Taller de Procesamiento de Lenguaje Natural

Modelos neuronales de etiquetado de secuencias

Escuela de Verano – EVIA 2023

David Vilares

Etiquetado de secuencias

Asignar a cada palabra de una oración una, y una única, etiqueta.

El	volcán	emitió	mucha	lava	durante	la	erupción
DET	NOUN	VERB	DET	NOUN	PREP	DEP	NOUN

Es la aproximación más simple de predicción estructurada.

Consiste en convertir texto plano en una representación con anotaciones sobre relaciones entre palabras o sus propiedades que pueden ser luego procesadas para extraer información.

Etiquetado de secuencias

Ventajas

Rápidos, potencialmente $\Theta(n)$, n es el tamaño de la oración.

Fáciles de entender y de implementar (lo veremos hoy!).

No es un concepto limitado a PLN. También se etiquetan secuencias de ADN, por ejemplo.

Etiquetado de secuencias

Desventajas

Sirven para extraer información superficial de la oración.

Para extraer información y relaciones más complejas entre palabras, se emplean algoritmos de análisis de árboles y grafos.

Tareas abordables como etiquetado de secuencias

Reconocimiento de entidades nombradas o *named-entity recognition* (NER).

Etiquetación morfológica o *part-of-speech (PoS) tagging*.

Etiquetación de roles semánticos o *semantic role labeling*.

Multi-span question answering as sequence labeling.

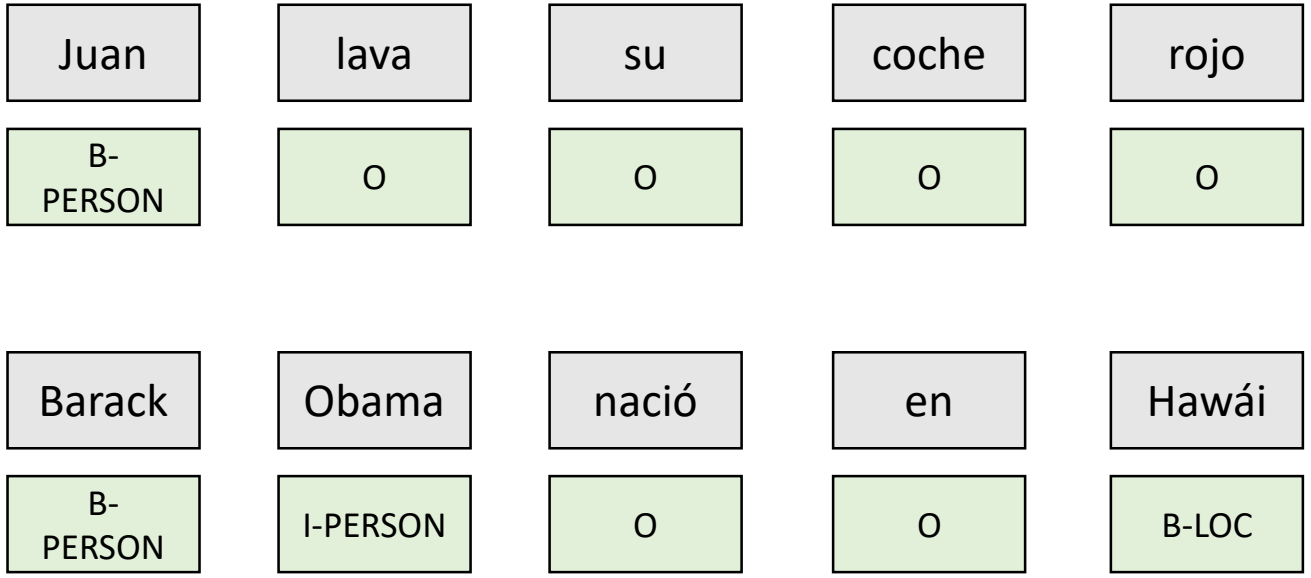
Named-entity recognition (NER)

Reconocimiento de entidades nombradas.

Identificar información clave en texto escrito, habiendo definido previamente un conjunto de categorías (nombres de personas, localizaciones, organizaciones, etc).

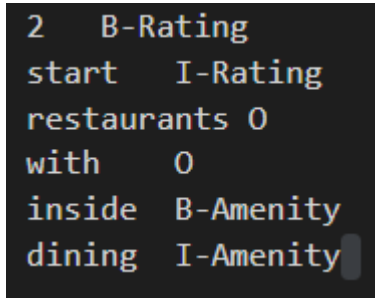
Una entidad puede definirse como un concepto del que se habla frecuentemente en un dominio. Por ejemplo, nombres de actores en una película, o los platos en un menú de un restaurante.

Named-entity recognition (NER)



Named-entity recognition (NER)

Suelen anotarse con el criterio de anotación BIO:

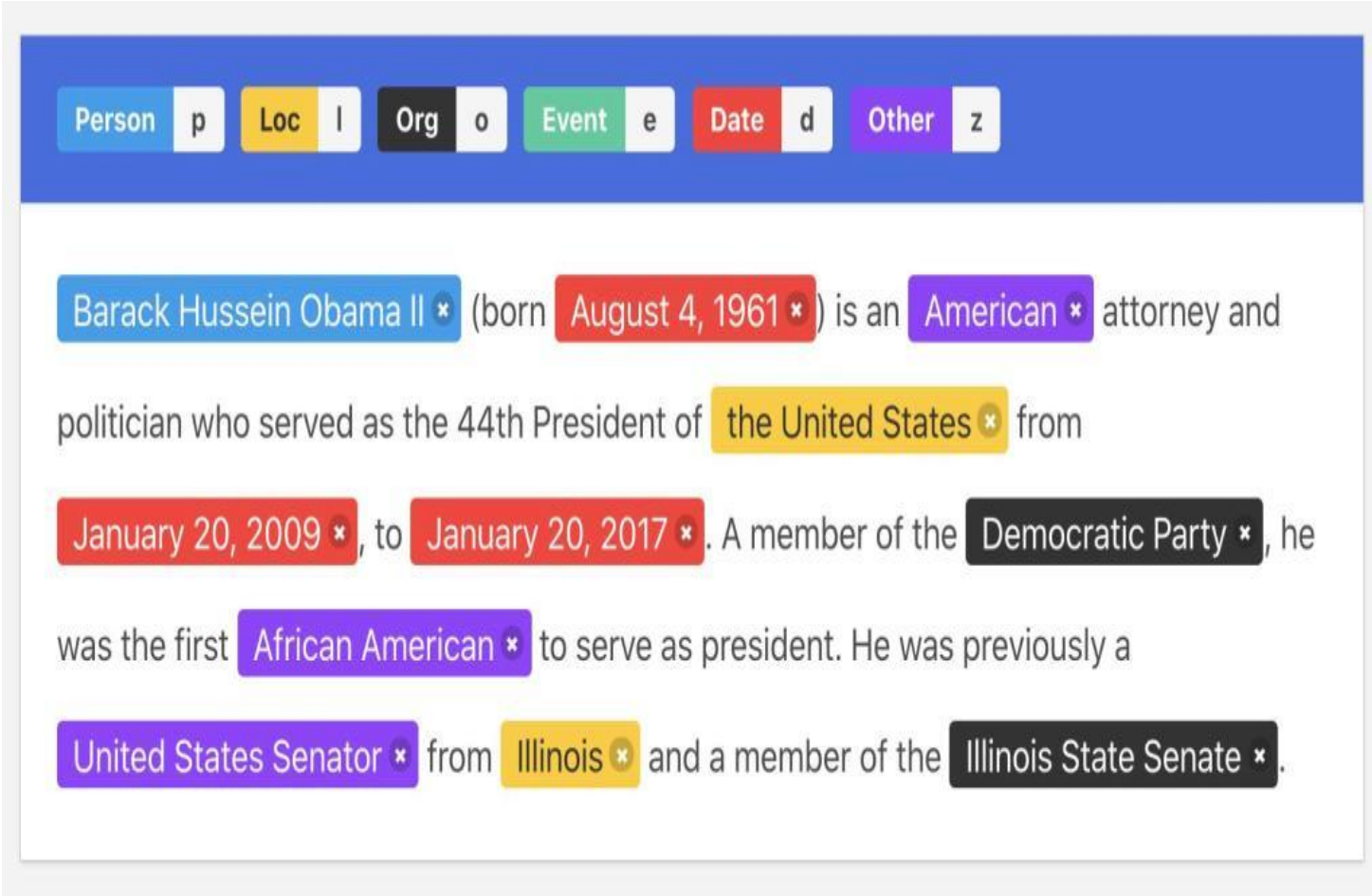


B: Indica que con esa palabra comienza una entidad.

