Escuela Latinoamericana de Aprendizaje Profundo

Curso de Lenguaje

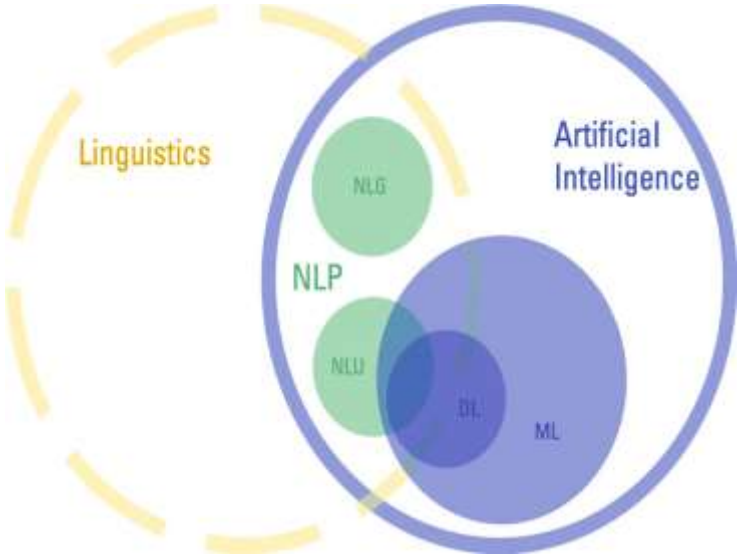
Día 1

Prof. Alexandra González Eras

Agenda

- Parte 1: NLP
  - Introducción a PLN
  - Análisis en PLN
  - Modelo de lenguaje
  - Redes neuronales RNN simple

Introducción a NLP



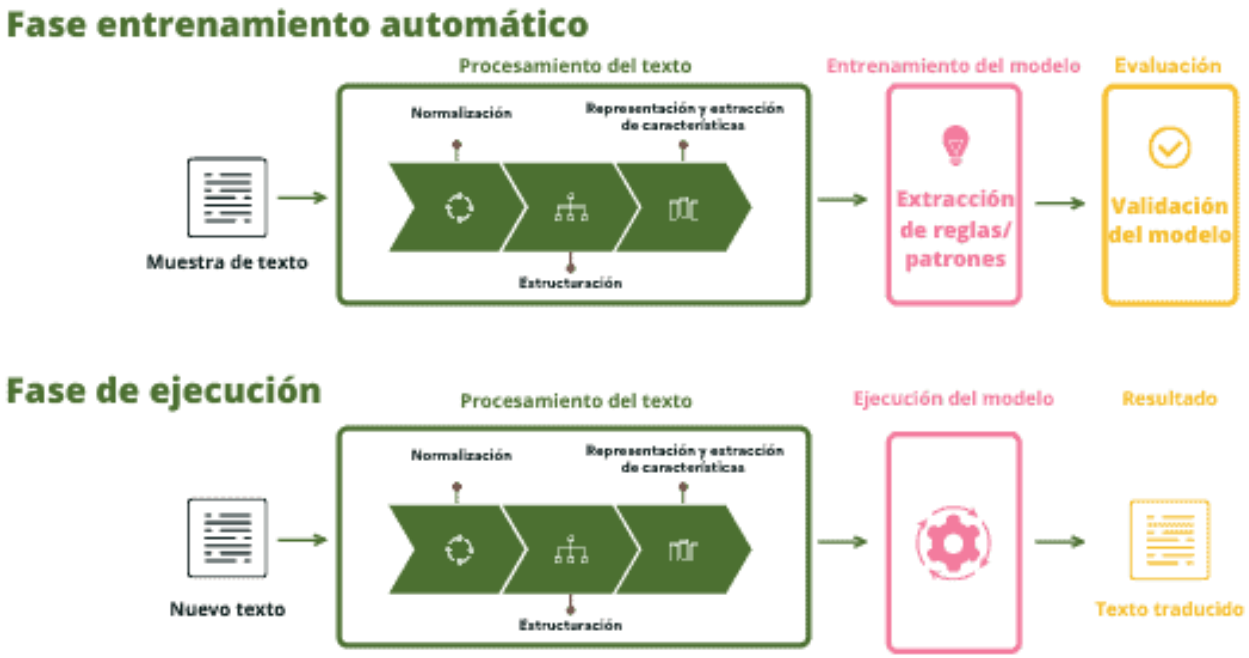
NLP: Natural Language Processing

NLG: Natural Language Generation

NLU: Natural Language Understanding

Análisis en PLN
Procesamiento de texto

Análisis en PLN
Enfoque Estadístico



Modelo de lenguaje
Bolsas de palabras

1. This pasta is very tasty and affordable.
2. This pasta is not tasty and is affordable.
3. This pasta is very very delicious.

Oraciones iniciales

- this pasta is very tasty and affordable.
- this pasta is not tasty and is affordable.
- this pasta is very very delicious.

Normalización (mayúsculas)

["and", "affordable.", "delicious.", "is", "not", "pasta", "tasty", "this", "very"]

Tokenización (Corpus)

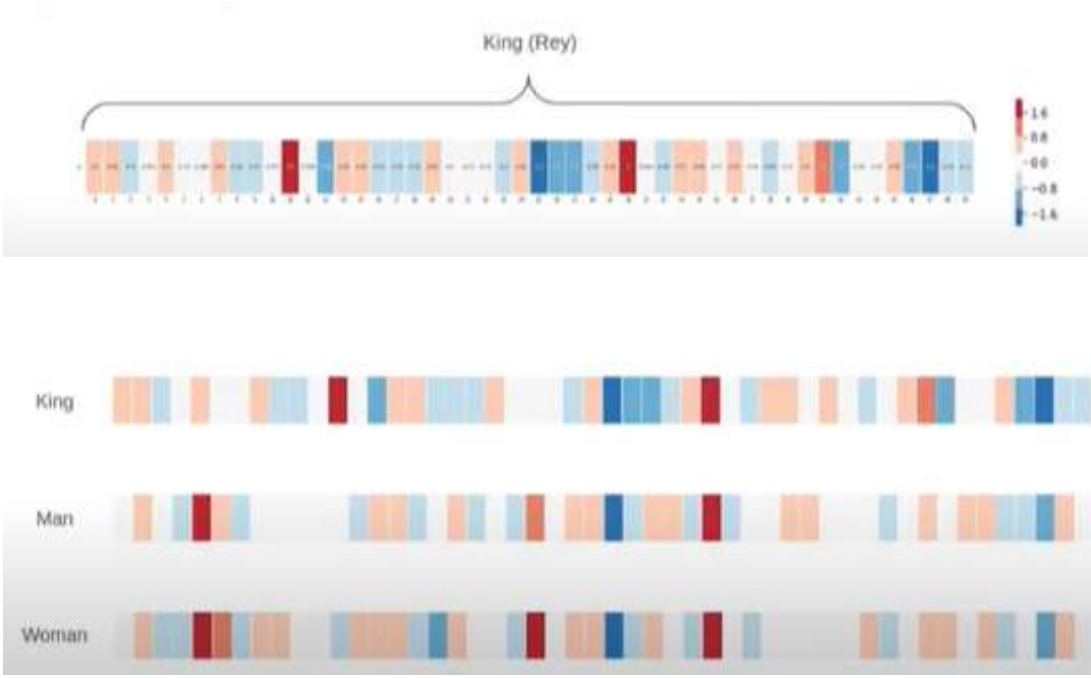
Corpus: contiene unidades de significado textual (palabras, frases, párrafos hasta conjuntos de documentos)

