ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE MADRID

NLP with Deep Learning

Eduardo Fernández Carrión

Evolución a las redes neuronales

- Ya por el **~2000** las RNN (GRU y LSTM) y las CNN estaban dando buenos resultados con series temporales.

Sin embargo, el uso de de NN en NLP los resultados no eran muy excitantes.

- En **2012**, acaba de publicarse Alexnet: una arquitectura pura de NN compleja que mostró unos resultados muy buenos en visión artificial.

Ante este escenario, la comunidad científica se volcó en usar NN también en NLP con arquitecturas más complejas, similares a las de AlexNet.

- Y en **2013** y **2014**, se publicó el algoritmo de Word2Vec y GloVe, respectivamente. Basados en NN.

Contenidos

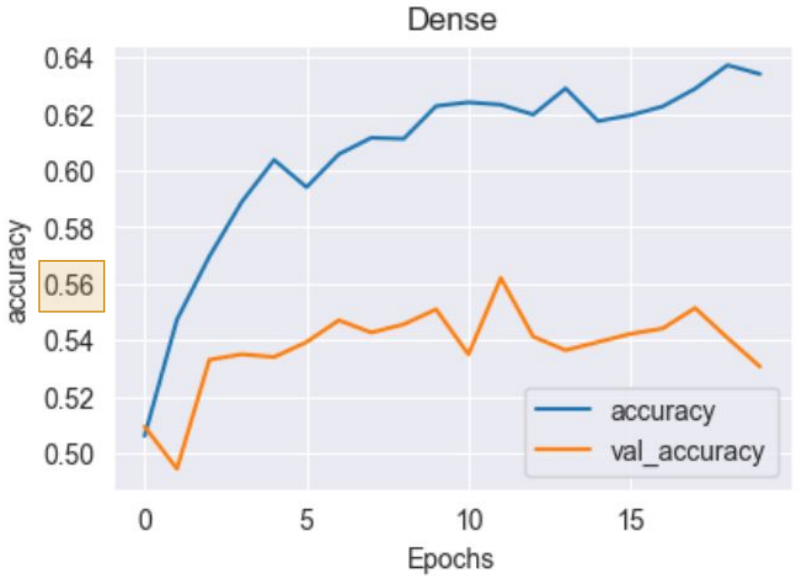
Crearemos modelos (sencillos)

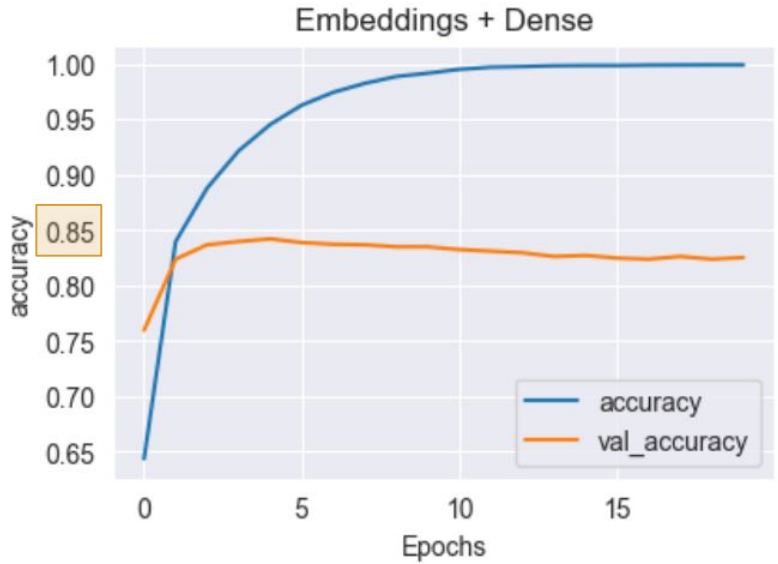
- con las arquitecturas más populares en NLP: CNN y LSTM.
- para dos casos de uso muy populares en NLP: Text Classification y Text Translation.
- con y sin arquitectura embeddings para observar la mejora

