidad de ellos. Por ejemplo, cuando se entrenan imágenes para problemas de visión computacional, todas ellas que se están entrenando son de dimensiones similares. Lo mismo sucede con las palabras.

```python
import tensorflow as TF
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
#Padding es usado para dar uniformidad en las sentencias
from tensorflow.keras.preprocessing.sequence import pad_sequences

oraciones = [
    'Yo quiero a mi madre.',
    'Yo quiero a mi padre!',
    'Yo quiero mucho a mi padre',
    'Yo quiero helado'
]
tokenizador = Tokenizer(num_words = 20, oov_token= "<FDV>")
tokenizador.fit_on_texts(oraciones)

indices_palabras = tokenizador.word_index
texto_mapeado = tokenizador.texts_to_sequences(oraciones)
texto_uniformidado = pad_sequences(texto_mapeado)

print(indices_palabras)
print(texto_mapeado)
print(texto_uniformidado)
```

```
(tensor35_gpu) D:\NLP>python example.py
{'mi': 5, 'madre': 7, 'mucho': 8, 'quiero': 3, 'yo': 2, '<FDV>': 1, 'padre': 6,
 'a': 4, 'helado': 9}

[[2, 3, 4, 5, 7], [2, 3, 4, 5, 6], [2, 3, 8, 4, 5, 6], [2, 3, 9]]

[[0 2 3 4 5 7]
 [0 2 3 4 5 6]
 [2 3 8 4 5 6]
 [0 0 0 2 3 9]]
```

NLP - Conceptos: Word Embedding

Las palabras asociadas se agrupan como vectores en un espacio multidimensional. El objetivo es encontrar palabras presentes en una oración con significados similares con la finalidad de colocarlas cerca una de la otra.

Ejemplo: "La película era **torpe** y **aburrida**"; "La película fue **divertida** y **emocionante**".

Palabras similares que muestran significados negativos o positivos, estarían mostrando similar sentimiento, por lo que podrían asociarse en vectores similares.

De manera general, palabras similares deben agruparse en categorías similares, por lo que podrían asociarse en vectores similares.

NLP - Conceptos: Word Embedding

Problema: Orange y apple no presentan cercanía en el vocabulario mapeado:

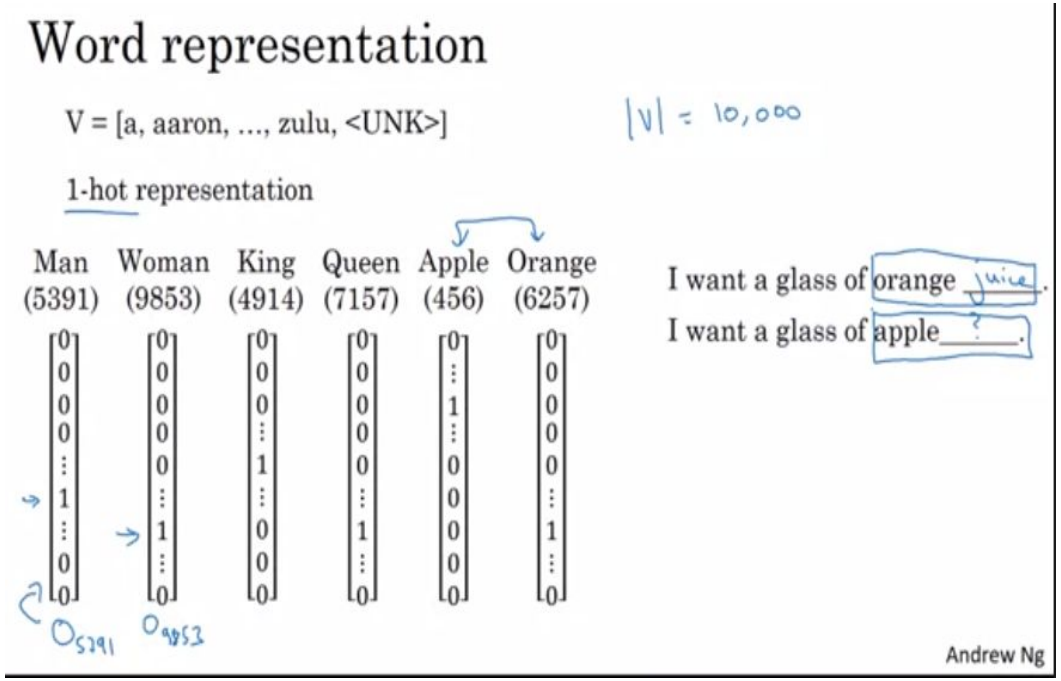


Figura 3.Representación de Palabras
Fuente: Ng (2020)

Algoritmos Word Embedding pueden aprender características similares para conceptos que parecen estar más relacionados:

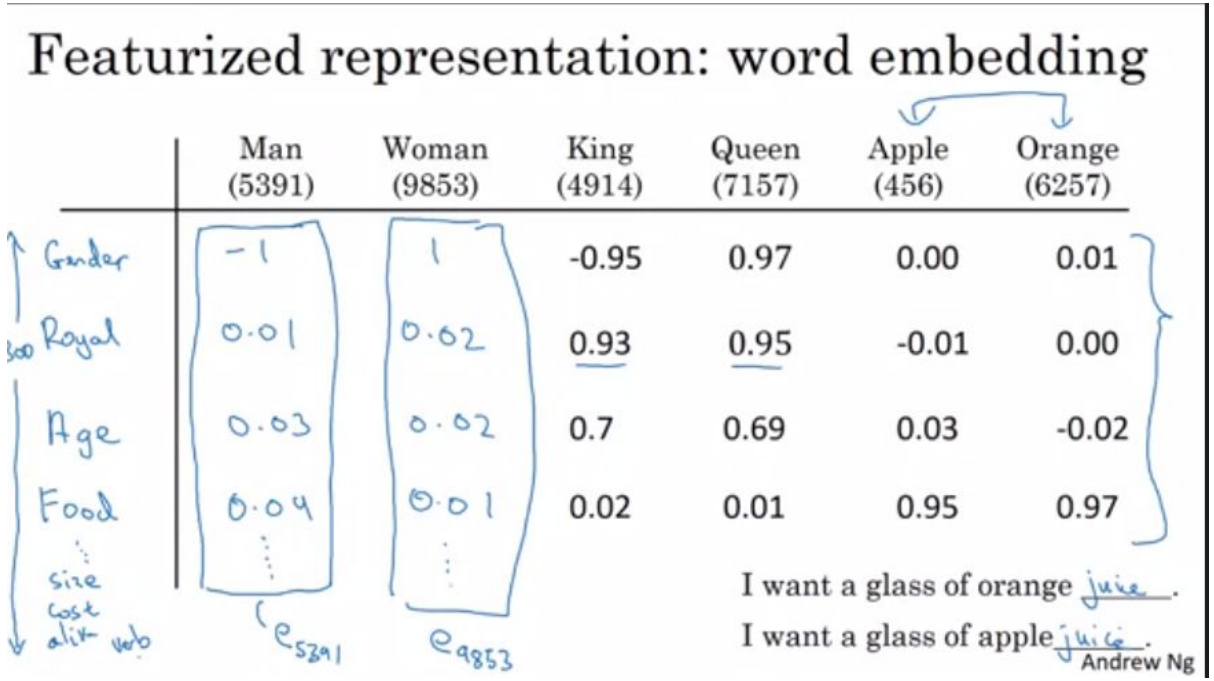


Figura 4.Representación Computarizada
Fuente: Ng (2020)