Vectores de frecuencias

s.no	and	afford able	delicio us	is	not	pasta	tasty	this	very
1	1	1	0	1	0	1	1	1	1
2	1	1	0	2	1	1	1	1	0
3	0	0	1	1	0	1	0	1	2

Sparce matrix

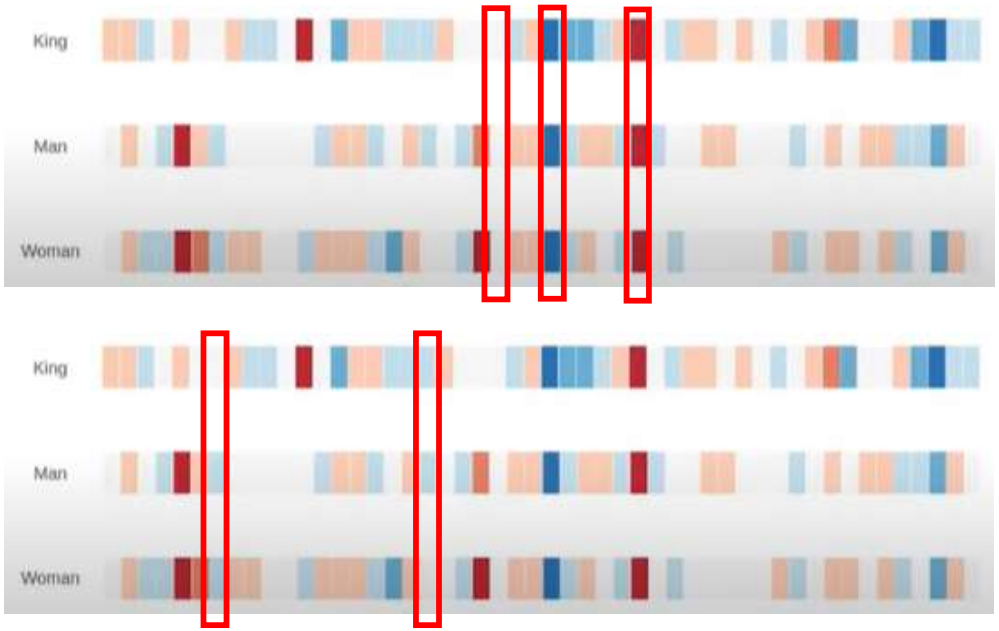
Modelo de lenguaje Word2Vec



Técnica basada en Aprendizaje Automático que aprende asociaciones de palabras a partir de corpus de texto.

Cada una de las palabras es codificada como un vector de números en un espacio dimensional, que puede ser emparejado con otros vectores que aparecen en contextos similares

