P1: Part-of-Speech (PoS) Tagging

Natural Language Understanding

Interuniversity Master's Degree in Artificial Intelligence

Academic Year 2025-2026

UNIVERSIDADE DA CORUÑA

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Universidade de Vigo

Part-of-speech tagging

Tagging each word with its grammatical category (noun, verb, adjective, etc.).

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

This is the simplest approach to structured prediction, that is, to convert unstructured text into a structured representation that can be used to automatically extract information.

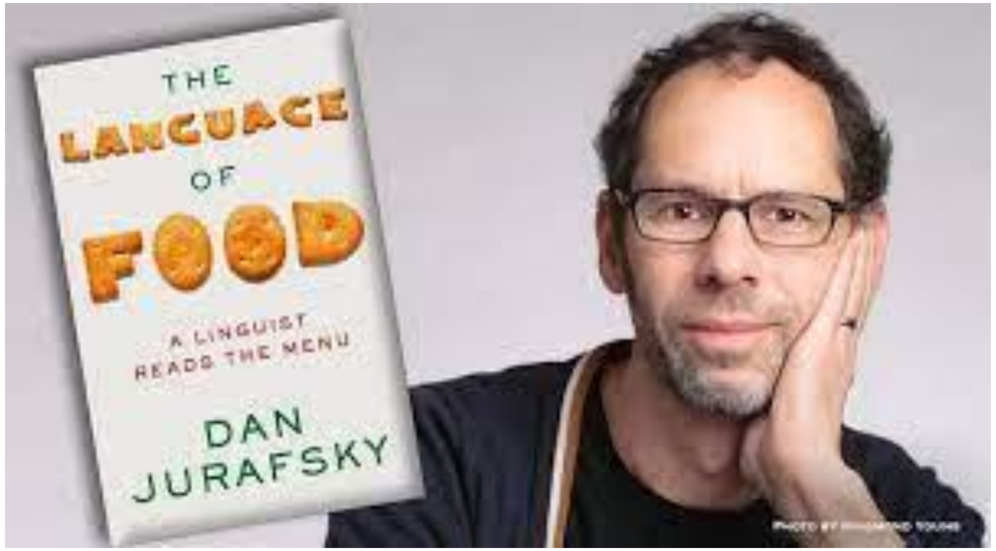
Part-of-speech tagging

```
First   ADV
come    VERB
,   PUNCT
first   ADV
serve   VERB
.   PUNCT
```

The idea is to preprocess the text with this information, obtaining useful input data for higher-level NLP tasks:

- Machine translation...
- Question answering...
- It is also valuable for interdisciplinary studies between computer science and linguistics, providing insights into how humans express themselves differently, for instance, based on the domain or the audience they are addressing.

Part-of-speech tagging



Why do we eat toast for breakfast, and then toast to good health at dinner?

What does the turkey we eat on Thanksgiving have to do with the country on the eastern Mediterranean?

Can you figure out how much your dinner will cost by counting the words on the menu?

In *The Language of Food*, Stanford University professor and MacArthur Fellow Dan Jurafsky peels away the mysteries from the foods we think we know.

Objectives

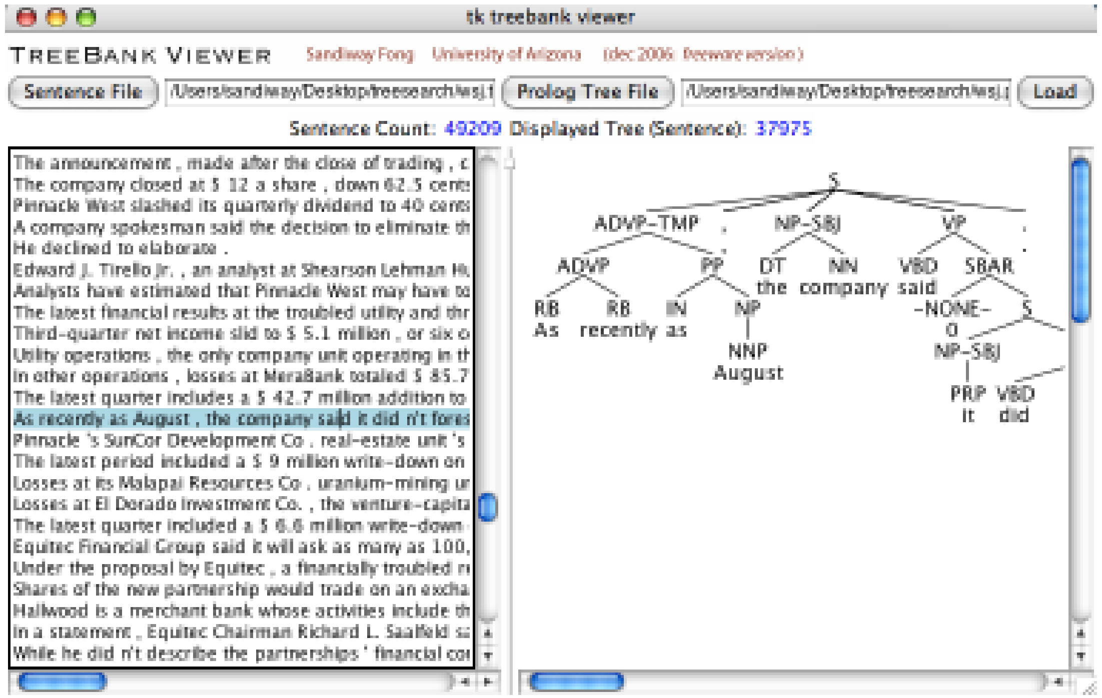
Work and process successfully Universal Dependency (UD) datasets