NLP - Conceptos: Word Embedding

Visualización:

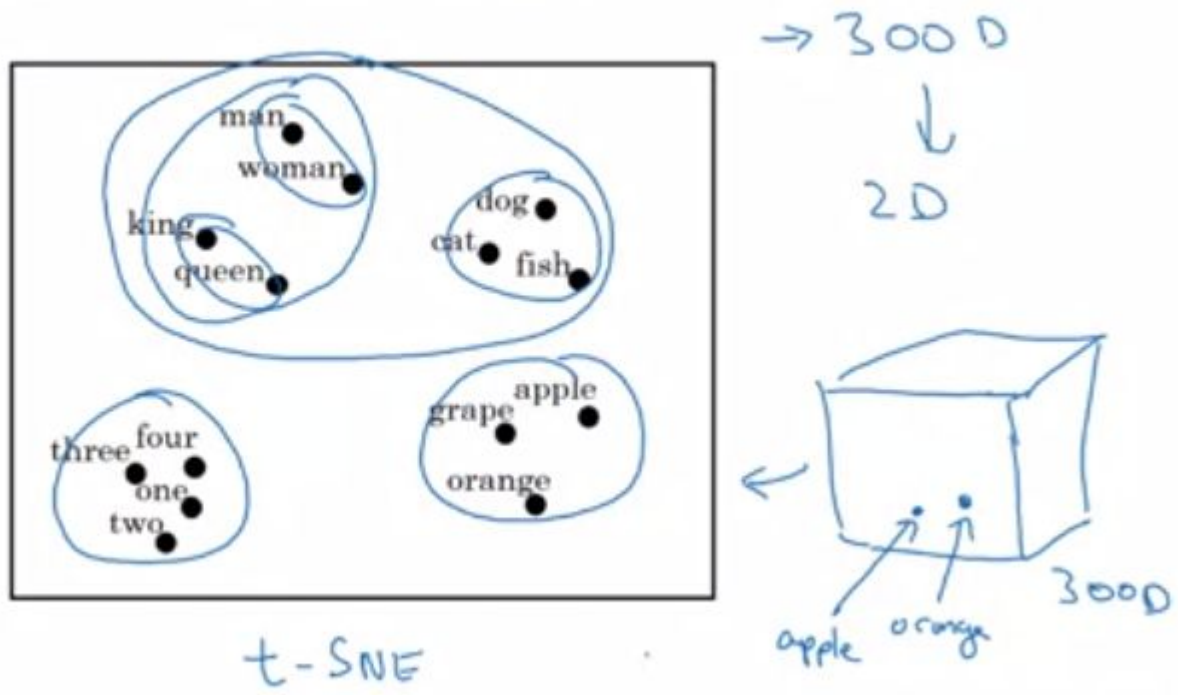


Figura 5. Visualización de Word Embedding
Fuente: Ng (2020)

Data Science

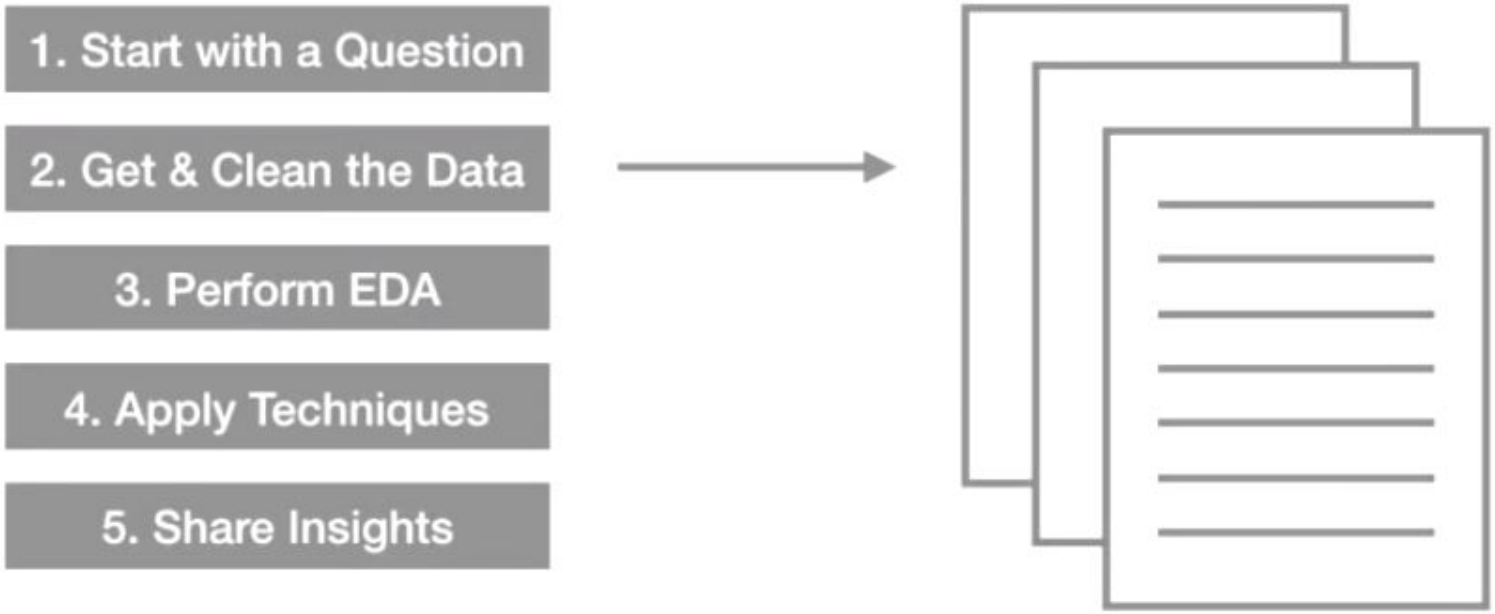
NLP - Conceptos: Stop words

Palabras inútiles o sin importancia, generalmente proposiciones de uso común que son configuradas para ser ignoradas en el análisis.