I: Indica que con esa palabra continúa el reconocimiento de una entidad que se abrió anteriormente (etiquetando una palabra previa con una B).

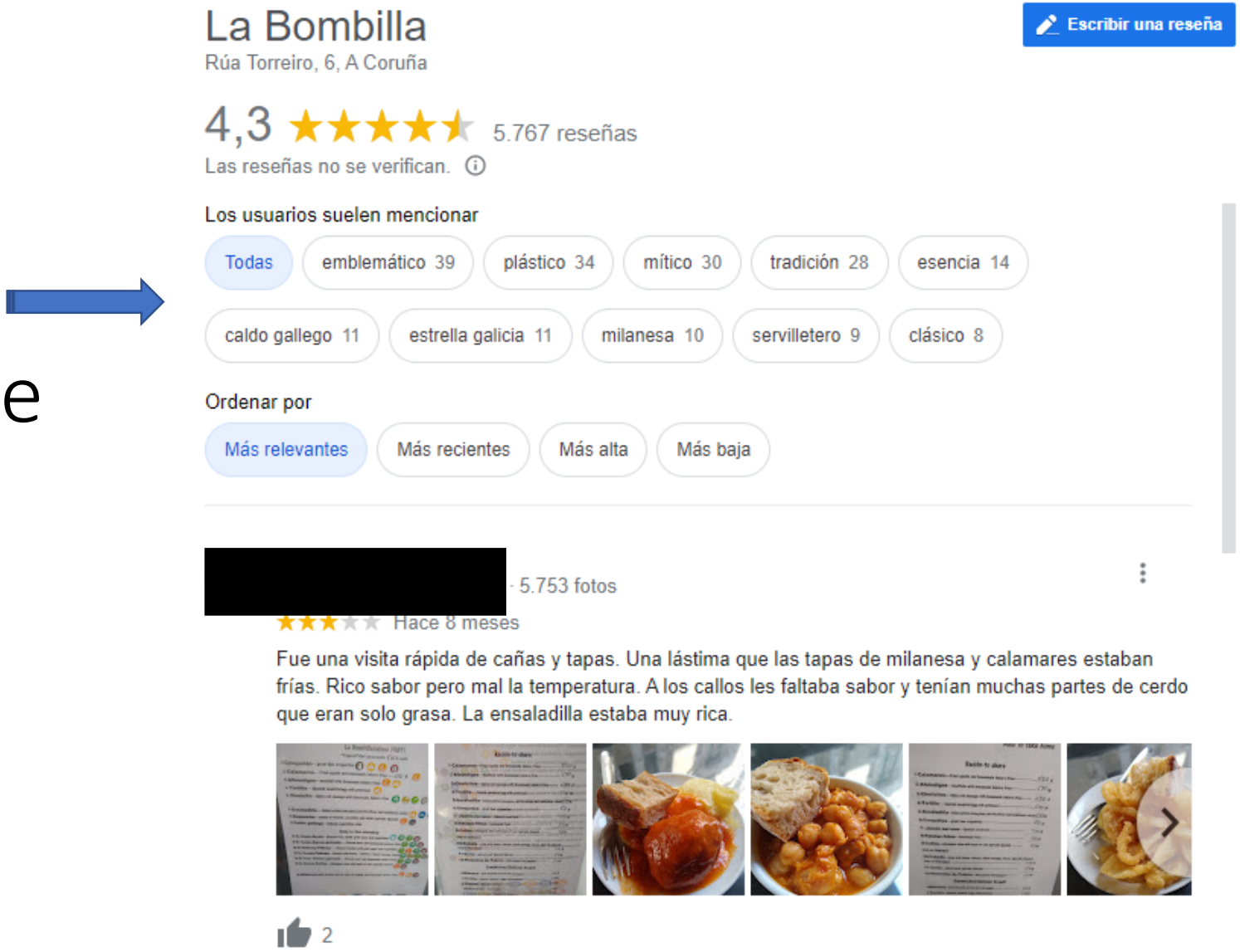
O: Etiqueta usada para identificar aquellas palabras que no aportan información relevante para el dominio.

Aplicaciones de named-entity recognition (NER)



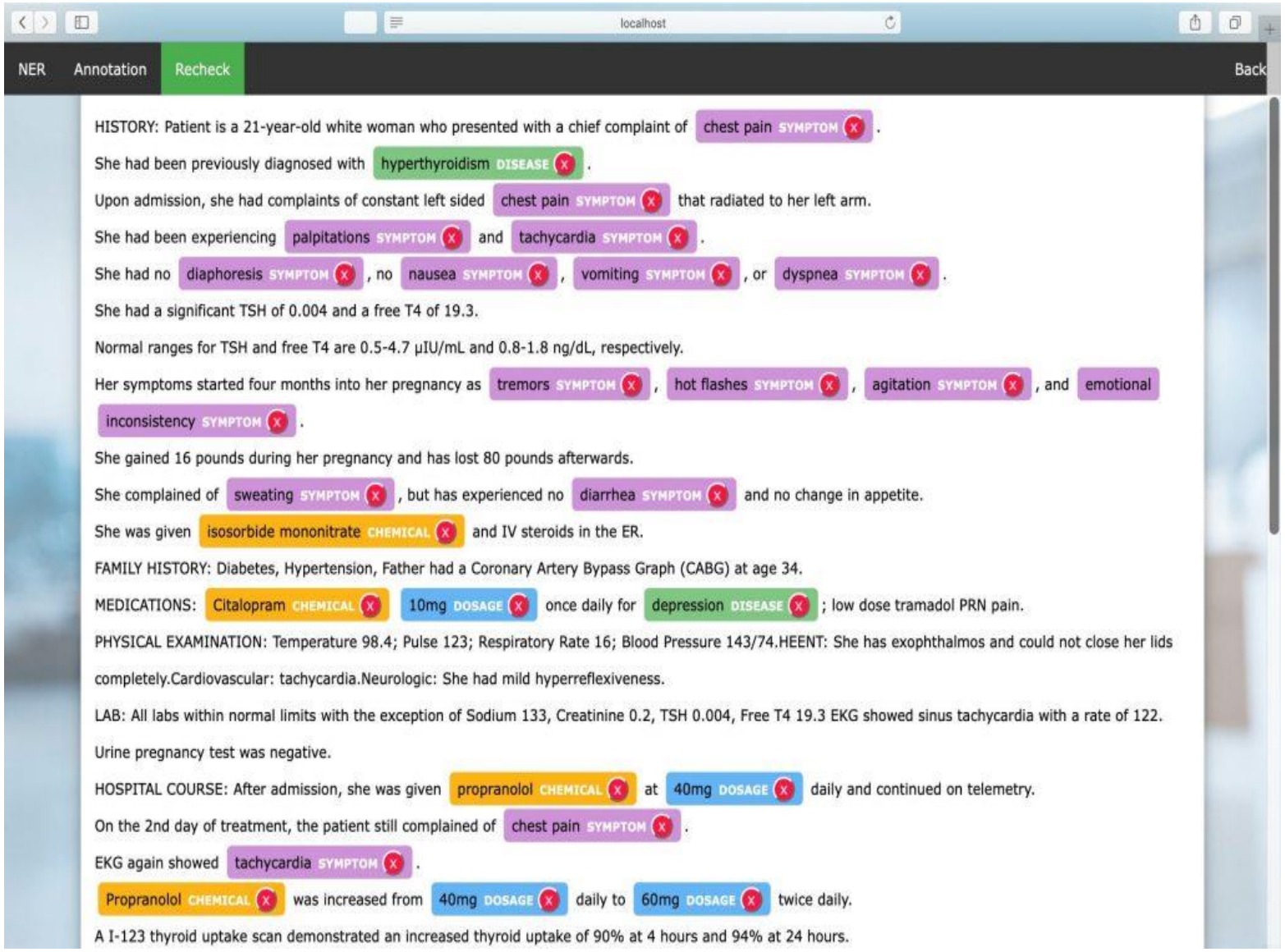
Extraída de: https://www.analyticsvidhya.com/blog/2021/11/a-beginners-introduction-to-ner-named-entity-recognition/

Aplicaciones de named-entity recognition (NER)



Extraída de Google Reviews

Aplicaciones de named-entity recognition (NER)



Extraída de: https://www.persistent.com/blogs/building-named-entity-recognition-models-for-healthcare/

Modelos de etiquetado de secuencia

Metodología estándar para la creación de los modelos.