Build and train your own Part-of-Speech (PoS) taggers based on neural models

Compare neural architectures and their performance across natural languages

Treebanks

- In linguistics, a treebank is a parsed text corpus that annotates syntactic or semantic sentence structure



Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330 (https://aclanthology.org/J93-2004)

Universal Dependencies (UD)

- Universal dependencies (UD) is a framework for morphosyntactic annotation of human language
  - used to create treebanks for more than 100 languages
    - facilitates multilingual parser development
    - cross-lingual learning
    - and parsing research from a language typology perspective
- Find out more at: https://universaldependencies.org/

Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, Daniel Zeman; Universal Dependencies. *Computational Linguistics* 2021; 47 (2): 255–308. doi: https://doi.org/10.1162/coli_a_00402

UD CoNLL-U Format

- Annotations are encoded in plain text files (UTF-8) with three types of lines:
  - Word lines containing the annotation of a word/token in 10 fields separated by single tab characters; see below.
  - Blank lines marking sentence boundaries.
  - Comment lines starting with hash (#).

- More at: https://universaldependencies.org/format.html

Sabine Buchholz and Erwin Marsi. 2006. [CoNLL-X Shared Task on Multilingual Dependency Parsing](https://aclanthology.org/W06-2920). In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City. Association for Computational Linguistics. https://aclanthology.org/W06-2920

Word lines in CoNLL-U Format

- Sentences consist of one or more word lines, and word lines contain the following fields:
  - ID: Word index, integer starting at 1 for each new sentence; may be a range for multiword tokens; may be a decimal number for empty nodes (decimal numbers can be lower than 1 but must be greater than 0). We will ignore multiword and empty tokens!
  - FORM: Word form or punctuation symbol.
  - LEMMA: Lemma or stem of word form.
  - UPOS: Universal part-of-speech tag.
  - XPOS: Language-specific part-of-speech tag; underscore if not available.
  - FEATS: List of morphological features from the universal feature inventory or from a defined language-specific extension; underscore if not available.
  - HEAD: Head of the current word, which is either a value of ID or zero (0).
  - DEPREL: Universal dependency relation to the HEAD (root iff HEAD = 0) or a defined language-specific subtype of one.
  - DEPS: Enhanced dependency graph in the form of a list of head-deprel pairs.
  - MISC: Any other annotation.