Data Science

Proceso de Desarrollo

Aplicamos los siguientes pasos:

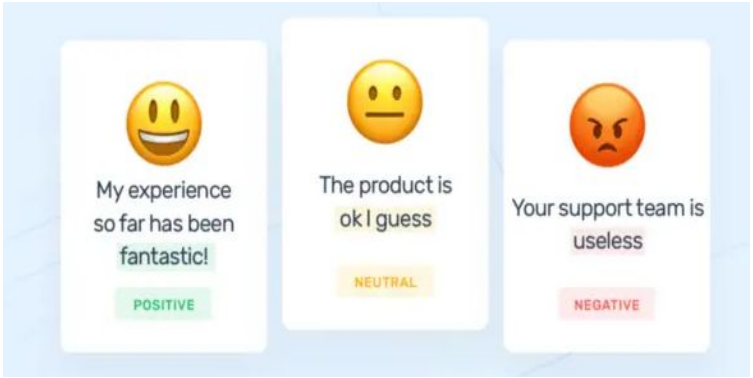


1) Start with a question

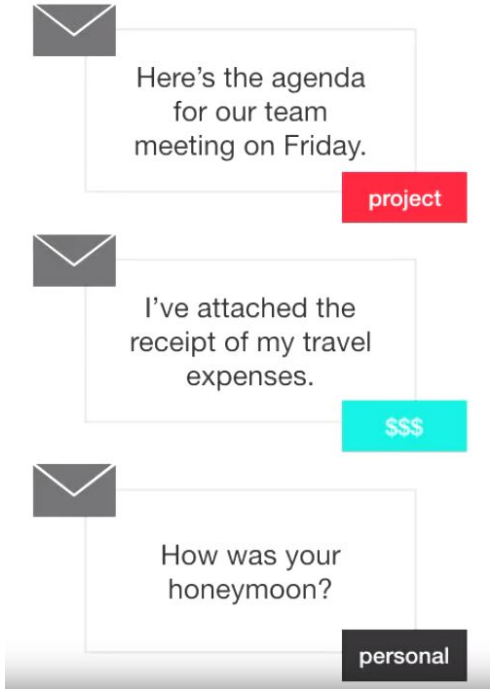
Data Science

NLP - Casos de Uso

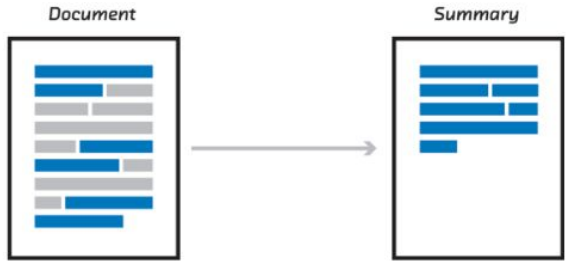
Sentiment Analysis

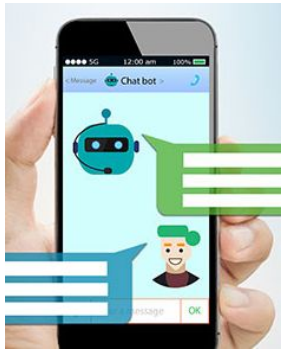


Topic Modeling



Text Generation





1) Topic

Data Science

Males/Females (Sorted by Popularity Ascending)

1-50 of 5,616,214 names. | Next »

Sort by: STARmeter▲ | A-Z | Birth Date | Death Date

1. **Ana de Armas**

Actress | Blade Runner 2049

Ana de Armas was born in Cuba on April 30, 1988. At the age of 14, she began her studies at the National Theatre School of Havana, where she graduated after 4 years. At the age of 16, she made her first film, Virgin Rose (2006), directed by Manuel Gutiérrez Aragón. A few titles came after until she...

2. **Sophia Lillis**

Actress | It

Sophia Lillis was born in Crown Heights, Brooklyn. She started her acting career at the age of seven, when her stepfather encouraged her to take acting classes at Lee Strasberg Theatre

```
:
# tenemos las 5 películas más populares para cada actor (donde aparecen como estrellas)
number_movies = 5

# 25 reviews para cada película compuesta con y sin spoilers, todas las clasificaciones y ordenadas por votos útiles
# Segun imdb "útil" no necesariamente implica que la revisión fue "positiva" o "favorable".
# Una revisión puede ser útil incluso si anañiza la película en criticas,
# porque puede ayudar a los lectores a decidir no perder su tiempo o dinero mirándola.
urls = ['https://www.imdb.com/title/tt8946378/reviews?ref_=tt_urv', # Knives Out
        'https://www.imdb.com/title/tt7979142/reviews?ref_=tt_urv', # The Night Clerk
        'https://www.imdb.com/title/tt1856101/reviews?ref_=tt_urv', # Blade Runner 2049
        'https://www.imdb.com/title/tt8079248/reviews?ref_=tt_urv', # Yesterday
        'https://www.imdb.com/title/tt1833116/reviews?ref_=tt_urv',

        'https://www.imdb.com/title/tt9446688/reviews?ref_=tt_urv', # I Am Not Okay with This
        'https://www.imdb.com/title/tt9086228/reviews?ref_=tt_urv', # Gretel & Hansel
        'https://www.imdb.com/title/tt1396484/reviews?ref_=tt_urv', # IT
        'https://www.imdb.com/title/tt7349950/reviews?ref_=tt_urv', # It Chapter Two
        'https://www.imdb.com/title/tt2649356/reviews?ref_=tt_urv', # Sharp Objects

        'https://www.imdb.com/title/tt2661044/reviews?ref_=tt_urv', #The 100
        'https://www.imdb.com/title/tt0460649/reviews?ref_=tt_urv', #How I Met Your Mother
        'https://www.imdb.com/title/tt0056758/reviews?ref_=tt_urv', #General Hospital
        'https://www.imdb.com/title/tt1587678/reviews?ref_=tt_urv', #Happy Endings
        'https://www.imdb.com/title/tt2477230/reviews?ref_=tt_urv', #The Night Shift

        'https://www.imdb.com/title/tt8806524/reviews?ref_=tt_urv', #Star Trek: Picard
        'https://www.imdb.com/title/tt0203259/reviews?ref_=tt_urv', #Law and Order
        'https://www.imdb.com/title/tt0364845/reviews?ref_=tt_urv', #NCIS
        'https://www.imdb.com/title/tt2193021/reviews?ref_=tt_urv', #Arrow
        'https://www.imdb.com/title/tt0112178/reviews?ref_=tt_urv', #Star Trek: Voyager

        'https://www.imdb.com/title/tt2119532/reviews?ref_=tt_urv', #Hacksaw Ridge
        'https://www.imdb.com/title/tt2177461/reviews?ref_=tt_urv', #A Discovery of Witches
        'https://www.imdb.com/title/tt1464540/reviews?ref_=tt_urv', #I Am Number Four
        'https://www.imdb.com/title/tt4786282/reviews?ref_=tt_urv', #Lights Out
        'https://www.imdb.com/title/tt2058673/reviews?ref_=tt_urv' #Point Break
        ]

actors = ['AnaA', 'SophiaL', 'LindseyM', 'JeriR','TeresaP']
```