Fase entrenamiento/desarrollo:

- Un conjunto de entrenamiento (train.txt) y otro de desarrollo (dev.txt), comúnmente en texto plano, con anotaciones a nivel de palabra.
- Un modelo supervisado capaz de aprender a partir de dichos datos.

Evaluación:

- Un conjunto de test (test.txt) que el modelo no haya visto antes, para estimar su rendimiento en entornos reales.

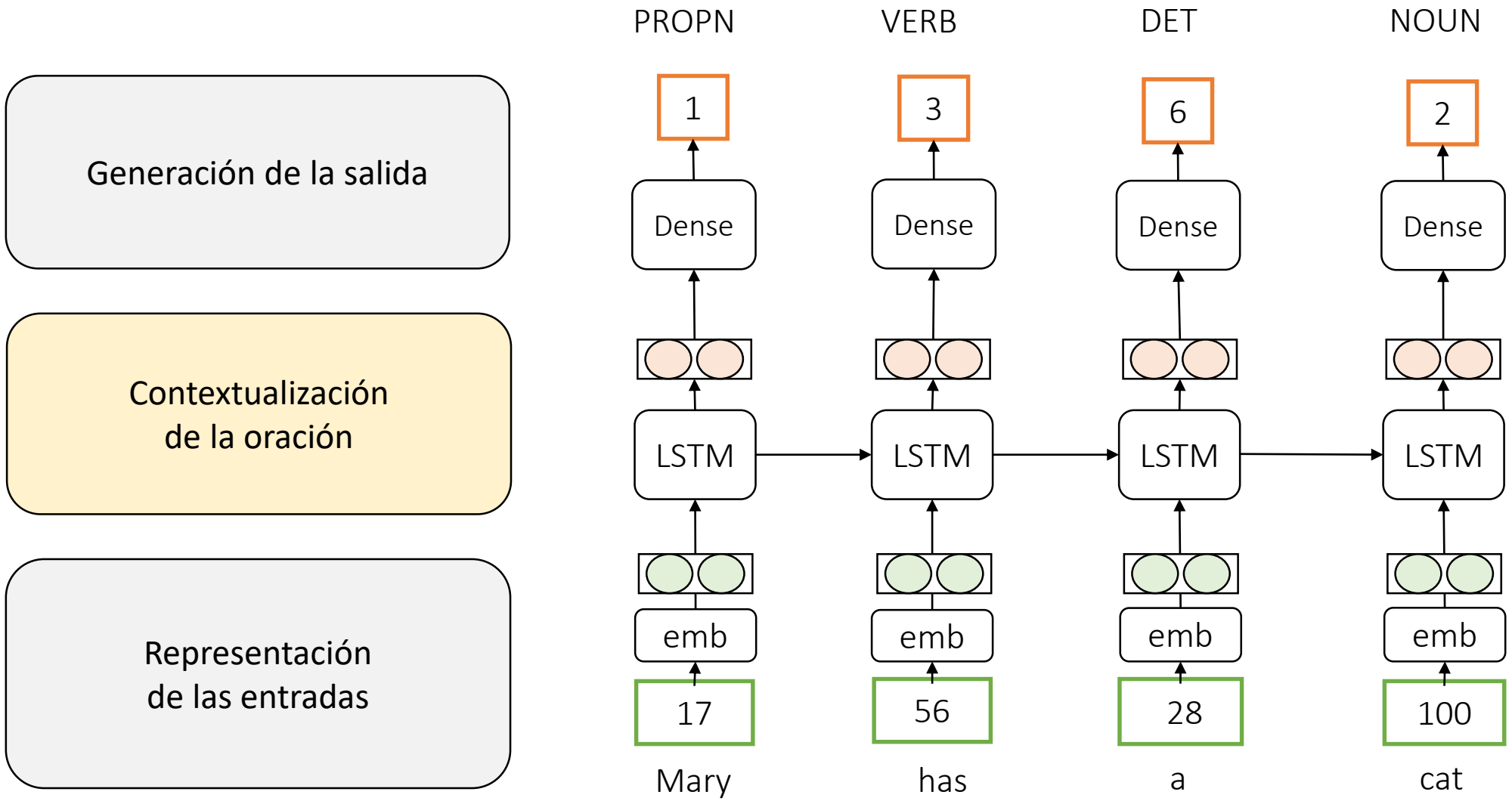
Modelos de etiquetado de secuencia

Snippets y feed-forward networks: Modelos con contexto limitado.

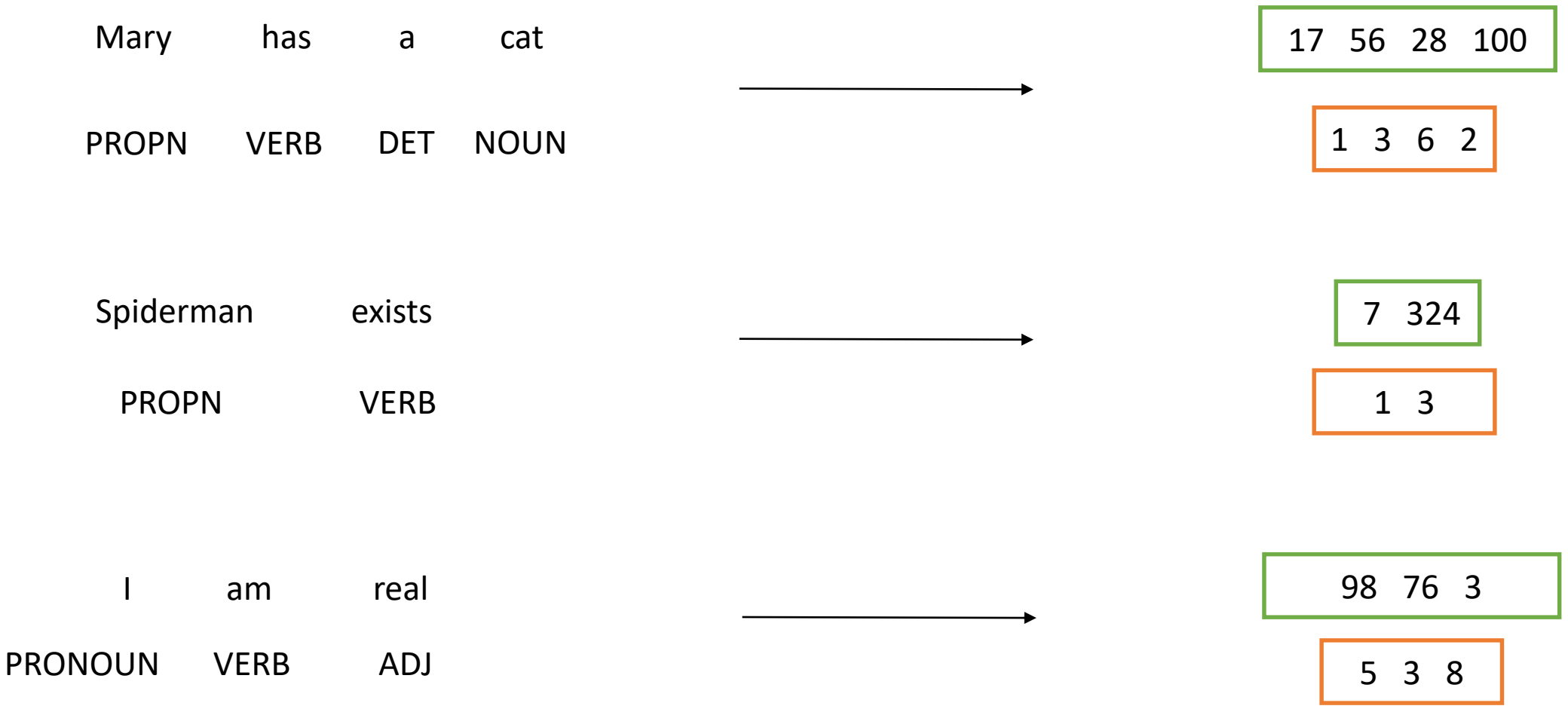
Long short-term memory networks (LSTMs): Modelos que pueden recordar u olvidar información sobre secuencias largas, lo que permite capturar dependencias lejanas en los datos.