Neural Network	Text Classification	Text Translation
Dense	✓	
Dense + Embeddings	✓	
CNN + Embeddings	✓	
LSTM + Embeddings	✓	
LSTM + Embeddings		✓

Embeddings

La mejora que se consigue en todas las NN impulsó el uso de Deep Learning en NLP



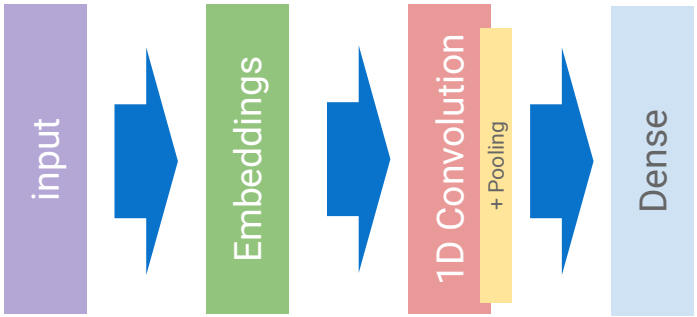


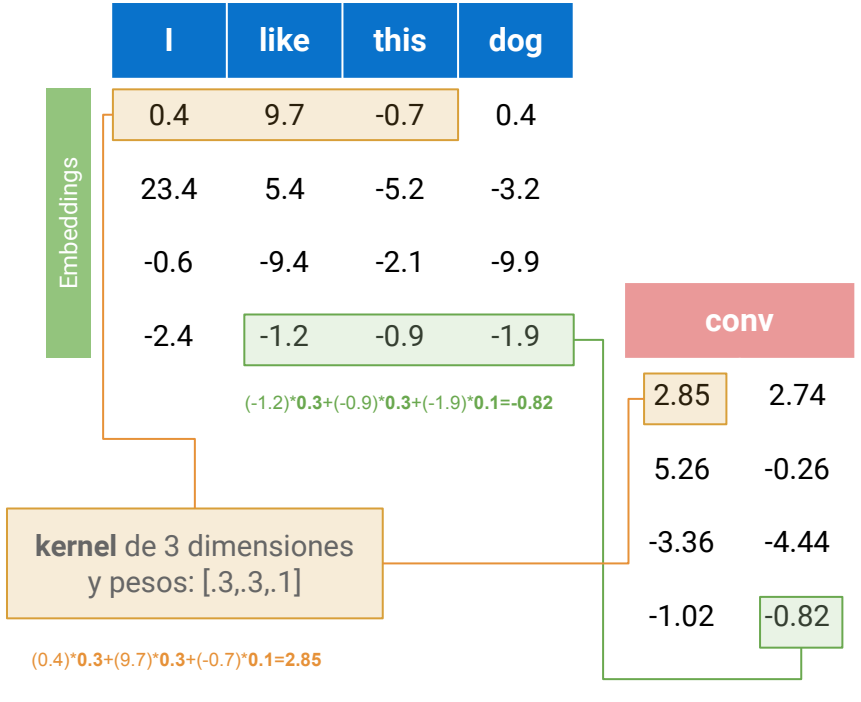
CNN

- Han sido adaptadas para NLP para identificar patrones locales o características en los datos, como la presencia de ciertas palabras o frases.
- Utilizan filtros (kernels) que se aplican a subsecuencias de la entrada para producir una representación condensada de características.
- Son efectivas para tareas de clasificación de texto y análisis de sentimientos, donde la presencia de ciertos n-gramas o combinaciones de palabras es más importante que el orden completo de las palabras.
- Generalmente más rápidas de entrenar y más fáciles de paralelizar que los modelos de secuencia.
- Combinadas con capas de Pooling, permiten destacar palabras más relevantes y disminuir el ruido.

1D Convolutions

La misma idea de la convolución en *computer vision* puede ser aplicada a series de secuencias ordenadas.