Finales de febrero 2020

2.1) Get Data

Data Science

Web Scraping:

- Obtener data de un website -> import requests
- Extraer partes de un website -> import BeautifulSoup

Guardar data:

- Serializar en Python Objects -> import pickle

```
def getReviews(soup):
    reviews = [p.text for p in soup.find_all('div', class_ ='text show-more__control')]
    return reviews

# datos de transcripción de enlaces imdb
def url_to_review(url):
    '''Devuelve datos de transcritos específicamente de revisiones imdb'''
    page = requests.get(url).text
    soup = BeautifulSoup(page, 'lxml')
    reviews = getReviews(soup)
    #print(url)
    return reviews
```

2.2) Data Cleaning

Data Science

Objetivo: Obtener la data en un formato limpio y estándar para su posterior análisis.

Pasos comunes de limpieza de datos en pre-análisis del texto:

- ★ Hacer texto todo en minúsculas
- ★ Eliminar todos los signos de puntuación
- ★ Eliminar valores numéricos
- ★ Eliminar texto común no sensible (/n)

```
def clean_text(text):
    text = text.lower()
    text = re.sub('\[.*?\]', '', text) #
    text = re.sub('[‘’“”…]', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\w*\d\w*', '', text)
    text = re.sub('\n', '', text)

    return text
```

Algunos pasos un poco más complejos:

- ★ Tokenizar texto
- ★ Eliminar stop-words
- ★ Lematización
- ★ Crear bi-gramos o tri-gramos
- ★ Tratar con errores tipográficos

Diferente tipos de análisis requieren diferentes data en diferentes formatos:

- Formato #1: Corpus = Lista/colección de documentos
- Formato #2: Matriz de Documentos de Textos (Bag of words)

Formato #1 - Corpus (Lista/colección de documentos/textos)

Data Science

```
import pandas as pd
pd.set_option('max_colwidth',250)

data_df = pd.DataFrame.from_dict(data_Reviews).transpose()
data_df.columns = ['transcript']
full_names = ['Ana de Armas', 'Sophia Lillis','Lindsey Morgan','Jeri Ryan','Teresa Palmer']
data_df['full_name'] = full_names

data_df
```

	transcript	full_name
AnaA	what an excellent film by rian johnson definitely feels like the film he was destined to make writing that is slick as hell sublime performances most notably daniel craig who brings his agame in a wonderfully charismatic turn superb editing and w...	Ana de Armas
SophiaL	am i the only one who felt that waydo not get me wrong this series is really entertaining and witty but it just felt like the intro to the story itselfbinge watching is done easily and quickly with seven minutes episodes on average which just fe...	Sophia Lillis
LindseyM	first off i enjoyed season of the season was inferior compared to other seasonsthe is an american postapocalyptic drama television series i love the because it has a unique story line years after a nuclear apocalypse prisoners sent down to...	Lindsey Morgan
JeriR	sorry I have gotta go with the negative nerds on this one i love star trek I am an old man and i watched tos episodes as a kid i shushed my kids during new tng episodes because i could not wait for the video tape to watch it i watched voyager th...	Jeri Ryan
TeresaP	we knew already that mel gibson is a filmmaker with a powerful vision and the craftsmanship to go with it extraordinary battle scenes violence gibson style which means peckinpah plus because here there is such a personal intention that makes ever...	Teresa Palmer

Data Science

- El texto debe estar tokenizado, es decir, dividido en partes más pequeñas.
- La técnica de tokenización más común es dividir el texto en palabras.
- Podemos hacer esto usando CountVectorizer de scikit-learn, donde cada fila representará un documento diferente y cada columna representará una palabra diferente.
- Además, podemos eliminar stop words(palabras comunes que no agregan ningún significado adicional al texto).

```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(stop_words='english')
data_cv = cv.fit_transform(data_clean.transcript)
data_dtm = pd.DataFrame(data_cv.toarray(), columns=cv.get_feature_names())
data_dtm.index = data_clean.index
data_dtm
```

	ab	aback	abandoned	abandonedcreepy	abbey	abbeys	abby	abbys	abc	abcs	...	ziva	zivas	zombie	zombies	zombiesbowers	zone	zoom
AnaA	1	1	0	1	1	0	0	0	0	0	...	0	0	0	0	0	1	0
SophiaL	0	0	2	0	0	0	0	0	0	0	...	0	0	7	1	1	0	0
LindseyM	0	0	1	0	0	0	1	0	8	1	...	0	0	0	0	0	0	0
JeriR	0	0	0	0	1	1	13	1	0	0	...	12	1	0	0	0	0	0
TeresaP	0	0	1	0	0	0	0	0	0	0	...	0	0	1	0	0	1	1

5 rows × 11399 columns

3. Perform EDA - Exploratory Data Analysis

Después del paso de limpieza de datos donde colocamos nuestros datos en algunos formatos estándar, el siguiente paso es echar un vistazo a los datos y ver si lo que estamos viendo tiene sentido.

Antes de aplicar algoritmos sofisticados, siempre es importante explorar primero los datos.

Al trabajar con datos numéricos, algunas de las técnicas de análisis de datos exploratorios (EDA) que podemos usar incluyen encontrar el promedio del conjunto de datos, la distribución de los datos, los valores más comunes, etc. La idea es la misma cuando se trabaja con texto datos. Vamos a encontrar algunos patrones más obvios con EDA antes de identificar los patrones ocultos con técnicas de Machine Learning.