Transformers: Modelos con una excelente capacidad parar capturar, mediante técnicas de self-attention, el contexto relevante para una palabra (independientemente de la distancia a la tarea objetivo) y para transferir conocimiento una vez han sido pre-entrenados como modelos de lenguaje sobre grandes cantidades de texto (BERT, RoBERTa, GPT, LLaMa).

Long short-term memory networks



Long short-term memory networks – Representación de las entradas

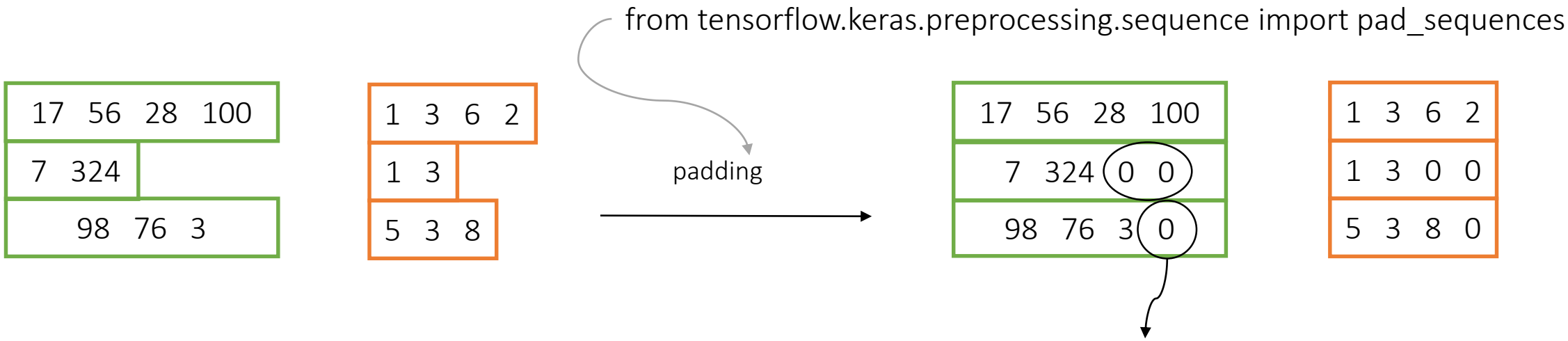


Long short-term memory networks – Representación de las entradas

Tendremos oraciones con diferentes longitudes.

Todas las oraciones en un batch de entrenamiento deben tener la misma longitud para que puedan ser enviadas a un modelo de Keras

Solución: incluir padding.



Long short-term memory networks – Representación de las entradas

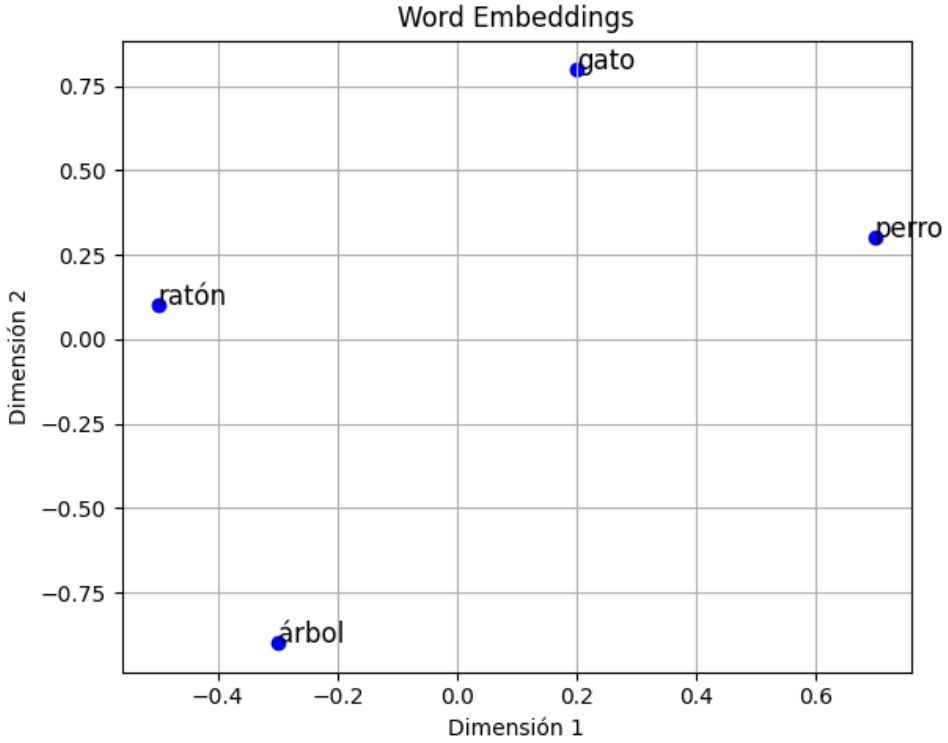
Embeddings: representación de una palabra (elemento discreto) como un vector de baja dimensionalidad. Palabras que ocurren en contextos similares tendrán *embeddings* similares.

"gato": [0.2, 0.8]

"perro": [0.7, 0.3]

“ratón": [-0.5, 0.1]

"árbol": [-0.3, -0.9]



Long short-term memory networks – Representación de las entradas

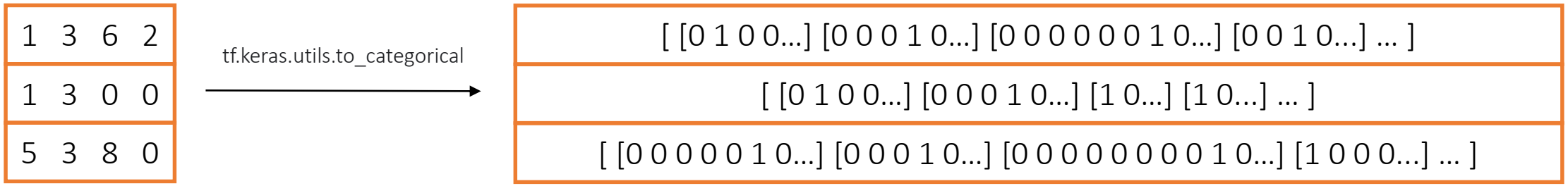
Embeddings inicializadas aleatoriamente y entrenadas conjuntamente con el resto de la red para la tarea objetivo. Palabras con contextos similares deberían tener embeddings cercanas al final del entrenamiento.

Embeddings estáticas: calculadas previamente (word2vec, GloVe) y enchufables a la red. Una misma palabra tiene siempre la misma embedding, independiente del contexto en el que aparezca.