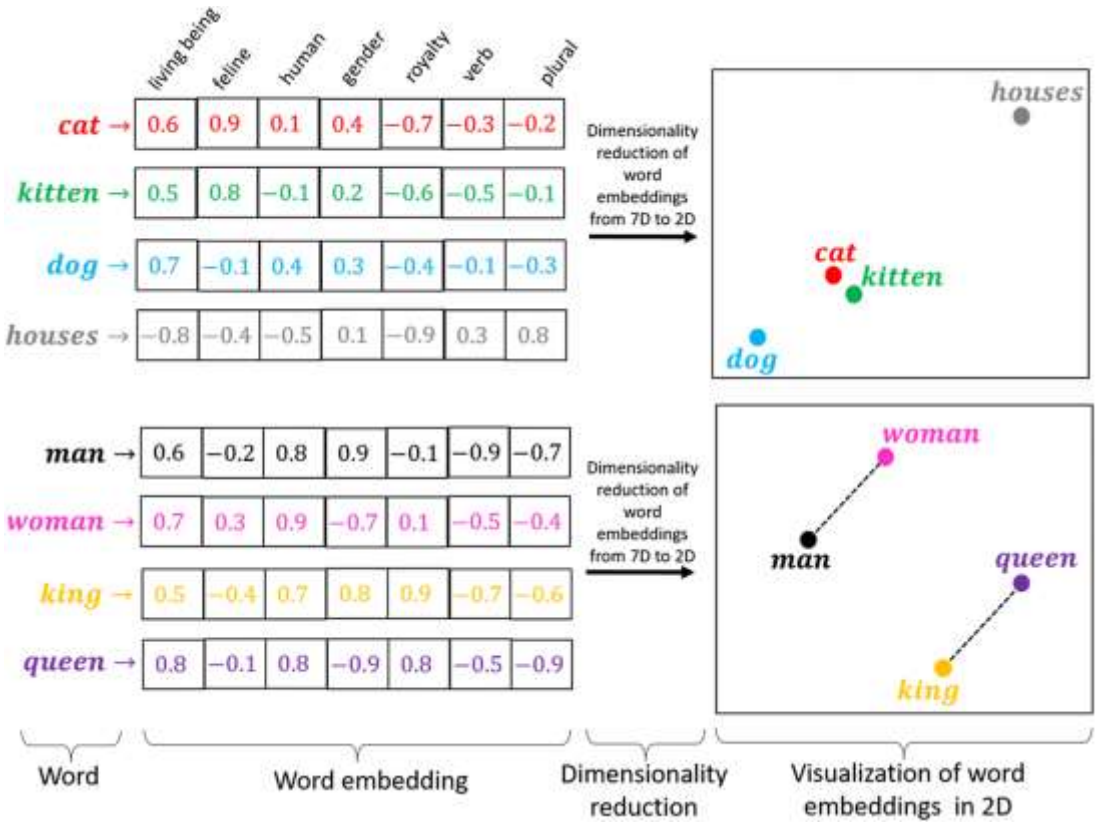
Modelo de lenguaje Word2vec



Existen atributos similares que permiten definir similitudes entre las palabras

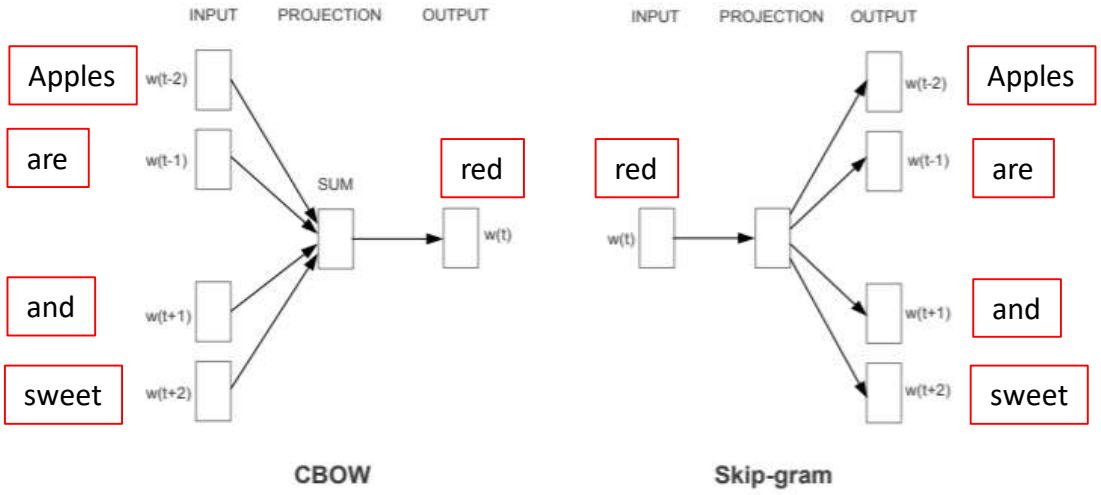
Existen atributos similares que permiten definir diferencias entre las palabras

Modelo de lenguaje Word2Vec



Fuente: https://medium.com/@hari4om/word-embedding-d816f643140

Modelos de lenguaje Word2vec



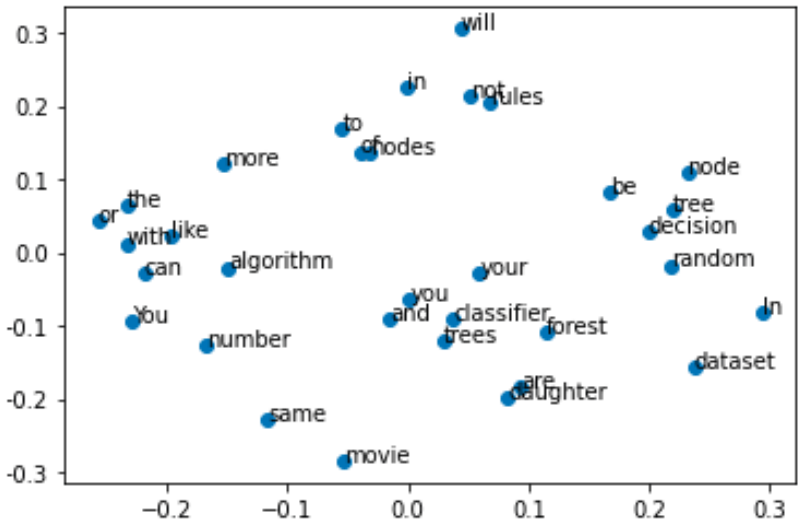
El modelo **CBOW** obtiene una representación (o predicción de una palabra central a partir de las palabras que se sitúan alrededor de ella (contexto)

El modelo **skip-gram** obtiene el contexto, representación (o predicción) de una palabra central a partir de las palabras que se sitúan alrededor de ella (contexto)

Modelos de lenguaje Word2vec Cbow

```python
# Predict center word using context words within window size
# import word2vec class from gensim
from gensim.models import Word2Vec
# apply word2vec to sentences
model_cbow = Word2Vec(sentences=word_tokenizer,
                      size=2, # number of dimensions default 100
                      window=2, # window size default 5
                      min_count=2, # minimum count of words to be consider default 5
                      workers=1, # workers default 3
                      sg=0 # method 0 for CBOW and 1 for skip gram
                      )

# total vocabulary words for word 2 vec model
words_cbow = list(model_cbow.wv.vocab)
print(f"Total number of words {len(words_cbow)}")
# we can get word embedding value for a particular word
model_cbow.wv.__getitem__('tree')
```

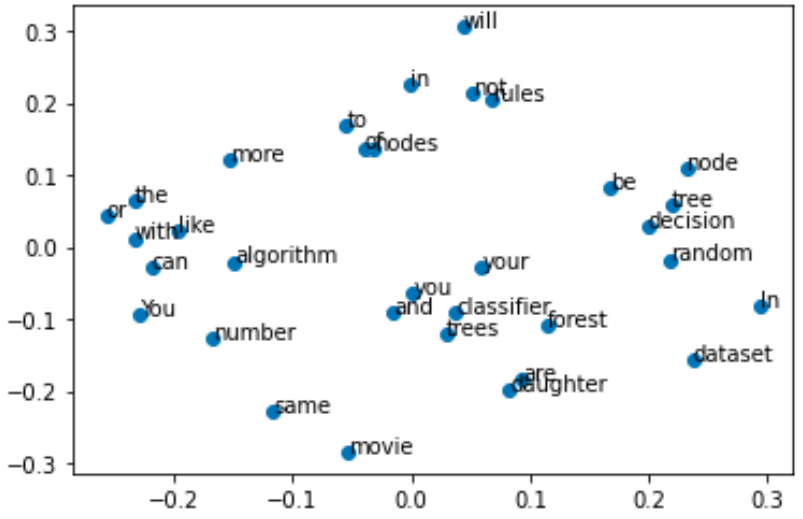
```python
X = model_cbow.wv.__getitem__(model_cbow.wv.vocab)
# create 2D model using PCA
pca_model = PCA(n_components=2)
result = pca_model.fit_transform(X)

# visualize pca model using matplotlib
plt.scatter(result[:,0], result[:,1])
words_cbow = list(model_cbow.wv.vocab)
for i, word in enumerate(words_cbow[:100]):
  plt.annotate(s=word,xy=(result[i,0], result[i,1]))
plt.show()
```



Modelos de lenguaje Word2vec skip-gram

```python
# import word2vec class from gensim
from gensim.models import Word2Vec
# apply word2vec to sentences
model_skip = Word2Vec(sentences=word_tokenizer,
                size=2, # number of dimensions default 100
                window=2, # window size default 5
                min_count=4, # minimum count of words to be consider default 5
                workers=1, # workers default 3
                sg=1 # method 0 for CBOW and 1 for skip gram
                )

# total vocabulary words for word 2 vec model
words = list(model_skip.wv.vocab)
print(f"Total number of words {len(words)}")

# we can get word embedding value for a particular word
wordembedding = model_skip.wv.__getitem__('tree')
print(wordembedding)