3. Perform EDA - Exploratory Data Analysis

	AnaA	SophiaL	LindseyM	JeriR	TeresaP
abilities	0	0	0	0	5
abilities changing	0	0	0	0	1
abilities connection	0	0	0	0	1
abilities known	0	0	0	0	1
abilities range	0	0	0	0	1
abilities strengths	0	0	0	0	1
ability	1	3	1	4	3

```
# Encuentra las N palabras principales que se dicen en cada crítica de la película, por actor
N =50
top_dict = {}
for c in data.columns:
    top = data[c].sort_values(ascending=False).head(N)
    top_dict[c]= list(zip(top.index, top.values))

print(top_dict['AnaA'][:10])
print(top_dict['SophiaL'][:10])
```

```
[('film', 190), ('movie', 132), ('good', 81), ('did', 79), ('story', 72), ('just', 67), ('like', 53), ('really', 49), ('great', 45), ('music', 43)]
[('movie', 190), ('just', 127), ('film', 121), ('like', 107), ('story', 78), ('did', 63), ('really', 61), ('series', 58), ('horror', 58), ('good', 54)]
```

3. Perform EDA - Exploratory Data Analysis

```
# Imprimir las 25 palabras principales que se dicen para cada crítica de cine para cada actor
for actor, top_words in top_dict.items():
    print(actor)
    print(', '.join([word for word, count in top_words[0:25]]))
    print('---')
```

```
AnaA
film, movie, good, did, story, just, like, really, great, music, beatles, time, acting, watch, does, plot, films, character, my
stery, think, cast, make, murder, bad, people
---
SophiaL
movie, just, film, like, story, did, really, series, horror, good, great, kids, time, pennywise, scene, cgi, book, scary, chara
cters, camille, way, does, acting, feel, character
---
LindseyM
like, just, characters, love, watch, really, great, season, watching, people, hospital, good, time, episode, think, funny, stor
y, character, series, general, friends, general hospital, writers, say, shows
---
JeriR
series, season, star, trek, star trek, like, just, voyager, characters, watch, episode, good, episodes, time, great, really, ch
aracter, shows, new, watching, make, better, story, cast, love
---
TeresaP
movie, film, just, like, story, good, does, characters, action, really, point, war, time, did, horror, original, make, scenes,
doss, break, number, character, great, point break, love
---
```

3. Perform EDA - Exploratory Data Analysis

Data Science

```
# Teniendo las 25 palabras principales que se dicen en cada review de la película, por actor
words = []
for palabra in data.columns:
    top = [word for (word, count) in top_dict[palabra]]
    for t in top:
        words.append(t)
```

```
# Analizamos las palabras principales más comunes presentes en los reviewws para los 5 actores
from collections import Counter
# Agregamos esta lista e identificamos las palabras más comunes junto con cuántas veces aparecen para cada reivew de película
Counter(words).most_common()
```

```
[('good', 5),
 ('did', 5),
 ('story', 5),
 ('just', 5),
 ('like', 5),
 ('really', 5),
 ('great', 5),
 ('time', 5),
 ('watch', 5),
 ('does', 5),
 ('character', 5),
 ('make', 5),
 ('people', 5),
 ('characters', 5),
 ('love', 5),
 ('better', 5),
 ('way', 5),
 ('think', 4),
 ('series', 4),
```

3. Perform EDA - Exploratory Data Analysis

Data Science

- Actualizamos nuestra data

```
add_stop_words = [word for word, count in Counter(words).most_common() if count >= 3]
```

```
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer

data_clean = pd.read_pickle('pickles/data_clean.pkl')

stop_words = text.ENGLISH_STOP_WORDS.union(add_stop_words)

# Recreamos la matrix de Documentos de Textos
cv = CountVectorizer(stop_words=stop_words, ngram_range=(1, 2))
data_cv = cv.fit_transform(data_clean.transcript)
data_stop = pd.DataFrame(data_cv.toarray(), columns=cv.get_feature_names())
data_stop.index = data_clean.index
```

	ab	ab school	aback	aback things	abandoned	abandoned fatherluke	abandoned incoherent	abandoned later	abandoned says	abandonedcreepy	...	zone melodrama	zoom	zoom close
AnaA	1	1	1	1	0	0	0	0	0	1	...	0	0	0
SophiaL	0	0	0	0	2	0	1	1	0	0	...	0	0	0
LindseyM	0	0	0	0	1	1	0	0	0	0	...	0	0	0
JeriR	0	0	0	0	0	0	0	0	0	0	...	0	0	0
TeresaP	0	0	0	0	1	0	0	0	1	0	...	1	1	1

```
['good',
 'did',
 'story',
 'just',
 'like',
 'really',
 'great',
 'time',
 'watch',
 'does',
 'character',
 'make',
 'people',
 'characters',
 'love',
 'better',
 'way',
 'think',
 'series',
 'film',
 'movie',
 'plot',
 'cast',
 'bad',
 'little',
 'lot',
 'best',
 'episode',
 'going',
 'season',
 'episodes',
 'watching']
```

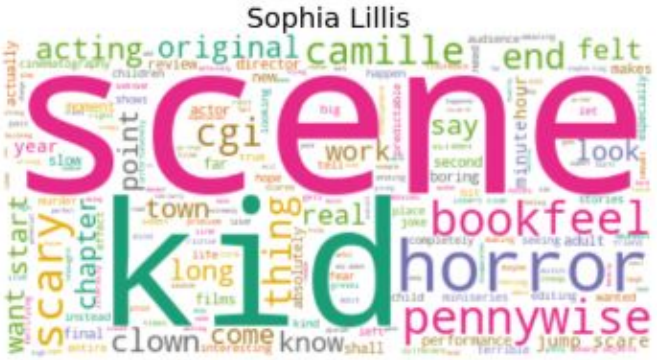
3. Perform EDA - Exploratory Data Analysis

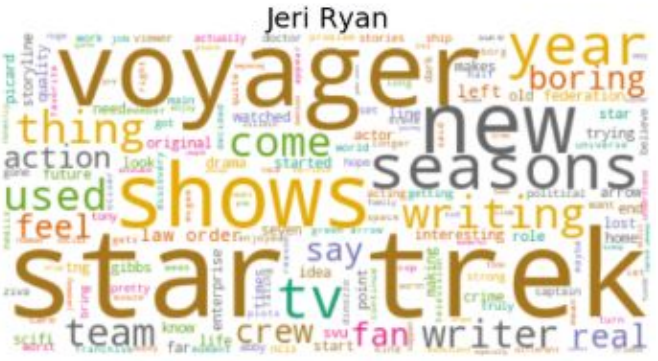
Data Science

- Visualizamos nuestra data mediante wordClouds

```
# Terminal / Anaconda Prompt: conda install -c conda-forge wordcloud
from wordcloud import WordCloud

wc = WordCloud(stopwords=stop_words, background_color="white", colormap="Dark2",
               max_font_size=150, random_state=42)
```

```
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [25, 25]

actors = ['Ana de Armas',
          'Sophia Lillis',
          'Lindsey Morgan',
          'Jeri Ryan',
          'Teresa Palmer'
         ]

# Creamos subplots para cada actor
for index, movie in enumerate(data.columns):
    wc.generate(data_clean.transcript[movie])

    plt.subplot(4, 2, index+1)
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(actors[index])

plt.show()
```