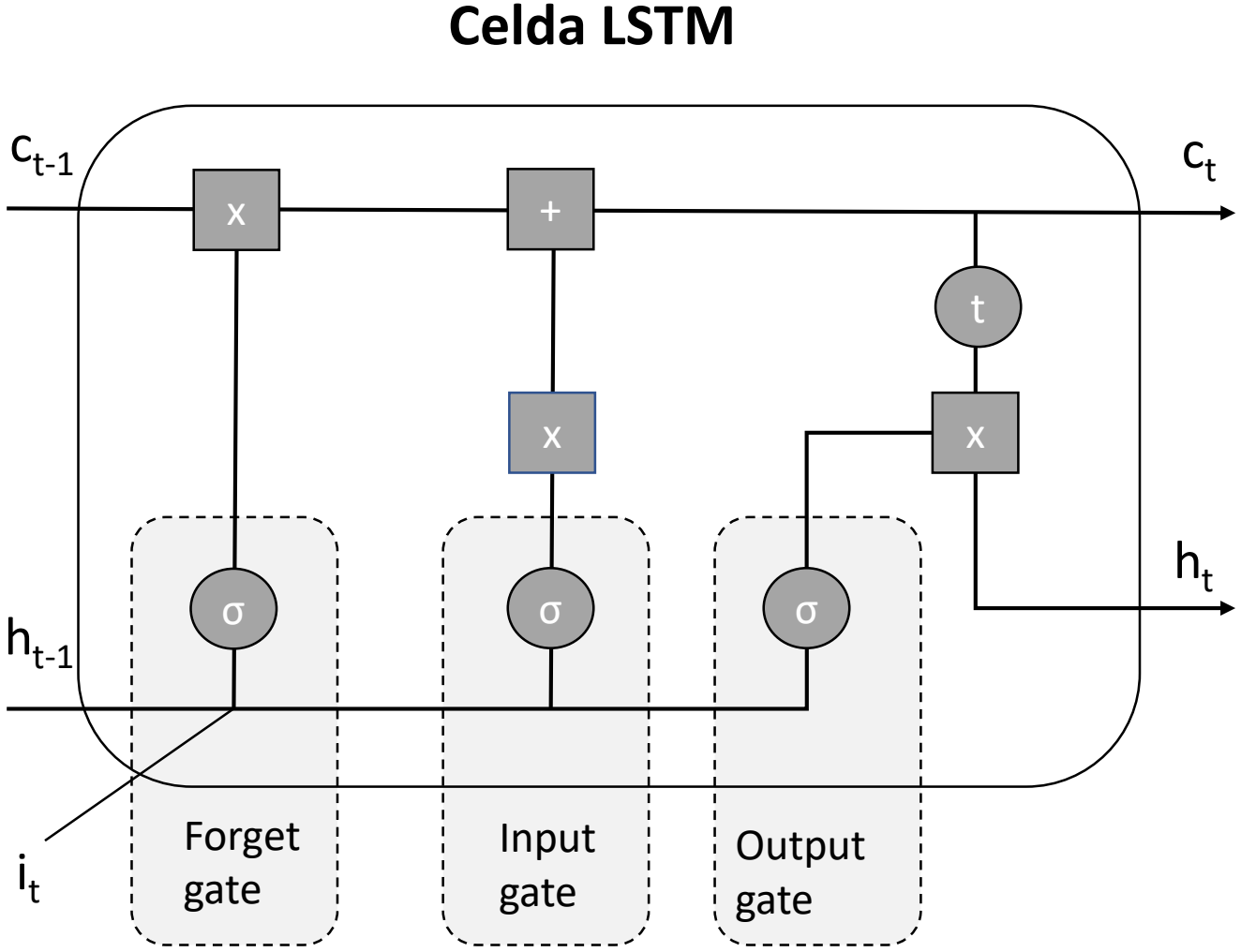
Embeddings **contextualizadas**: calculadas a través de un modelo de lenguaje que ha sido pre-entrenado (BERT, GPT) y que se conectan al resto de la red.

Long short-term memory networks – Representación de etiquetas de salida

Necesitaremos transformar las etiquetas de salida (inicialmente representadas como una cadena de caracteres) en un one-hot vector que identifica la clase que se debe predecir.



Long short-term memory networks



Cell state: La "memoria" de la LSTM. En cada *timestep* (cada palabra en el caso de problemas de PLN) se actualiza en base a la forget y la input gate. Se almacena en c_t.

Input gate: Determina que valores de la entrada en un timestep dado añadir al cell state.

Forget gate: Determina que valores eliminar del cell state para un timestep dado.

Output gate: Determina la cantidad de cell state que transmitir a la salida, que se manifiesta en el hidden output state (h_t).

Long short-term memory networks – Definición de la red

Capa de entrada: https://www.tensorflow.org/api_docs/python/tf/keras/Input

Capa de embeddings: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

Capa LSTM: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

Capa bidireccional: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional

Capa Dense: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense

Capa TimeDistributed [importante!]: https://www.tensorflow.org/api_docs/python/tf/keras/layers/TimeDistributed

tf.keras.layers.TimeDistributed

Un wrapper para aplicar una capa a cada timestep de una entrada temporal (como por ejemplo una oración).

La dimension en el eje 1 es la que se considera como la dimension temporal.

En nuestro caso, la forma de la entrada para nuestro etiquetador es

(batch_size, max_sentence_length, word_embedding_size).

Dada una capa TimeDistributed, aplicará la capa ‘envuelta’ a todos los elementos de entrada de la secuencia.

tf.keras.layers.TimeDistributed

En este caso usaremos la capa TimeDistributed para aplicar la capa de clasificación de salida Dense sobre la representación oculta de cada palabra.

Se aplicará a cada uno de los vectores de salida generados por la LSTM.

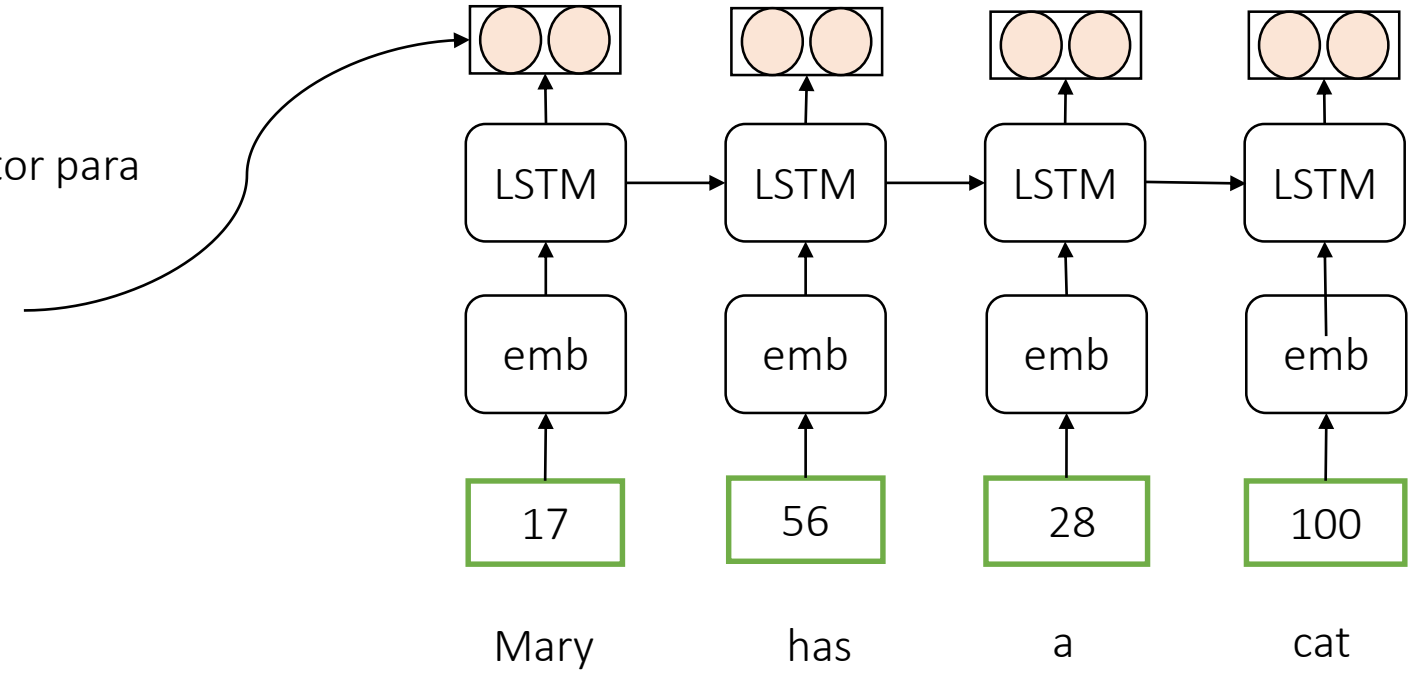
Permitirá obtener las etiquetas de salida para cada token de entrada de manera transparente.

tf.keras.layers.TimeDistributed

model = …

…
Añadir la capa de entrada
Añadir la capa de Embeddings
Añadir la capa de LSTMs
…

La salida de la LSTM debe ser un vector para cada palabra (chequear el parámetro return_sequences de la capa LSTM).