```

```python
X = model_skip.wv.__getitem__(model_skip.wv.vocab)
# create 2D model using PCA
pca_model = PCA(n_components=2)
result = pca_model.fit_transform(X)

# visualize pca model using matplotlib
plt.scatter(result[:,0], result[:,1])
words_skip = list(model_skip.wv.vocab)
for i, word in enumerate(words_skip[:100]):
  plt.annotate(s=word,xy=(result[i,0], result[i,1]))
plt.show()
```



Modelo de lenguaje Google Word2Vec

② Word2Vec ventana = 2

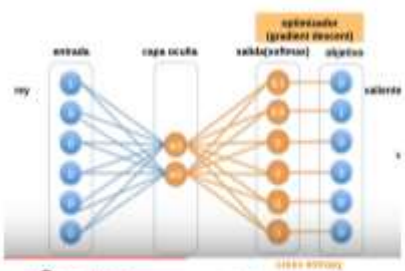
palabra	palabra one hot vector	vecino	vecino one hot vector
rey	[1,0,0,0,0,0]	valiente	[0,1,0,0,0,0]
rey	[1,0,0,0,0,0]	hombre	[0,0,1,0,0,0]
valiente	[0,1,0,0,0,0]	rey	[1,0,0,0,0,0]
valiente	[0,1,0,0,0,0]	hombre	[0,0,1,0,0,0]
hombre	[0,0,1,0,0,0]	rey	[1,0,0,0,0,0]
hombre	[0,0,1,0,0,0]	valiente	[0,1,0,0,0,0]
reina	[0,0,0,1,0,0]	hermosa	[0,0,0,0,1,0]
reina	[0,0,0,1,0,0]	mujer	[0,0,0,0,0,1]
hermosa	[0,0,0,0,1,0]	reina	[0,0,0,1,0,0]
hermosa	[0,0,0,0,1,0]	mujer	[0,0,0,0,0,1]
mujer	[0,0,0,0,0,1]	reina	[0,0,0,1,0,0]
mujer	[0,0,0,0,0,1]	hermosa	[0,0,0,0,1,0]

① Texto de entrenamiento

rey valiente hombre

reina hermosa mujer

③ Entrenamiento



④ Modelo de representación vectorial

palabra	vector	word embedding
rey	[1,0,0,0,0,0]	[1,1]
hombre	[0,0,1,0,0,0]	[1,3]
reina	[0,0,0,1,0,0]	[5,5]
mujer	[0,0,0,0,0,1]	[5,7]

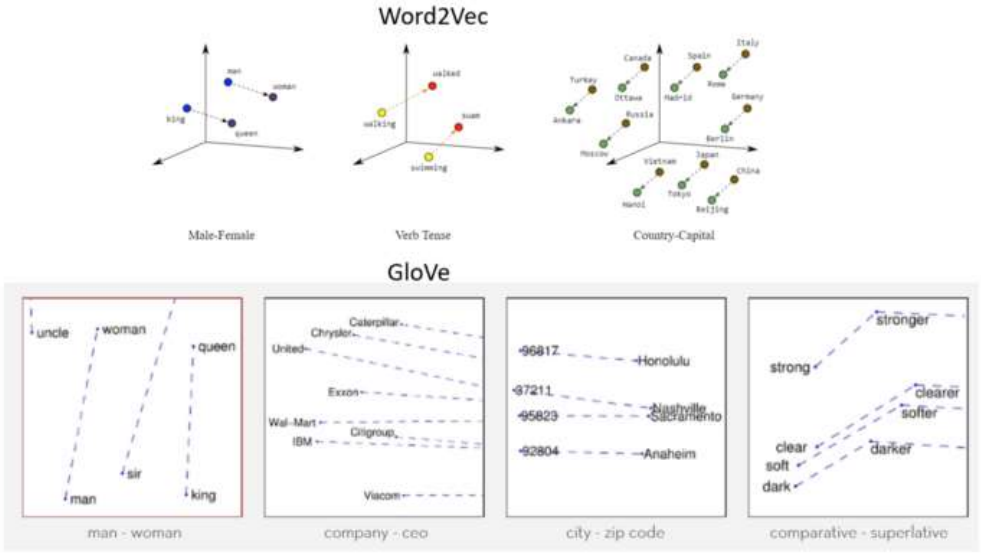

```python
from gensim.models import KeyedVectors
filename = 'GoogleNews-vectors-negative300.bin'
model = KeyedVectors.load_word2vec_format(filename, binary=True)
```

```python
# just predict or word from king-man+women = ?
result = model.most_similar(positive=['woman','king'], negative=['man'],topn=1)
print(f"Result ? mark in the following sentence King - man = ? - woman :- {result}")
```

Output

```
Result ? mark in the follwing sentence King - man = ? - woman :- [('queen', 0.7118192911148071)]
```

Modelos de lenguaje Glove



GloVe es un algoritmo de aprendizaje no supervisado para obtener representaciones vectoriales de palabras. El entrenamiento se realiza en estadísticas globales agregadas de co-ocurrencia palabra-palabra de un corpus, y las representaciones resultantes muestran subestructuras lineales interesantes del espacio vectorial de palabras.

Modelos de lenguaje: Word2vec Glove

```
# import glove2word2vec() from gensim
from gensim.scripts.glove2word2vec import glove2word2vec
# glove file total 4 subfiles with 50, 100, 200, and 300 dimensions
# load 100 dimensions word embedding file
from gensim.models import KeyedVectors
input_file = "glove.6B.100d.txt"
output_file = "glove.6B.100d.txt.word2vec"

# convert input file to word2vec format using glove2word2vec function
glove2word2vec(input_file, word2vec_output_file=output_file)
# load word2vec file to model , this file contain ASCII values
# so binary value is False
model = KeyedVectors.load_word2vec_format(output_file, binary=False)
# same example as above
result = model.most_similar(positive=['woman','king'], negative=['man'],topn=1)
print(f"Result :- {result}")
```

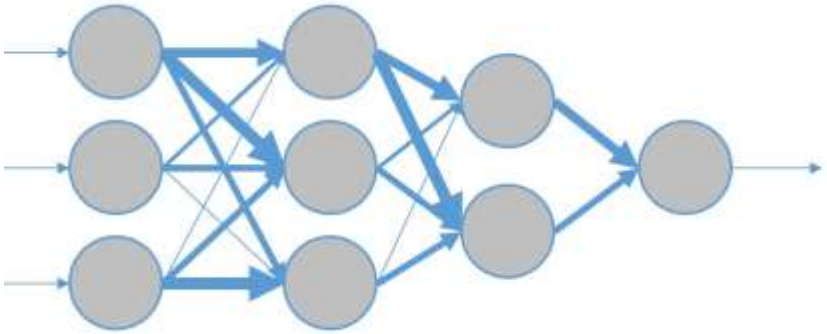
```
Result :- [('queen', 0.7698541283607483)]
```