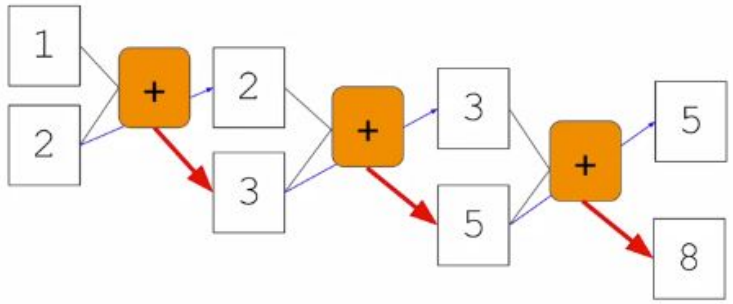


4. Apply Techniques - Text Generation
Redes neuronales para NLP

Para que una red neuronal tome en cuenta la secuencia de las palabras, actualmente se utilizan Arquitecturas de Redes Neuronales especializadas, como por ejemplo: RNN o LSTM.

Siendo esta última una de las más importantes debido a que el contexto de las palabras es preservado desde un inicio hasta su fin (sin importar que tan larga sea la oración).

Serie Fibonacci



Esta secuencia funciona de manera recurrente y puede ser derivada a través de una red neuronal.

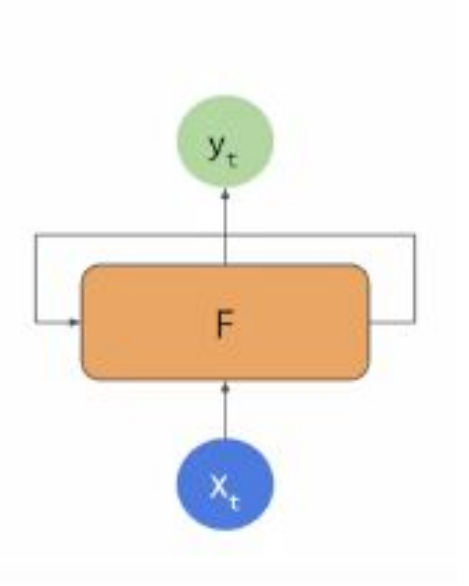
1	2	3	5	8	13	21	34	55	89

n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9

$$n_x = n_{x-1} + n_{x-2}$$

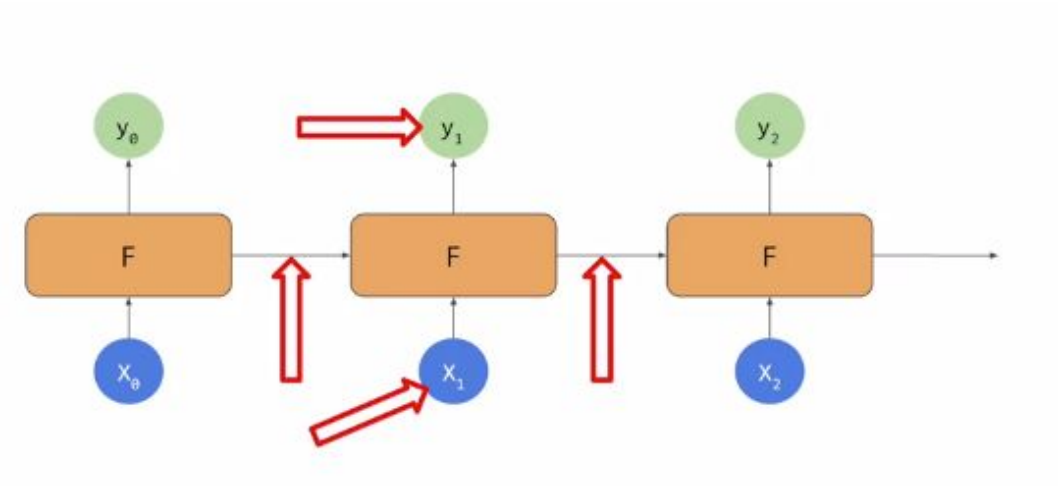
Serie Fibonacci - función recurrente

La función recurrente de la serie de Fibonacci se puede representar mediante el diagrama a la derecha, donde Xt son los números iniciales de la serie.