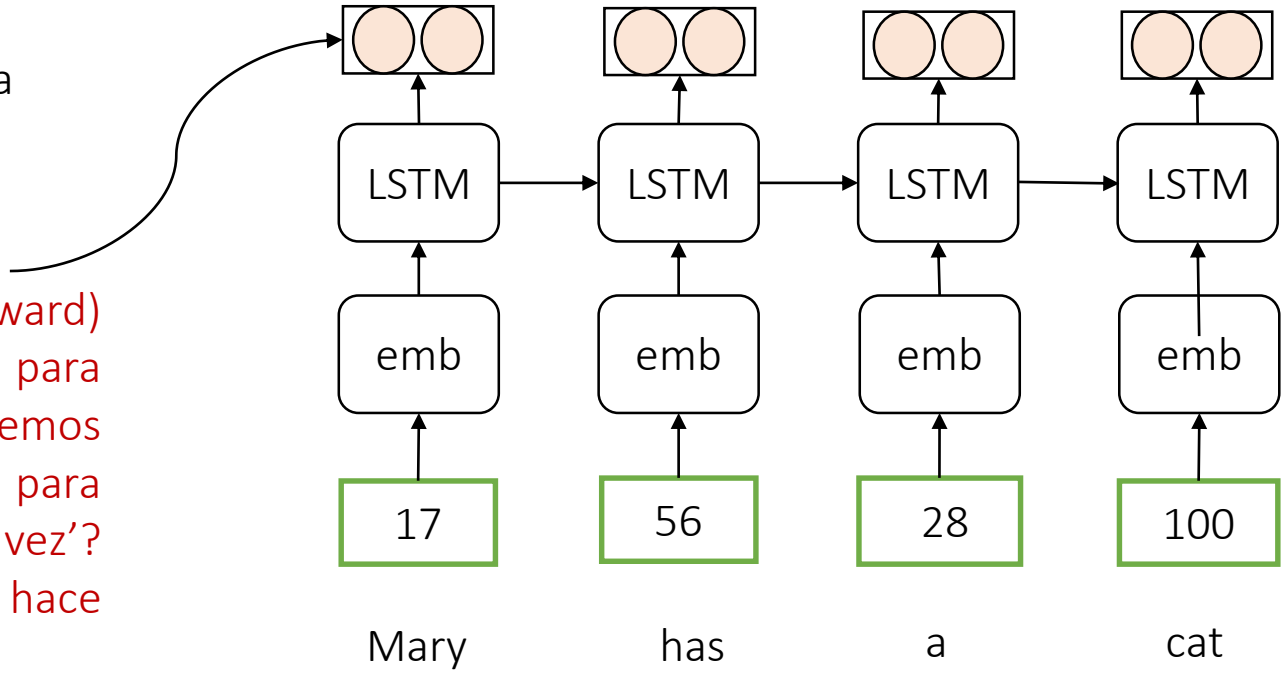
tf.keras.layers.TimeDistributed

model = …

…
Añadir la capa de entrada
Añadir la capa de Embeddings
Añadir la capa de LSTMs
…

La salida de la LSTM debe ser un vector para cada palabra (chequear el parámetro return_sequences de la capa LSTM).

Sin embargo, la capa Dense (redes feed-forward) que viene a continuación no está pensada para ser aplicada sobre secuencias, ¿cómo podemos obtener entonces fácilmente la etiqueta para todos los elementos de entrada 'de una vez'? Usando el wrapper TimeDistributed, que hace justamente eso.



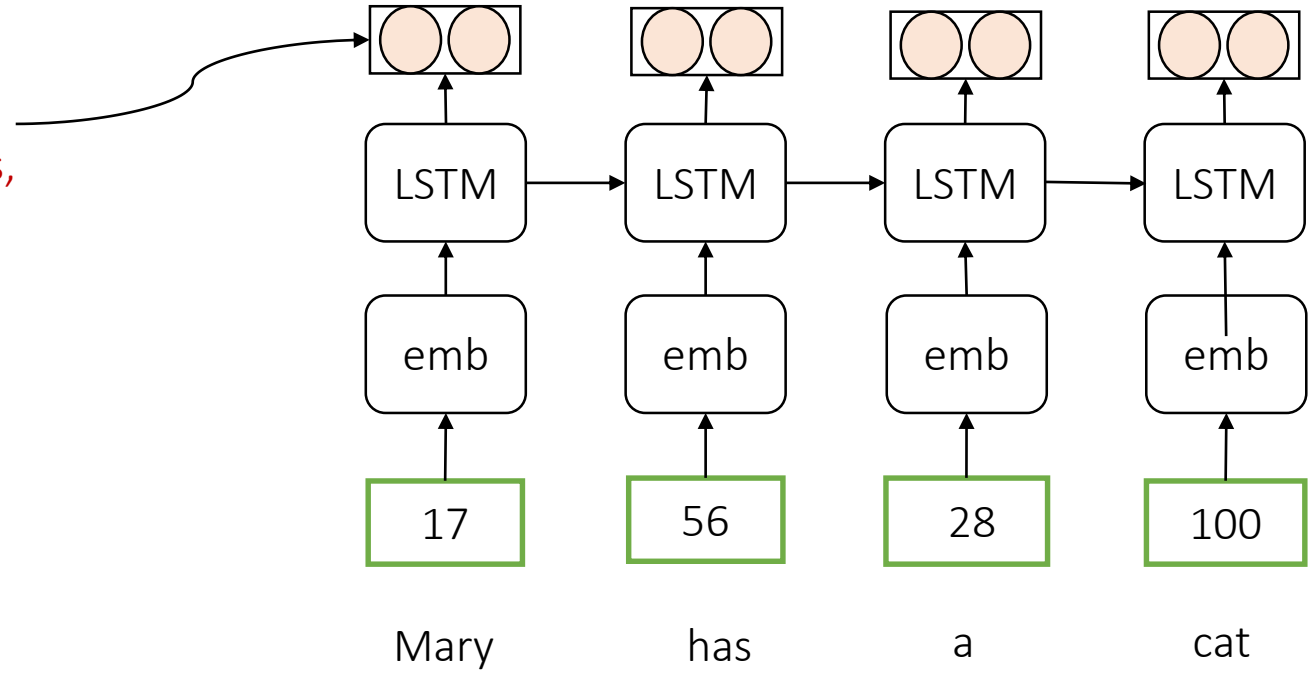
tf.keras.layers.TimeDistributed

model = …

…
Añadir la capa de entrada
Añadir la capa de Embeddings
Añadir la capa de LSTMs
…

model.add(TimeDistributed(Dense(nlabels, activation='softmax’)))

…
model.fit(…)