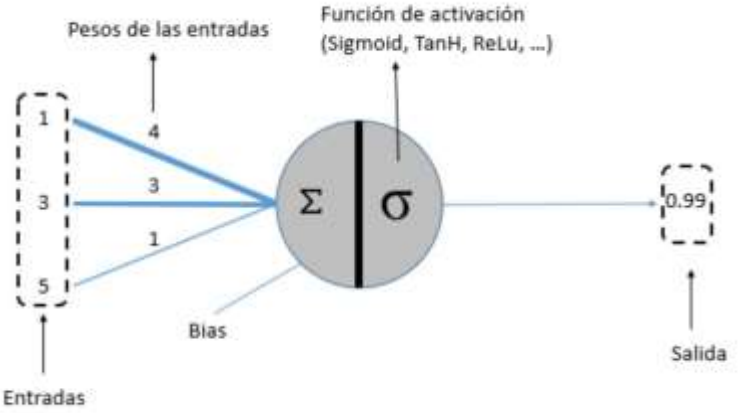
Para esto, tenemos que hacer una tarea solicitada previamente. Tenemos que convertir el archivo de glove word embedding a word2vec usando la función **glove2word2vec()**. Para estos archivos se ha tomado 100 dimensiones del archivo **glove.6B.100d.txt**

https://nlp.stanford.edu/data/glove.6B.zip

Modelos Neuronales

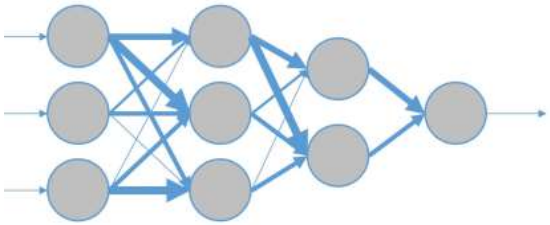
Las redes de neuronas son un tipo de técnica de Aprendizaje Automático, permite construir modelos de razonamiento mediante la simulación de los modelos cognitivos humanos, mediante funciones lineales y no lineales

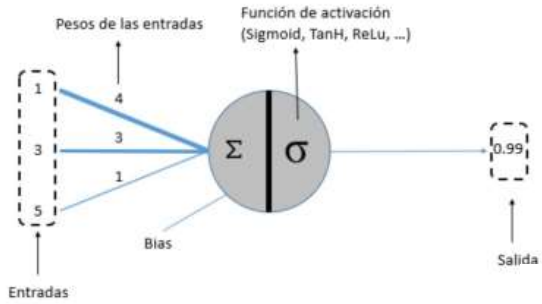


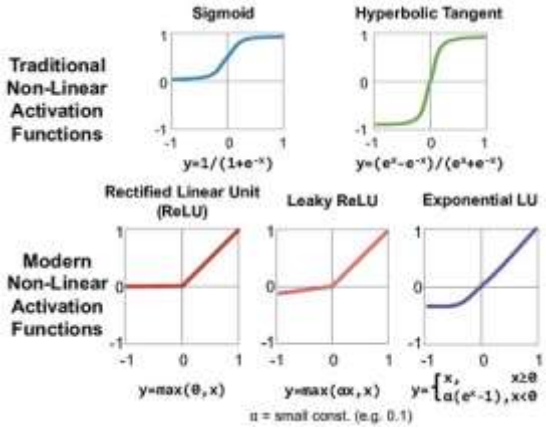


Modelos Neuronales

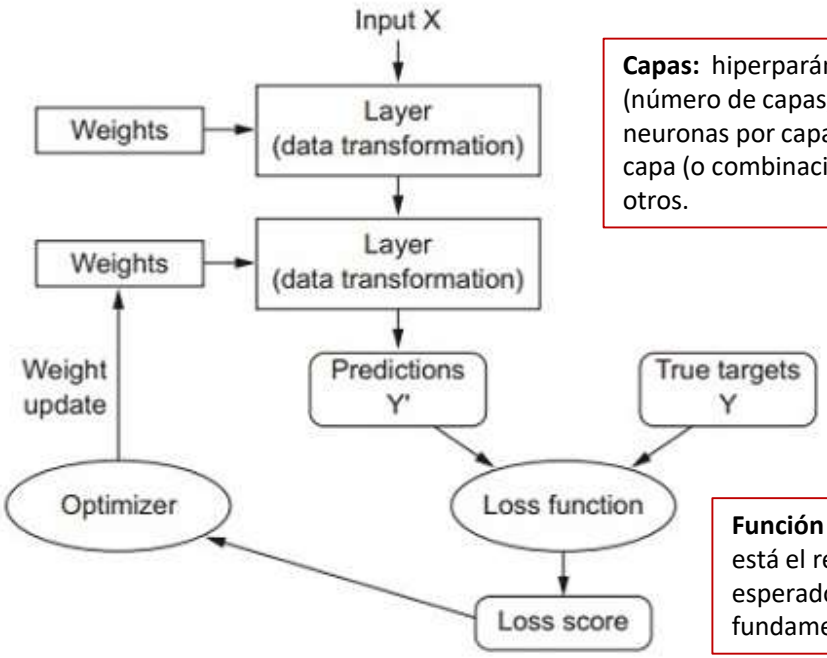
Las redes de neuronas son un tipo de técnica de Aprendizaje Automático, permite construir modelos de razonamiento mediante la simulación de los modelos cognitivos humanos, mediante funciones lineales y no lineales







Modelos Neuronales



Capas: hiperparámetros para capas (número de capas, número de neuronas por capa, etc), y tipo de capa (o combinaciones de capa) entre otros.

Predicciones: según la Función de activación, rige la salida de las neuronas

Optimizador: según la función de pérdida, 'decide' como modificar los parámetros de las capas, típicamente sus pesos.