Example

Open

en_partut-ud-train.conllu
~/Downloads

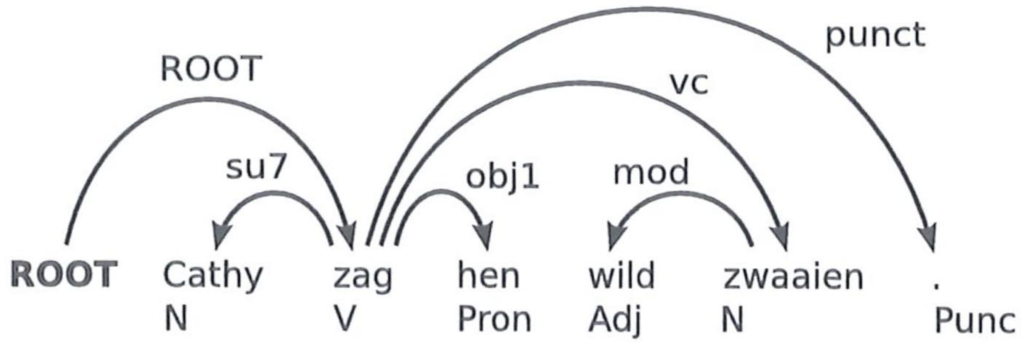
```
# sent_id = en_partut-ud-3
# text = Distribution of this license does not create an attorney-client relationship.
1	Distribution	distribution	NOUN	S	Number=Sing	7	nsubj	_	_
2	of	of	ADP	E	_	4	case	_	_
3	this	this	DET	DD	Number=Sing|PronType=Dem	4	det	_	_
4	license	license	NOUN	S	Number=Sing	1	nmod	_	_
5	does	do	AUX	VM	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	7	aux	_	_
6	not	not	PART	PART	Polarity=Neg	7	advmod	_	_
7	create	create	VERB	V	Mood=Ind|Number=Plur|Tense=Pres|VerbForm=Fin	0	root	_	_
8	an	a	DET	RI	Definite=Ind|Number=Sing|PronType=Art	12	det	_	_
9	attorney	attorney	NOUN	S	Number=Sing	12	nmod	_	SpaceAfter=No
10	-	-	PUNCT	FF	_	9	punct	_	SpaceAfter=No
11	client	client	NOUN	S	Number=Sing	9	compound	_	_
12	relationship	relationship	NOUN	S	Number=Sing	7	obj	_	SpaceAfter=No
13	.	.	PUNCT	FS	_	7	punct	_	_

# sent_id = en_partut-ud-4
# text = Creative Commons provides this information on an "as-is" basis.
1	Creative	Creative	PROPN	SP	_	3	nsubj	_	_
2	Commons	Commons	PROPN	SP	_	1	flat	_	_
3	provides	provide	VERB	V	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	0	root	_	_
4	this	this	DET	DD	Number=Sing|PronType=Dem	5	det	_	_
5	information	information	NOUN	S	Number=Sing	3	obj	_	_
6	on	on	ADP	E	_	13	case	_	_
7	an	a	DET	RI	Definite=Ind|Number=Sing|PronType=Art	13	det	_	_
8	"	"	PUNCT	FB	_	11	punct	_	SpaceAfter=No
9	as	as	ADP	E	_	11	mark	_	SpaceAfter=No
10	-	-	PUNCT	FF	_	9	punct	_	SpaceAfter=No
11	is	be	VERB	V	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	13	amod	_	SpaceAfter=No
12	"	"	PUNCT	FB	_	11	punct	_	_
13	basis	basis	NOUN	S	Gender=Masc|Number=Sing	3	obl	_	SpaceAfter=No
14	.	.	PUNCT	FS	_	3	punct	_	_

# sent_id = en_partut-ud-5
# text = Creative Commons makes no warranties regarding the information provided, and disclaims liability for damages resulting from its use.
1	Creative	Creative	PROPN	SP	_	3	nsubj	_	_
2	Commons	Commons	PROPN	SP	_	1	flat	_	_
3	makes	make	VERB	V	Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin	0	root	_	_
4	no	no	DET	DI	PronType=Ind	5	det	_	_
5	warranties	warranty	NOUN	S	Number=Plur	3	obj	_	_
6	regarding	regard	VERB	V	Number=Sing|Tense=Pres|VerbForm=Part	5	acl	_	_
7	the	the	DET	RD	Definite=Def|PronType=Art	8	det	_	_
8	information	information	NOUN	S	Number=Sing	6	obj	_	_
9	provided	provide	VERB	V	Tense=Past|VerbForm=Part	8	acl	_	SpaceAfter=No
10	,	,	PUNCT	FF	_	12	punct	_	_
11	and	and	CCONJ	CC	_	12	cc	_	_
```

UPOS

Universal part-of-speech tags (UPOS). Typos and abbreviations are given the category of the unabbreviated or correct word.