Keras CNN

```
keras.layers.Conv1D(
    filters,
    kernel_size,
    strides=1,
    padding="valid",
    data_format=None,
    dilation_rate=1,
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Pooling layers

- MaxPooling1D layer
- MaxPooling2D layer
- MaxPooling3D layer
- AveragePooling1D layer
- AveragePooling2D layer
- AveragePooling3D layer
- GlobalMaxPooling1D layer
- GlobalMaxPooling2D layer
- GlobalMaxPooling3D layer
- GlobalAveragePooling1D layer
- GlobalAveragePooling2D layer
- GlobalAveragePooling3D layer

Ventajas de las pooling layers:

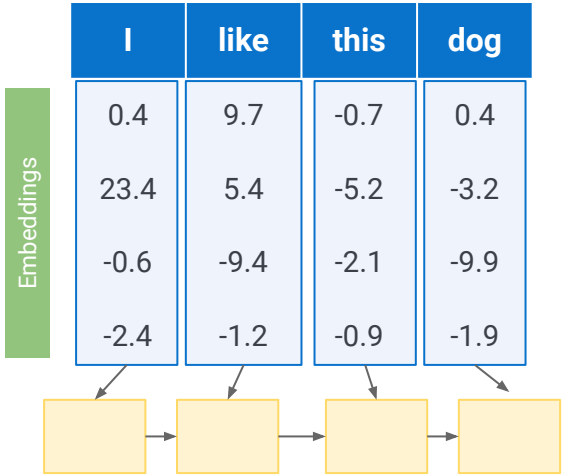
- Reducen dimensionalidad
- Controlan el overfitting
- Aumentan la robustez del modelo a variaciones y ruidos espaciales

LSTM

- Los modelos de secuencia (RNN, GRU, LSTM) están específicamente diseñados para manejar secuencias de datos. Por tanto, el orden y el contexto de las palabras son importantes.
- Capturan información a lo largo del tiempo manteniendo un estado (memoria) que se actualiza a medida que se procesa cada elemento de la secuencia.
- Sin embargo, los lenguajes deben capturarse en dos sentidos (bidireccional).
- Son particularmente buenos para tareas como modelado de lenguaje, generación de texto, y traducción automática.

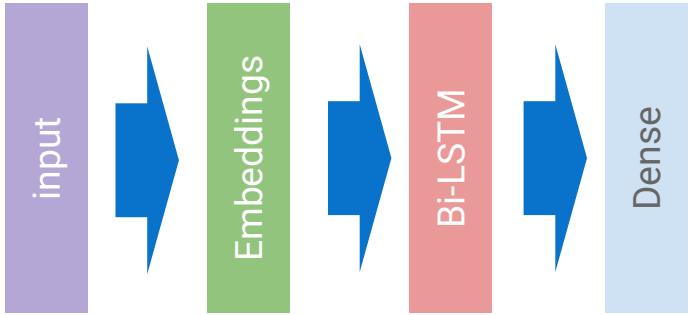
LSTM

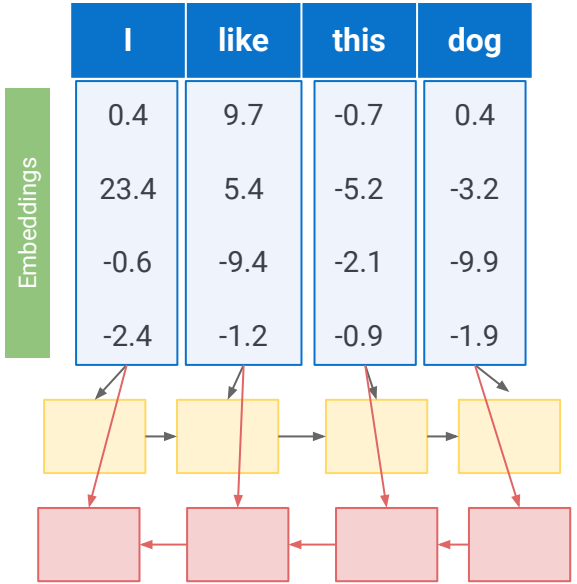
La misma idea de la RNN en *series temporales* puede ser aplicada a series de secuencias de texto ordenadas.



Bi-LSTM

A diferencia de series temporales, en lenguaje natural el contexto se extrae de oraciones completas. Por ejemplo, en alemán, el verbo está al final. O en inglés, los adjetivos no se colocan igual que en español.





Keras LSTM

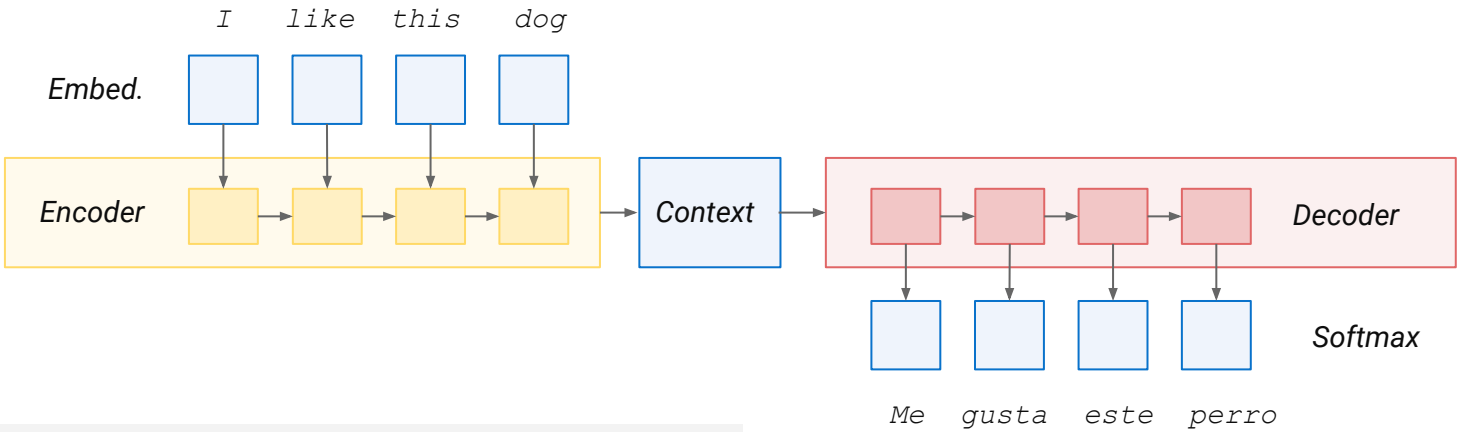
```
keras.layers.LSTM(
    units,
    activation="tanh",
    recurrent_activation="sigmoid",
    use_bias=True,
    kernel_initializer="glorot_uniform",
    recurrent_initializer="orthogonal",
    bias_initializer="zeros",
```