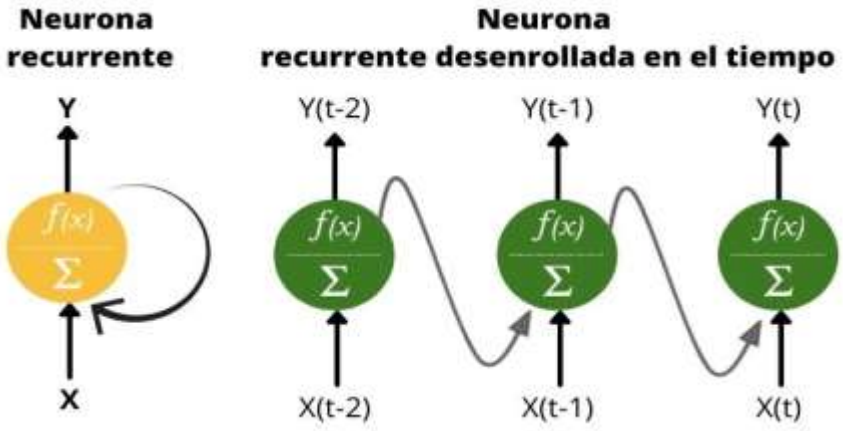
Función de pérdida: calcula cuán lejos está el resultado que produce la red del esperado, y que constituye la entrada fundamental para el aprendizaje.

Modelos Neuronales

Limitaciones:

- Las redes de neuronas están diseñadas para tener siempre el mismo tamaño de input y de output siempre. Sin embargo, **en los problemas secuenciales, los inputs y los outputs pueden tener tamaños muy diferentes dependiendo de la observación.**
  - Por ejemplo a la hora de traducir, cuando queremos trabajar tres palabras tenemos un input y output mucho más pequeños que cuando queremos traducir un párrafo.
- Las redes de neuronas no comparten características entre las diferentes posiciones. Es decir, **las NN asumen que cada input (y output) es independiente uno del otro.**

Redes de Neuronas Recurrentes



S O L

La casa amarilla

S

La

O

cas a

L

amarilla

Las redes de neuronas recurrentes pueden aprender una función de probabilidad

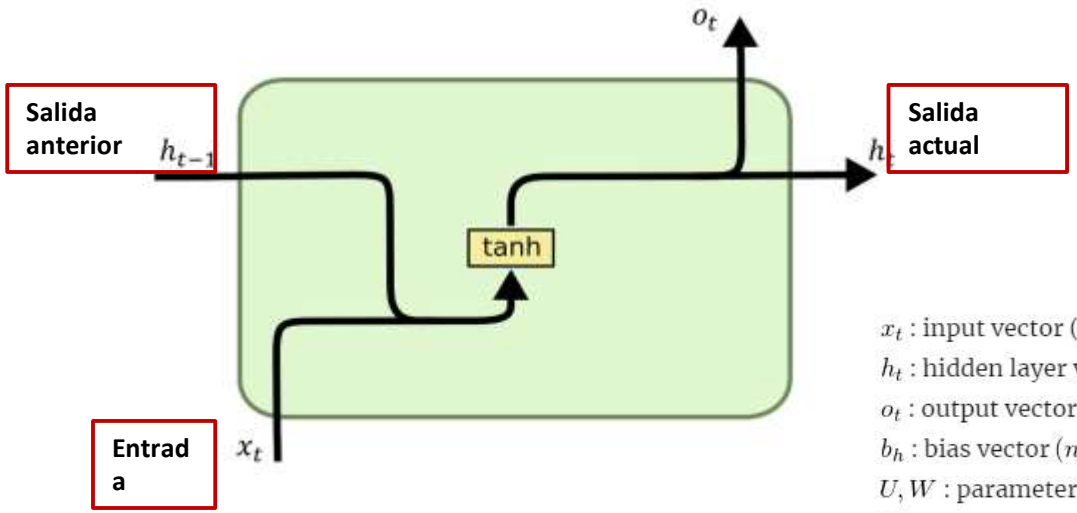
P= (Palabra|palabras previas)

¿Por favor podrías venir ahora?

Información previa

Predicción

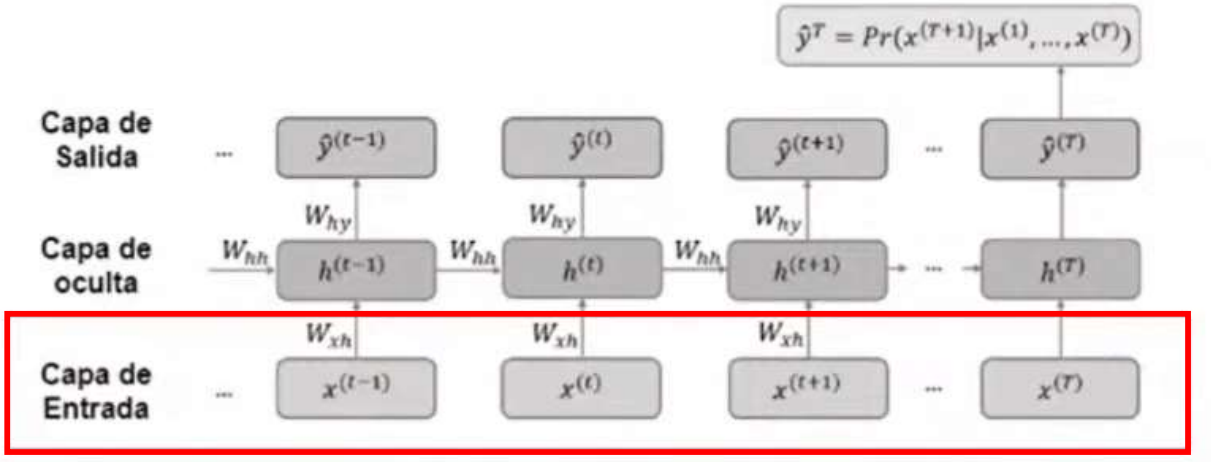
Neurona de una RNN Simple



x_t : input vector $(m \times 1)$.
h_t : hidden layer vector $(n \times 1)$.
o_t : output vector $(n \times 1)$.
b_h : bias vector $(n \times 1)$.
U, W : parameter matrices $(n \times m)$.
V : parameter matrix $(n \times n)$.
σ_h, σ_y : activation functions.