Traditional POS	UPOS	Category
noun	NOUN	common noun
	PROPN	proper noun
verb	VERB	main verb
	AUX	auxiliary verb or other tense, aspect, or mood particle
adjective	ADJ	adjective
	DET	determiner (including article)
	NUM	numeral (cardinal)
adverb	ADV	adverb
pronoun	PRON	pronoun
preposition	ADP	adposition (preposition/postposition)
conjunction	CCONJ	coordinating conjunction
	SCONJ	subordinating conjunction
interjection	INTJ	interjection
–	PART	particle (special single word markers in some languages)
–	X	other (e.g., words in foreign language expressions)
–	SYM	non-punctuation symbol (e.g., a hash (#) or emoji)
–	PUNCT	punctuation

Example of a tree for UD in CoNLL-U format

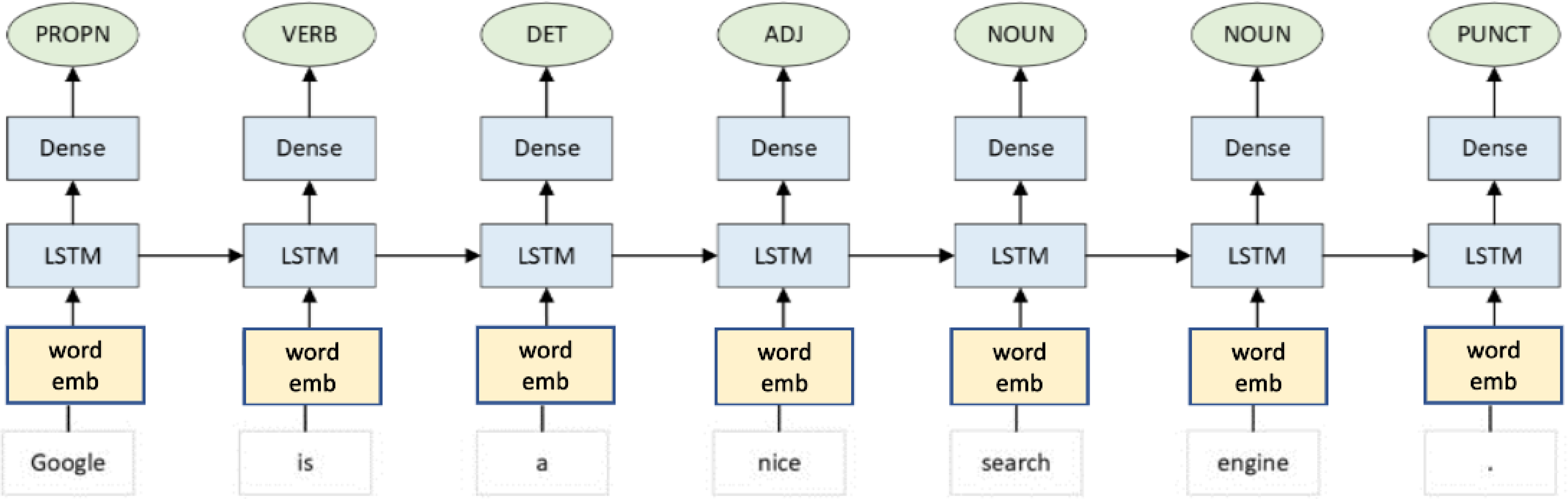


1	Cathy	Cathy	N	N	eigen|ev|neut	2	su7
2	zag	zie	V	V	trans|ovt|1of2of3|ev	0	ROOT
3	hen	hen	Pron	Pron	per|3|mv|datofacc	2	obj1
4	wild	wild	Adj	Adj	attr|stell|onverv	5	mod
5	zwaaien	zwaai	N	N	soort|mv|neut	2	vc
6	.	.	Punc	Punc	punt	5	punct

UD Treebanks datasets

- English: https://github.com/UniversalDependencies/UD_English-EWT/tree/master
- Galician: https://github.com/UniversalDependencies/UD_Galician-TreeGal/tree/master
- And many more: https://universaldependencies.org/#download