```
keras.layers.Bidirectional(
    layer, merge_mode="concat", weights=None, backward_layer=None, **kwargs
)
```

Encoder-Decoder (seq2seq)

Las arquitecturas de tipo **seq2seq** (secuencia a secuencia), también conocidas como **encoder-decoder**, son modelos de aprendizaje profundo diseñados para transformar **secuencias de entrada en secuencias de salida**.

Se compone de dos partes principales: el codificador (encoder) y el decodificador (decoder)



Son especialmente útiles para tareas como la traducción automática y la generación de texto.

Encoder-Decoder

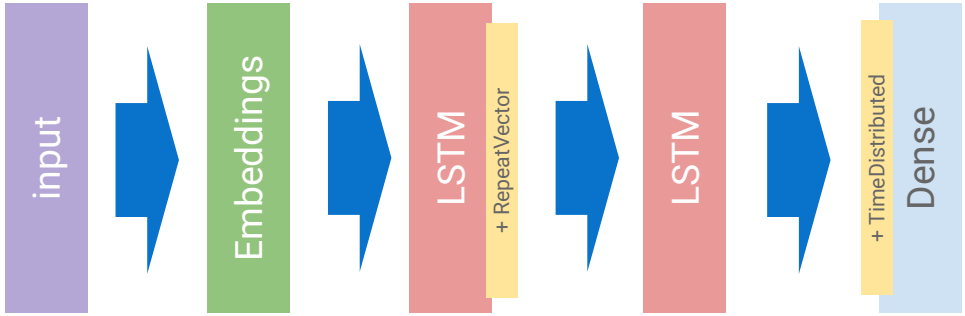
Encoder

- Toma una secuencia de entrada y la procesa para generar una representación compacta de esta.
- El propósito del encoder es comprender y codificar la información de entrada en un formato que el modelo pueda utilizar para realizar la tarea deseada.
- En tareas de traducción, por ejemplo, el encoder leería la oración en el idioma fuente y la transformaría en un conjunto de vectores que capturan su significado.

Decoder

- Toma el vector de contexto generado por el encoder y comienza el proceso de generar la secuencia de salida, paso a paso.
- En cada paso, se basa en el estado actual y la información recibida del encoder para predecir el siguiente elemento de la secuencia de salida. Continúa este proceso hasta que se genera una señal de parada o hasta que se alcanza un límite predefinido de longitud de secuencia.
- En el caso de la traducción, el decodificador usaría el vector de contexto para empezar a generar la oración en el idioma objetivo, palabra por palabra.

Keras LSTM



RepeatVector layer

`RepeatVector` class