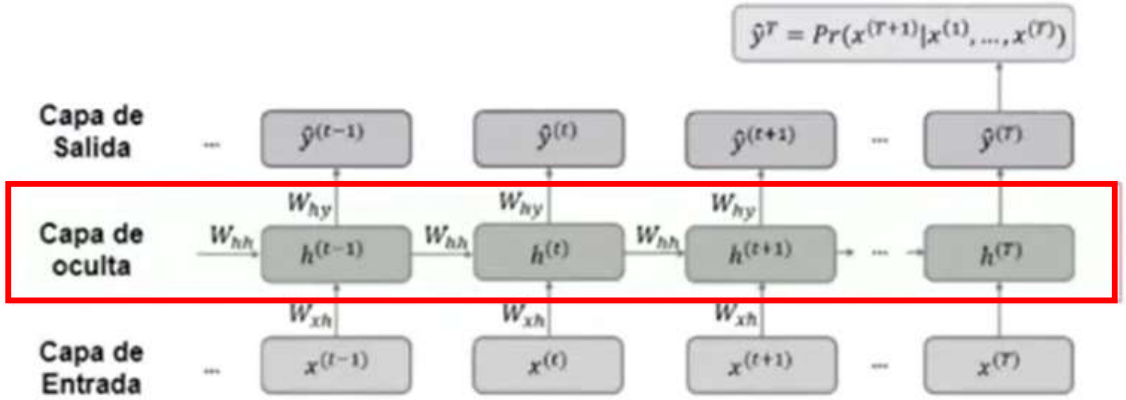
RNN Simple

Cada capa de entrada se corresponde con la información de entrada de la red y que, en la mayoría de los casos será una palabra
Las entradas deben codificarse mediante un vector de valores numéricos pudiendo ser one-hot o embeddings



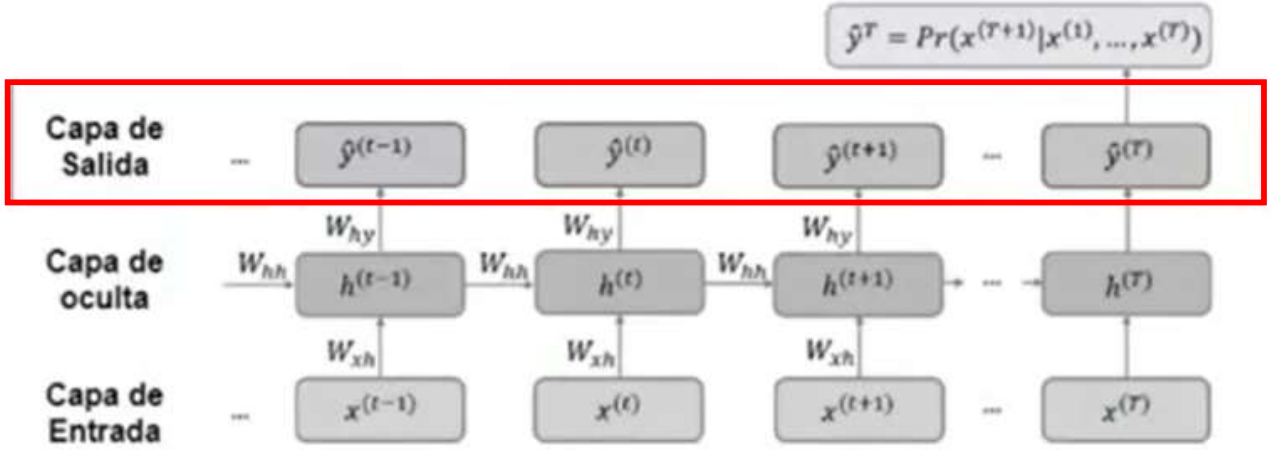
RNN Simple

Compuesta por un número finito de capas ocultas que corresponden a las capas intermedias de la red donde están los estados ocultos h(t), que corresponden con la salida de la capa oculta que será usada como entrada de la capa siguiente y como entrada a la propia capa con el objetivo de tener memoria a corto plazo.



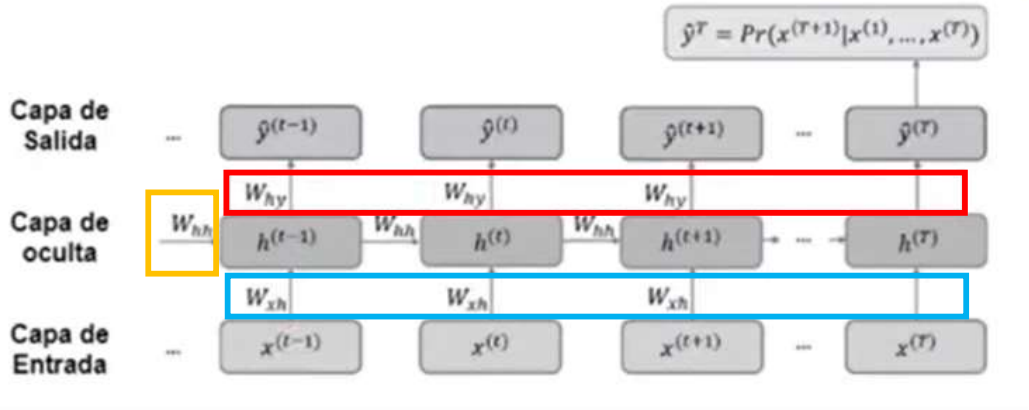
La entrada de las neuronas de la capa oculta combina la información de entrada x(t) y el valor del estado oculto h(t-1) que es la salida en el instante de tiempo inmediatamente anterior

RNN Simple



La capa de salida se utiliza para modelar una distribución de probabilidad para el instante de tiempo t

RNN Simple



Why Las funciones de activación de las diferentes capas ocultas se corresponden con funciones no lineales que normalmente son de tipo sigmoidal

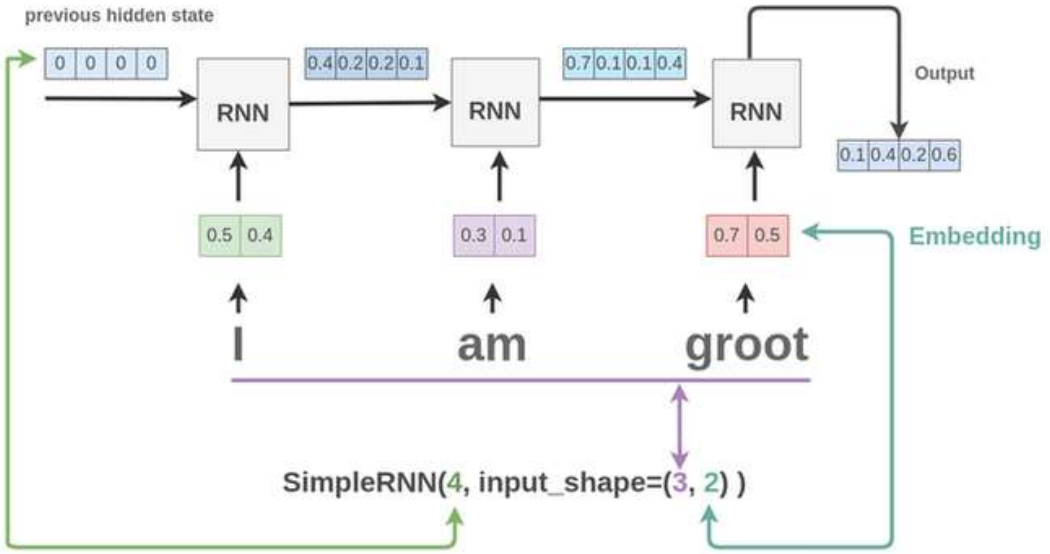
Estas funciones dependen del tipo de salida a emplear, como las de procesos de clasificación