Architecture

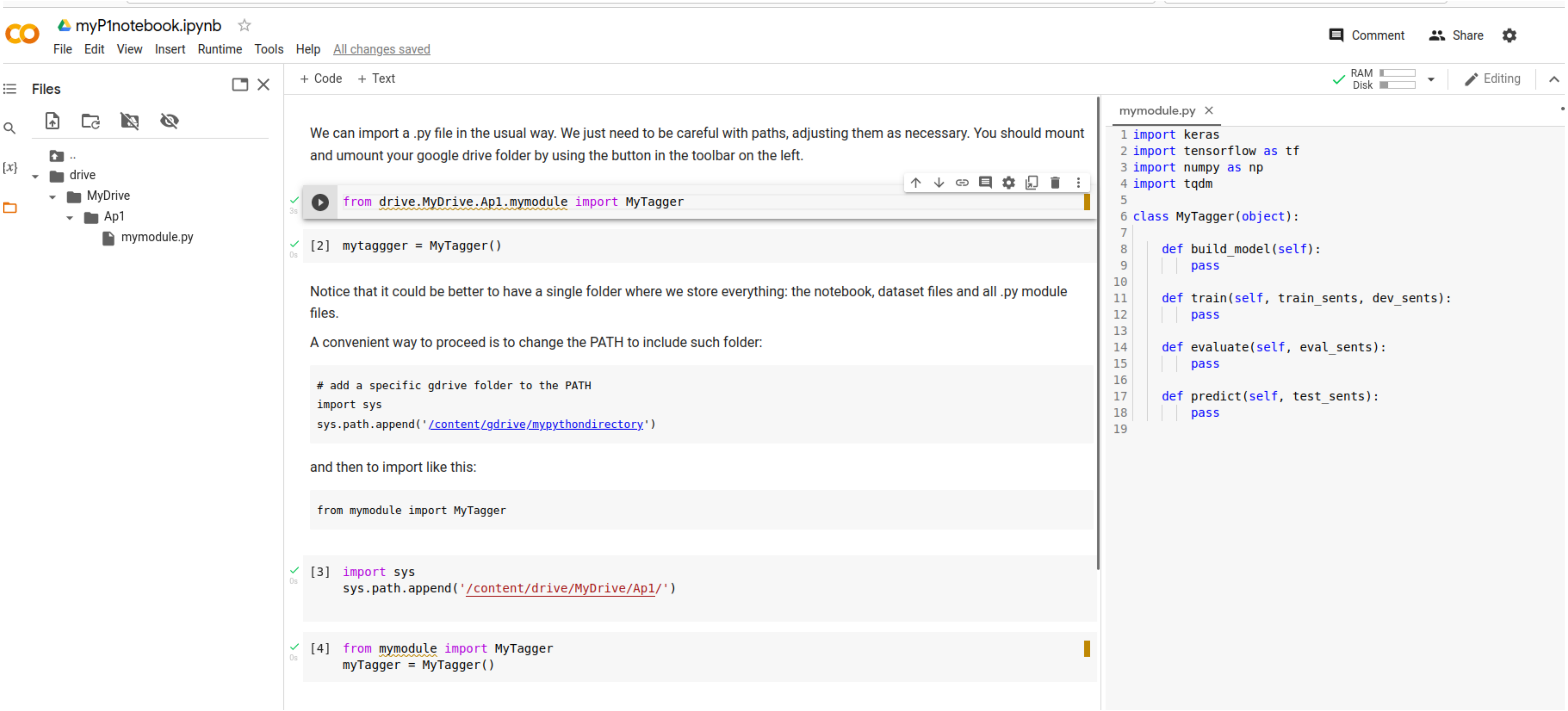


Tips and hints

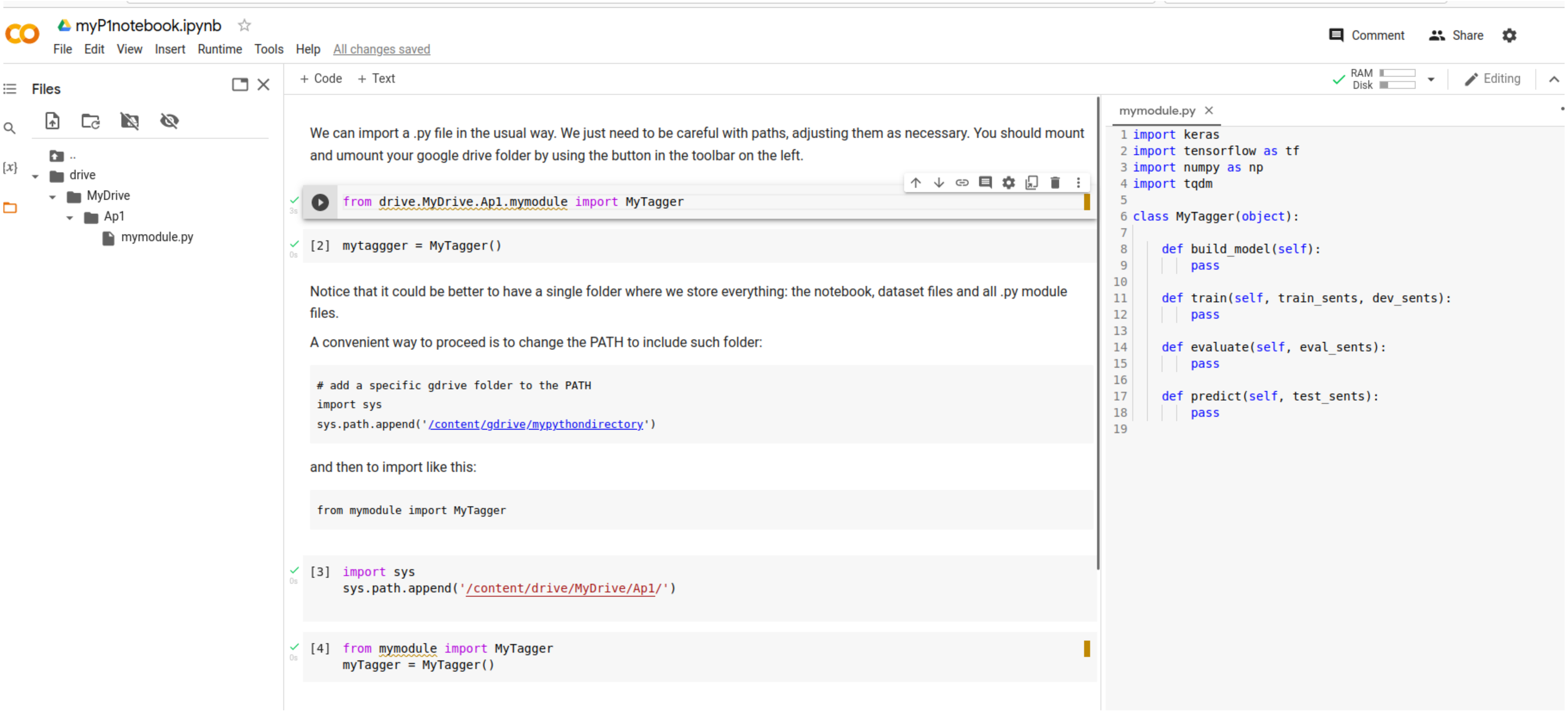
Organize your code

- Include a user manual explaining how to run
- Define classes to create reusable pieces of code
  - See our tutorial "Python Basics introduction and OOP" available in the Virtual Campus.
- Your main notebook should make easy to relaunch all the relevant tasks: training, evaluation and generation of labels, etc.
- You may have several .py files that can be imported in your main notebook (see next slide)
- Consider saving and loading your good trained models.