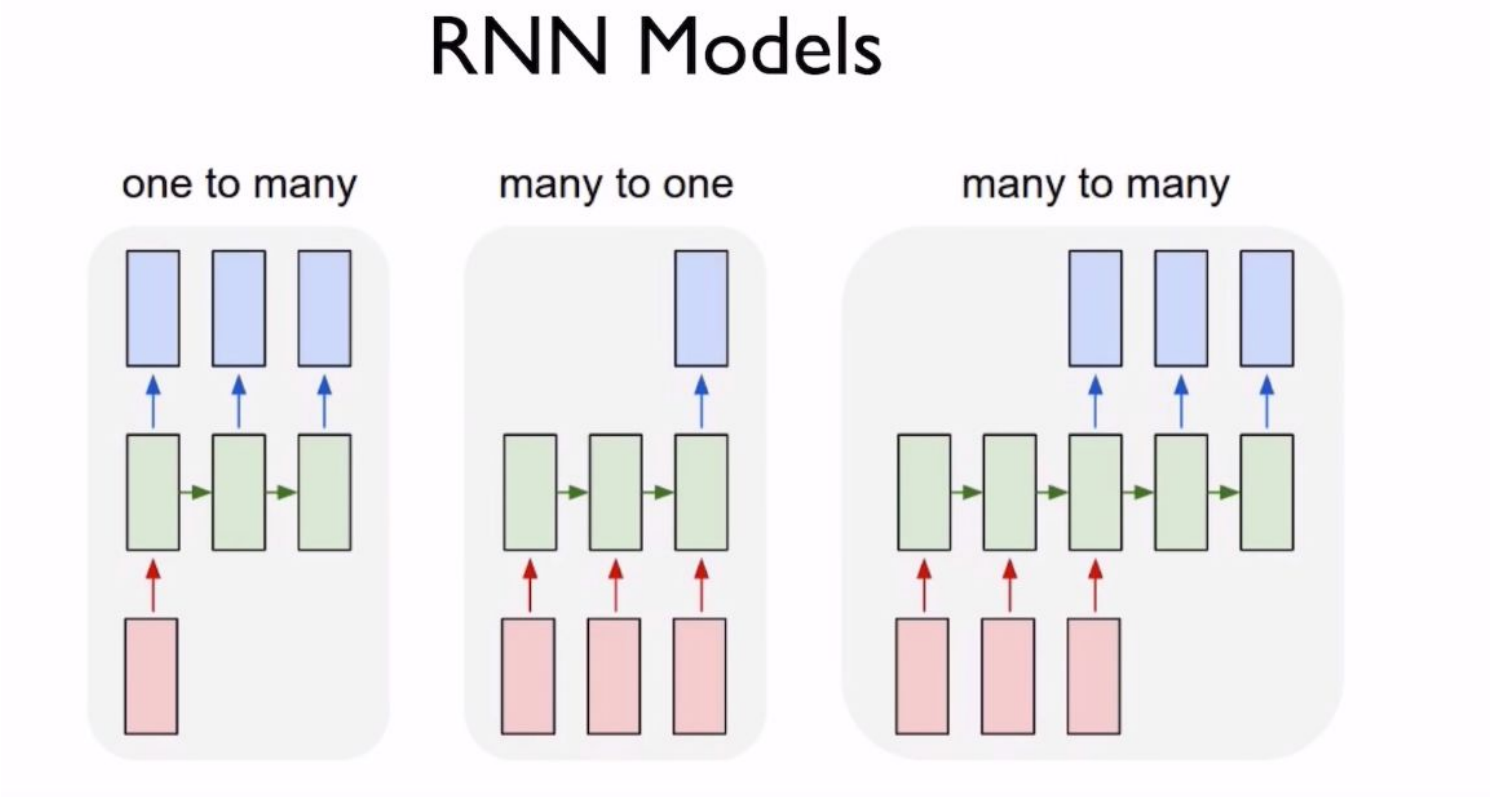
Luego, la función genera un yt, es decir, la suma de los dos primeros números y el valor de la suma se lleva a la siguiente iteración donde se agrega al segundo número y genera otro valor. De esa forma, la secuencia continúa.

Serie Fibonacci - función recurrente



Observamos que la salida depende en gran medida del paso anterior inmediato y es menos dependiente de los pasos iniciales si la serie es particularmente grande, es decir, y2 depende en gran medida del paso anterior (X1, F, y1) y es menos dependiente de (X0, F, y0).

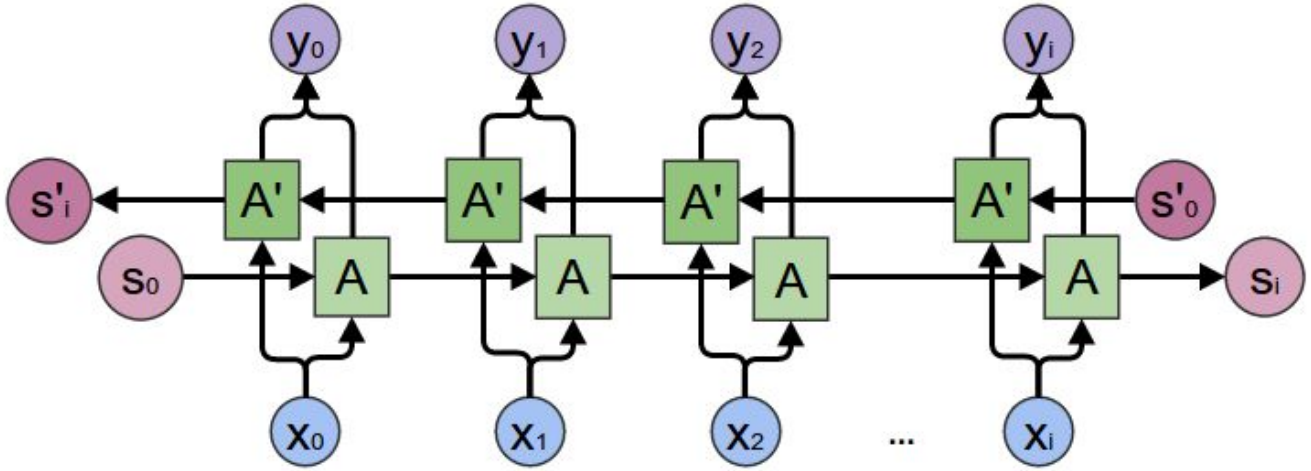
Recurrent Neural Network (RNN)



[A friendly introduction to Recurrent Neural Networks]

Bidirectional Recurrent Neural Network(BRNN)

Los BRNN son especialmente útiles cuando se necesita el contexto de la entrada.

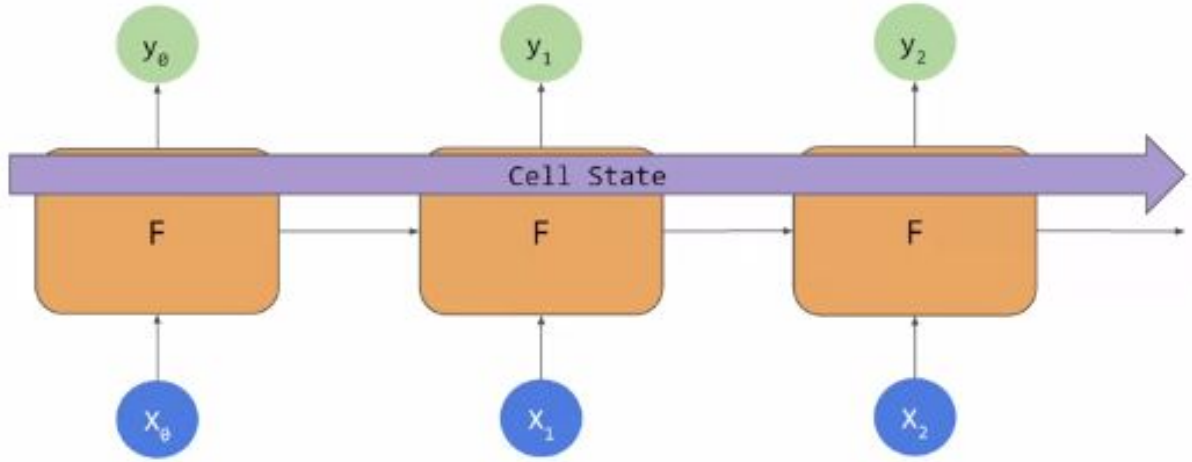


Al usar dos direcciones de tiempo, se puede usar la información de entrada del pasado e información futura del tiempo actual, a diferencia del RNN estándar que requiere de información solo pasada para incluirse en la información futura.