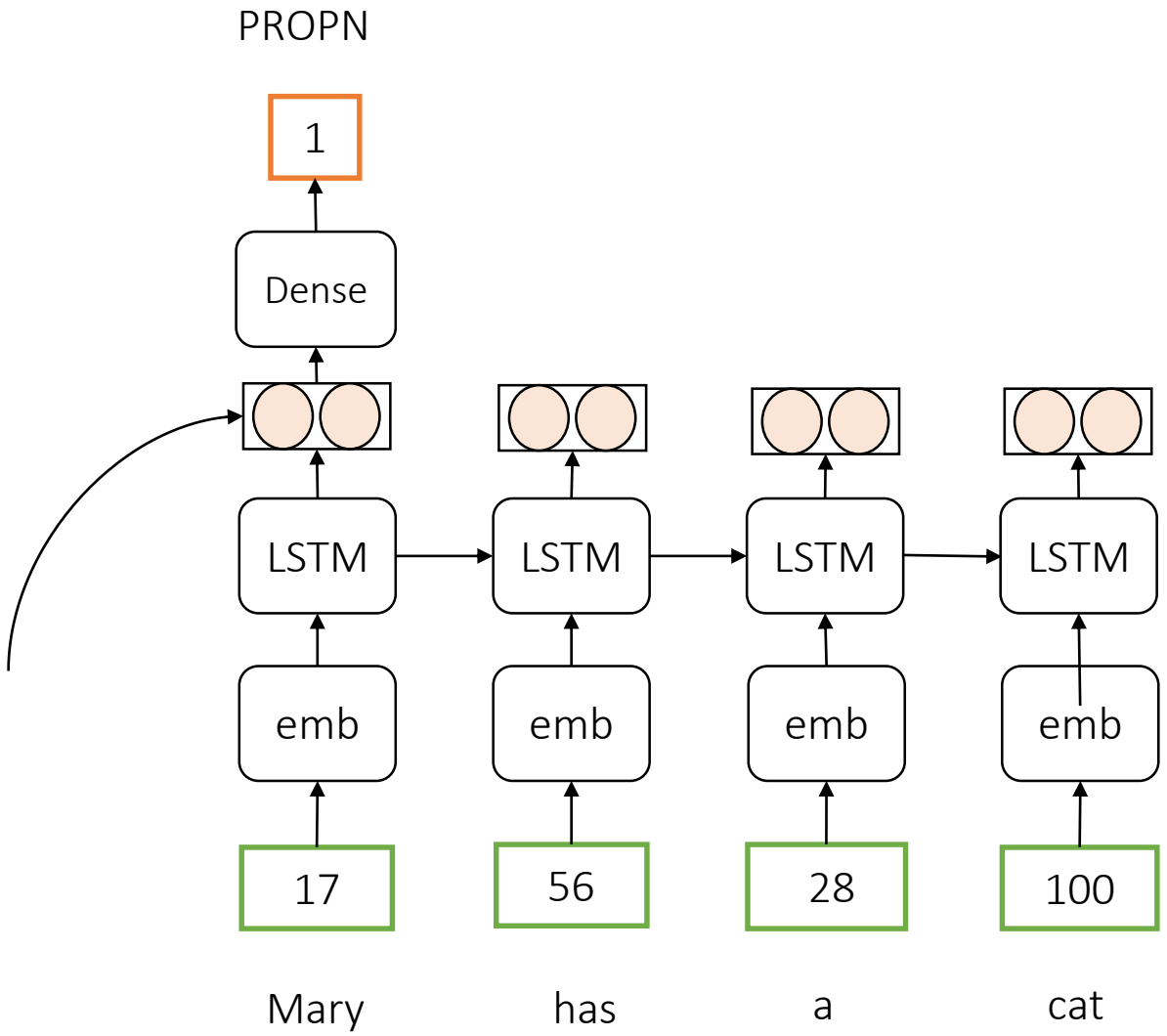
tf.keras.layers.TimeDistributed

model = …

…
Añadir la capa de entrada
Añadir la capa de Embeddings
Añadir la capa de LSTMs
…

model.add(TimeDistributed(Dense(nlabels, activation='softmax')))

…
model.fit(…)



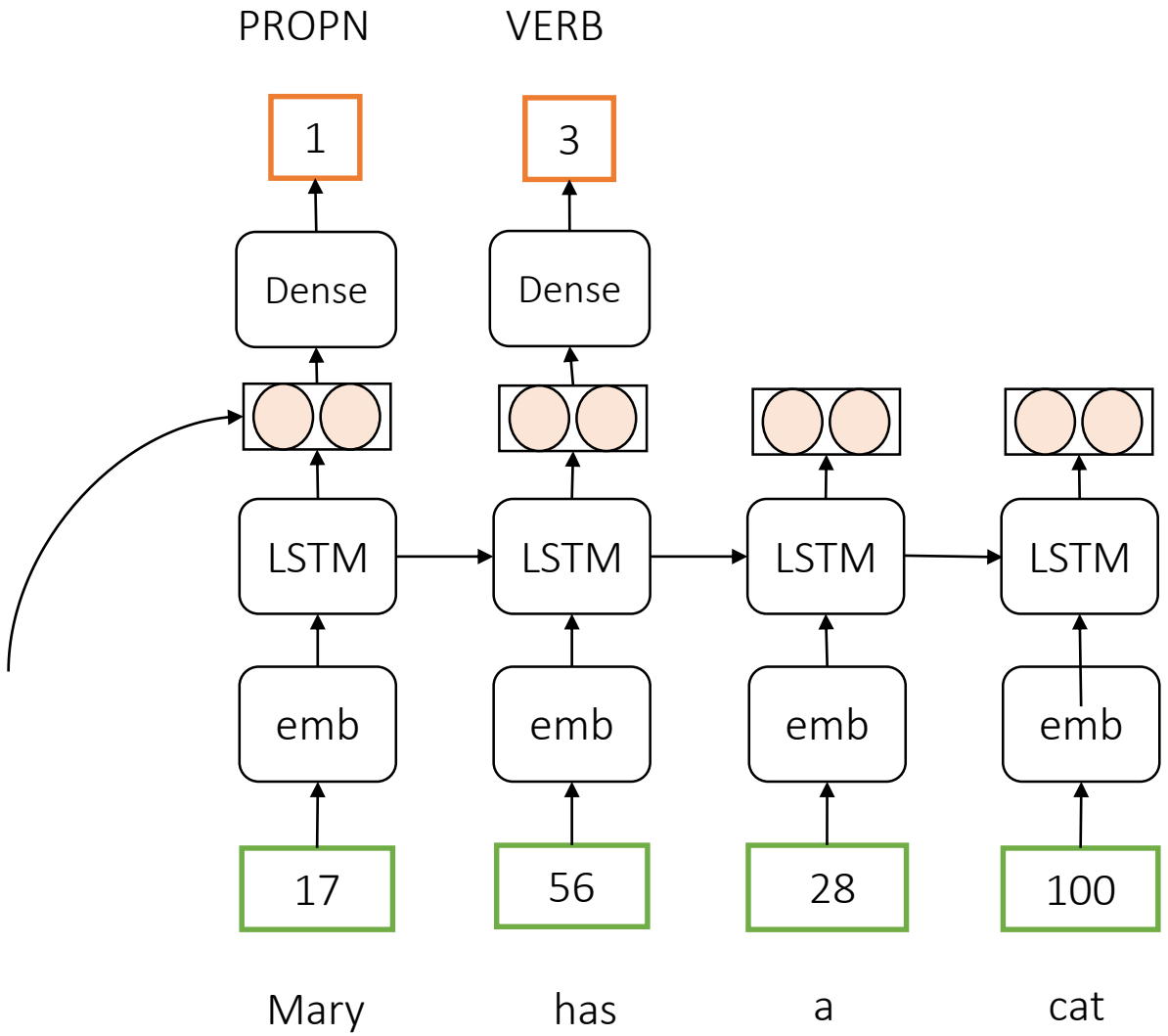
tf.keras.layers.TimeDistributed

model = …

…
Añadir la capa de entrada
Añadir la capa de Embeddings
Añadir la capa de LSTMs
…

model.add(TimeDistributed(Dense(nlabels, activation='softmax')))

…
model.fit(…)