```
keras.layers.RepeatVector(n, **kwargs)
```

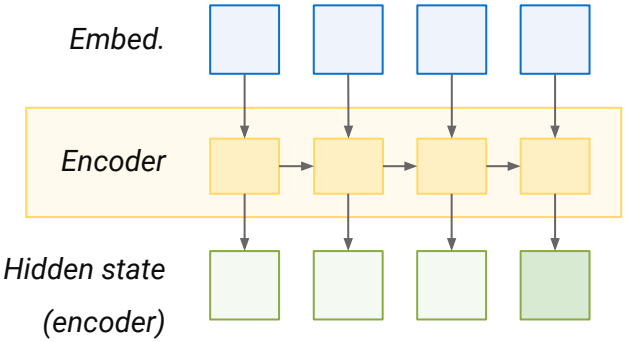
TimeDistributed layer

`TimeDistributed` class

```
keras.layers.TimeDistributed(layer, **kwargs)
```

Encoder

1. LSTM con tantos time steps como mi secuencia de entrada



2. Repetimos el último hidden state, tantas veces como sea mi secuencia de salida

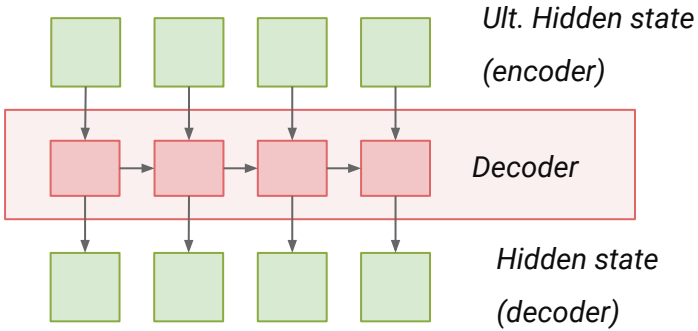
return_sequences=True → Devuelve todas las hidden states

return_sequences=False → Devuelve la última hidden state

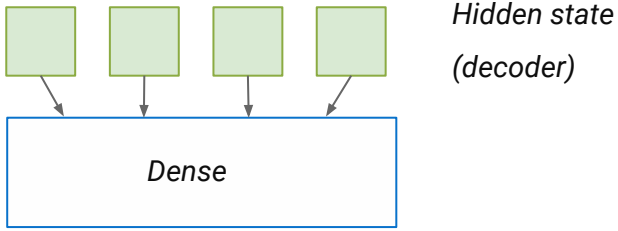
La dimensión de cada hidden state la configuramos en el LSTM *(units)*

Decoder

3 LSTM con tantos time steps como mi secuencia de salida



4 Una capa Densa con la entrada de todas las hidden states



Aplicamos la capa dense (+softmax) para predecir cada palabra que proviene de cada hidden state. Y se hace iterativamente.

ntic master
revolucionamos la comunicación

UNIVERSIDAD COMPLUTENSE MADRID