LSTM - Long Short Term Memory (Type of Recurrent Neural Network)

Link: LSTM a detalle

En este tipo de redes, además del contexto que se pasa como en un RNN, el LSTM tiene un pipeline de contexto adicional llamada **Cell State** que pasa a través de la red. Esto ayuda a mantener el contexto de los tokens anteriores o los pasos relevantes en los posteriores.



El Cell State puede ser bidireccional para que los tokens que aparecen más adelante en una oración puedan afectar a los tokens anteriores.

LSTM - Long Short Term Memory - Entrenamiento

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

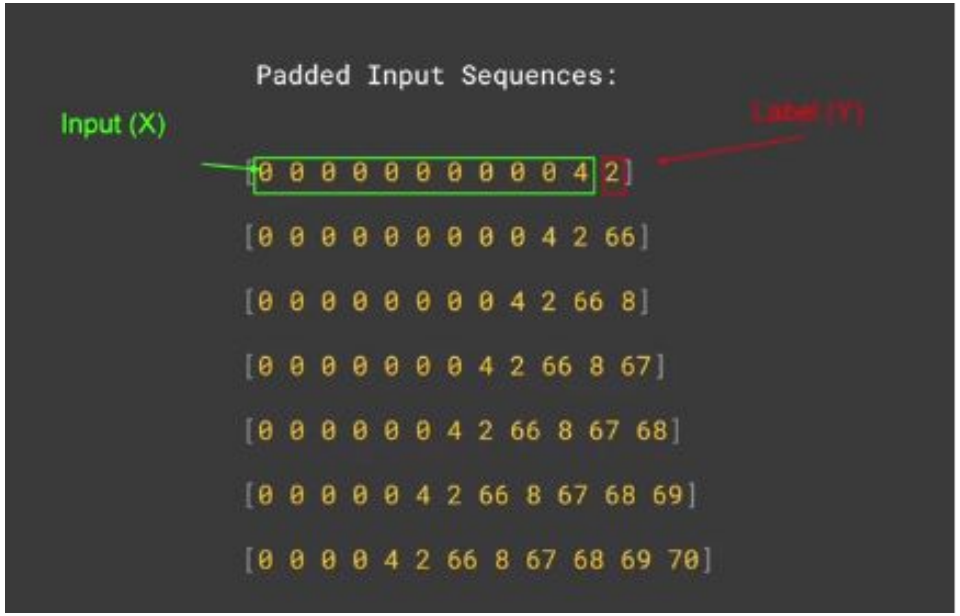
```
Line:                          Input Sequences:

[4 2 66 8 67 68 69 70]         [4 2]
                               [4 2 66]
                               [4 2 66 8]
                               [4 2 66 8 67]
                               [4 2 66 8 67 68]
                               [4 2 66 8 67 68 69]
                               [4 2 66 8 67 68 69 70]
```

```
# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

```
Line:                          Padded Input Sequences:

[4 2 66 8 67 68 69 70]         [0 0 0 0 0 0 0 0 0 0 4 2]
                               [0 0 0 0 0 0 0 0 0 4 2 66]
                               [0 0 0 0 0 0 0 0 4 2 66 8]
                               [0 0 0 0 0 0 0 4 2 66 8 67]
                               [0 0 0 0 0 0 4 2 66 8 67 68]
                               [0 0 0 0 0 4 2 66 8 67 68 69]
                               [0 0 0 0 4 2 66 8 67 68 69 70]
```

LSTM - Long Short Term Memory - Entrenamiento

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# creamos en conjunto de datos de entrenamiento y sus salidas esperadas
xs, labels = input_sequences[:,:-1],input_sequences[:,-1]

ys = tf.keras.utils.to_categorical(labels, num_classes=total_words)
```



LSTM - Long Short Term Memory - Entrenamiento

```
model = Sequential()
model.add(Embedding(total_words, 400, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(200)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

#earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=5, verbose=0, mod
history = model.fit(xs, ys, epochs=5, verbose=1, batch_size=20)

print(model.summary())
```

```
Train on 34710 samples
Epoch 1/5
34710/34710 [==============================] - 50s 1ms/sample - loss: 7.1555 - accuracy: 0.0820
Epoch 2/5
34710/34710 [==============================] - 47s 1ms/sample - loss: 6.9321 - accuracy: 0.1058
Epoch 3/5
34710/34710 [==============================] - 48s 1ms/sample - loss: 6.6561 - accuracy: 0.1101
Epoch 4/5
34710/34710 [==============================] - 47s 1ms/sample - loss: 6.0101 - accuracy: 0.1231
Epoch 5/5
34710/34710 [==============================] - 47s 1ms/sample - loss: 5.8317 - accuracy: 0.1281
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 68, 400)           2035200
_________________________________________________________________
bidirectional (Bidirectional (None, 400)               961600
_________________________________________________________________
dense (Dense)                (None, 5088)              2040288
=================================================================
Total params: 5,037,088
Trainable params: 5,037,088
Non-trainable params: 0
```

Entrenamos la red por 5 épocas.
Época: es cuando todo un conjunto de datos se pasa hacia adelante y hacia atrás a través de la red neuronal solo una vez.

LSTM - Long Short Term Memory - Evaluación

```
seed_text = "I really hate"
next_words = 27

for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predicted = model.predict_classes(token_list, verbose=0)
    output_word = ""
    for word, index in tokenizer.word_index.items():
        if index == predicted:
            output_word = word
            break
    seed_text += " " + output_word
print(seed_text)
```

```
I really hate a unintentionally add and i have been predictable and respectful and gang extra on the first episodes of the firs
t episodes of the first minutes by wan's
```

Bibliografía

- Ng,A. (2020) . [Material de Clase]. Sequence Models. Deeplearning.ai. Extraído de: https://www.coursera.org/lecture/nlp-sequence-models/

- Misra, R., & Arora, P. (2020). Sarcasm Detection using Hybrid Neural Network. Retrieved 9 February 2020, from https://arxiv.org/abs/1908.07414

- Moroney,L. (2020) [Material de Clase]. Natural Language Processing in Tensorflow. Deeplearning.ai. Extraído de: https://www.coursera.org/learn/natural-language-processing-tensorflow/lecture

- Moujahid, A. (2016). A practical introduction to deep learning with caffe and python. Extraído de: http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/. [Online; accesado Febrero 5, 2020].