Wxh es la matriz de pesos que conecta las neuronas de la capa de entrada con las neuronas de la capa oculta

Whh es la matriz de peso que conecta las neuronas de la capa oculta consigo mismas, simulando el proceso de memoria a corto plazo

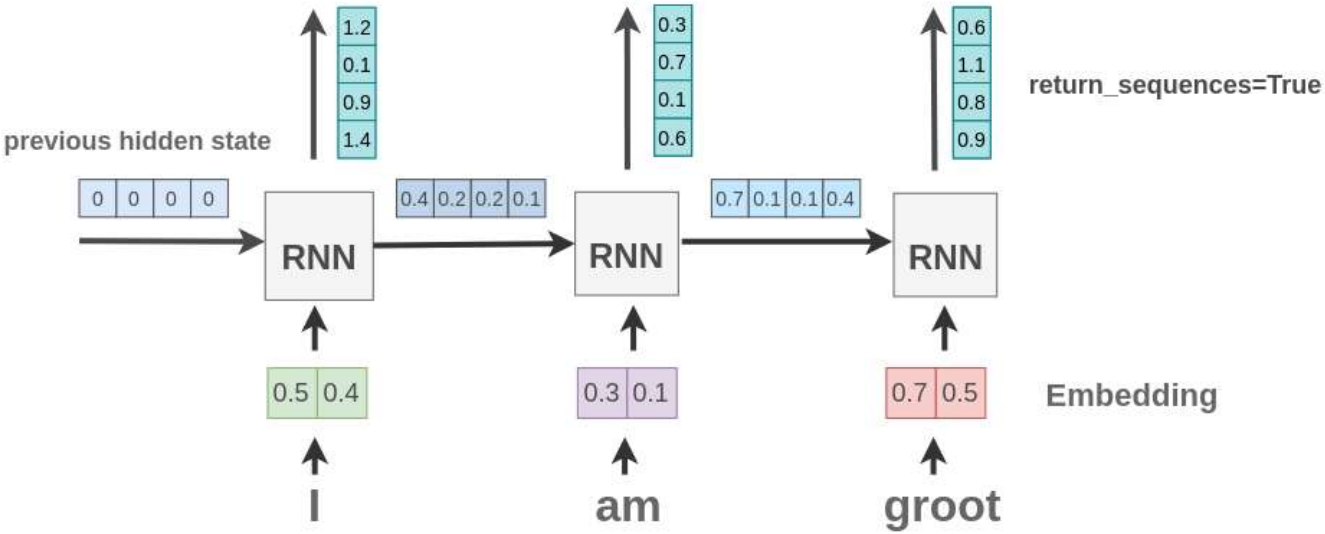
RNN Simple

Word	E1	E2
I	0.5	0.4
am	0.3	0.1
groot	0.7	0.5



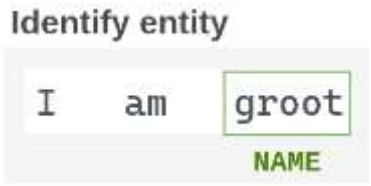

```
model = Sequential()
model.add(SimpleRNN(4, input_shape=(3, 2)))
model.add(Dense(1))
```

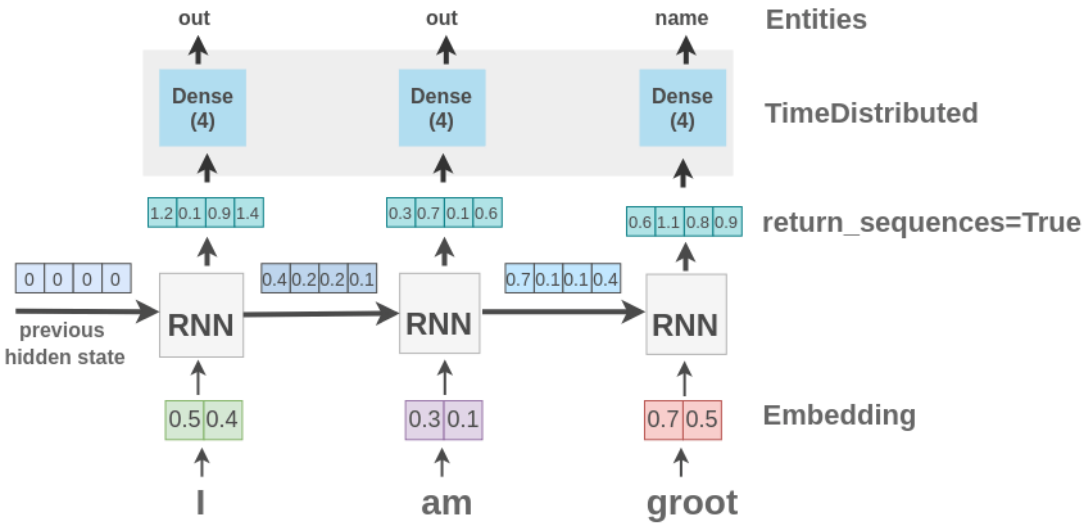
RNN Simple: Múltiples salidas




```
model = Sequential()
model = Sequential()
model.add(SimpleRNN(4, input_shape=(3, 2),
                    return_sequences=True))
```

RNN Simple: Time Distributed layer






```
model = Sequential()
model.add(SimpleRNN(4, input_shape=(3, 2),
                    return_sequences=True))
model.add(TimeDistributed(Dense(4, activation='softmax')))
```

RNN Simple

Ventajas

- Procesan entradas de orden secuencial, lo cual permite su uso en análisis de texto
- Procesan entradas (y salidas) de cualquier longitud, sin que estas longitudes afecten al tamaño del modelo
- Los pesos se comparten a lo largo del tiempo

Desventajas

- No consideran entradas futuras para el estado actual
- **Difícil acceso a información de estados muy antiguos**.
- **Desaparición/explosión del gradiente:** es complicado capturar las dependencias a largo plazo debido a la disminución/aumento exponencial del gradiente a medida que se crean más capas.

Casos de revisión

- Caso 1: BOW
- Caso 2: Embedding
- Caso 3: NN y RNN

Recursos

- https://pharos.sh/python-para-pnl-desarrollo-de-un-relleno-de-texto-automatico-con-n-grams/Caso 2: Preprocesamiento
- https://pharos.sh/python-para-pnl-introduccion-a-la-biblioteca-stanfordcorenlp/
- https://colab.research.google.com/github/stanfordnlp/stanza/blob/master/demo/Stanza_CoreNLP_Interface.ipynb#scrollTo=KxJeJ0D2LoOs
- https://nlp.stanford.edu/projects/glove/