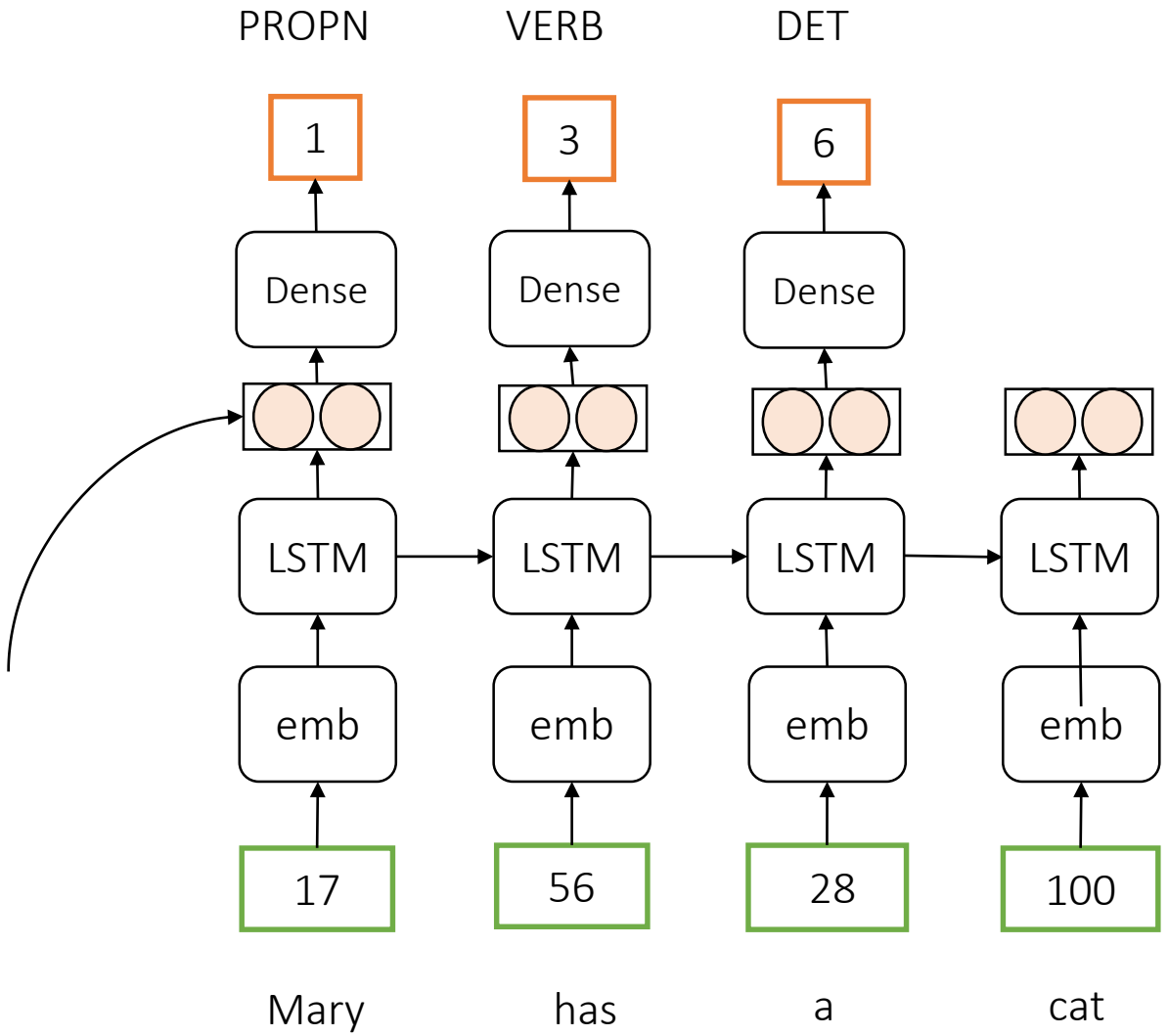
tf.keras.layers.TimeDistributed

model = …

…
Añadir la capa de entrada
Añadir la capa de Embeddings
Añadir la capa de LSTMs
…

model.add(TimeDistributed(Dense(nlabels, activation='softmax’)))

…
model.fit(…)



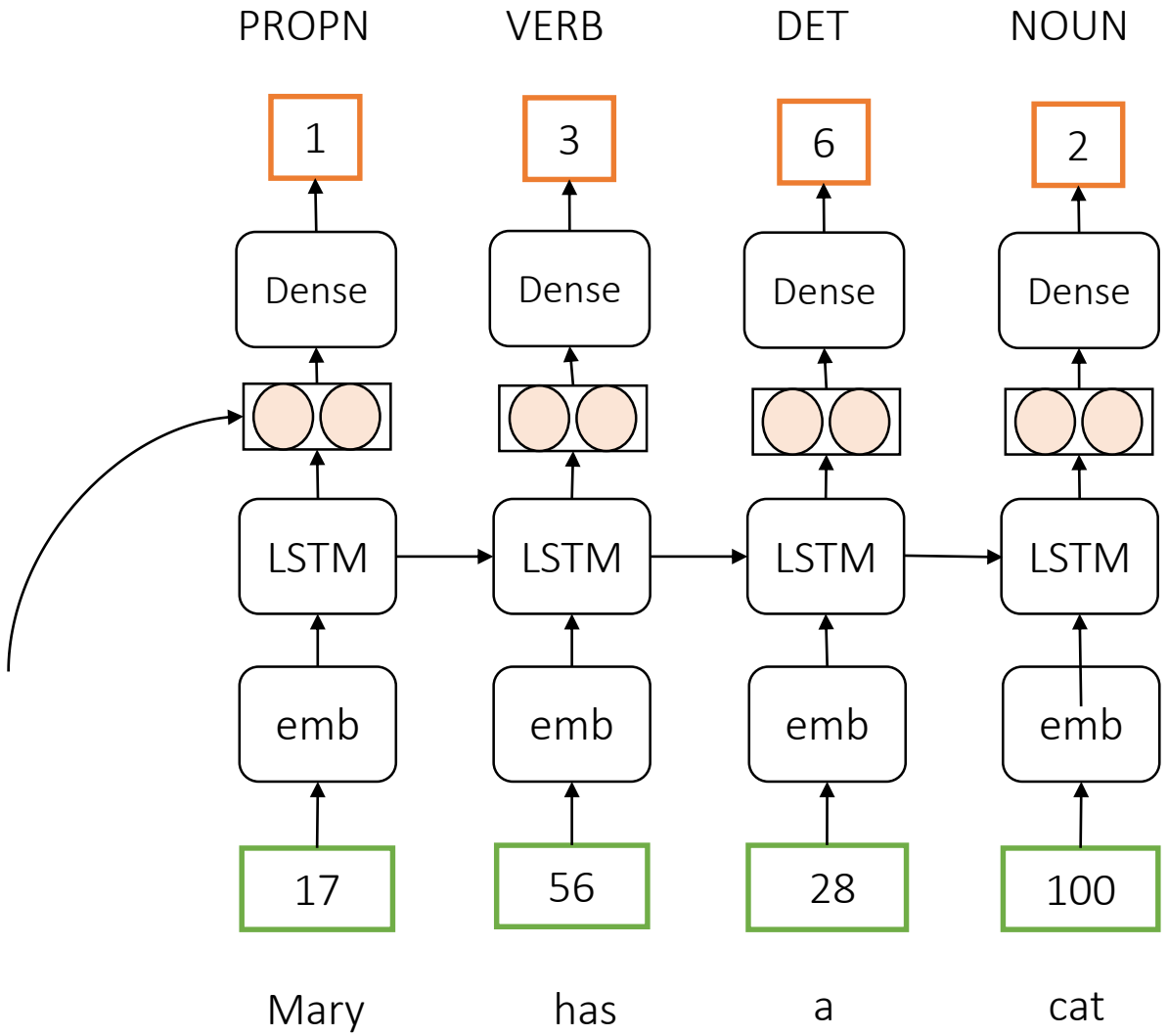
tf.keras.layers.TimeDistributed

model = …

…
Añadir la capa de entrada
Añadir la capa de Embeddings
Añadir la capa de LSTMs
…

model.add(TimeDistributed(Dense(nlabels, activation='softmax')))

…
model.fit(…)



Parámetros de entrenamiento

Es un problema de clasificación multi-clase.

Optimizaremos el modelo con categorical_crossentropy loss: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/categorical_crossentropy

Alternativamente a categorical_crossentropy loss también podemos usar una sparse categorical crossentropy loss (en ese caso no necesitaremos la función to_categorical): https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

Como optimizador, podemos usar SGD, Adam, etc.