- Include a brief discussion of the implementation decisions, differences across the evaluated models that you might have explored, as well as an analysis of the performance across the evaluated datasets
  - In a separate PDF not exceeding 3 pages and written using Calibri font style and a size of minimum 11pt. Put your names on it.
  - You may include the necessary comments in python comments/jupyter notebook text cells

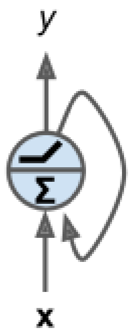
Example: using .py files in your notebook



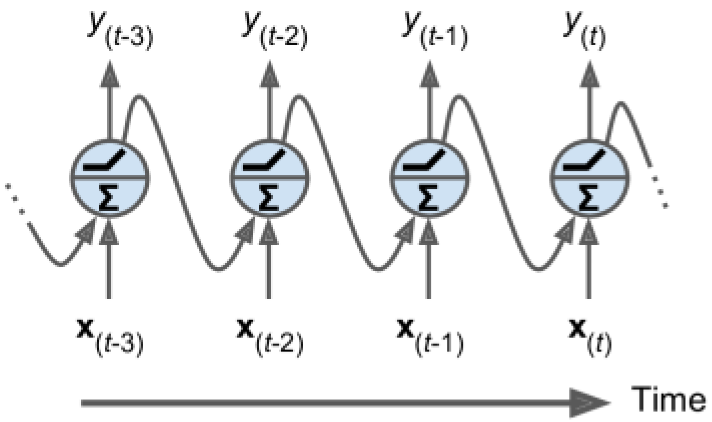
Example: using .py files in your notebook



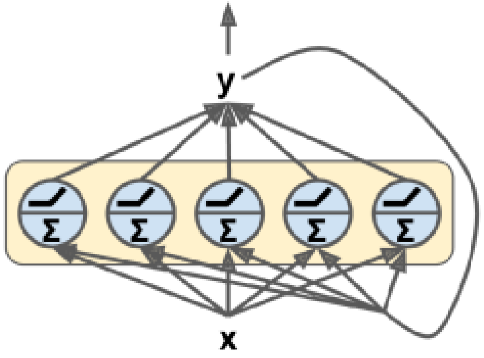
Recurrent neurons and layers



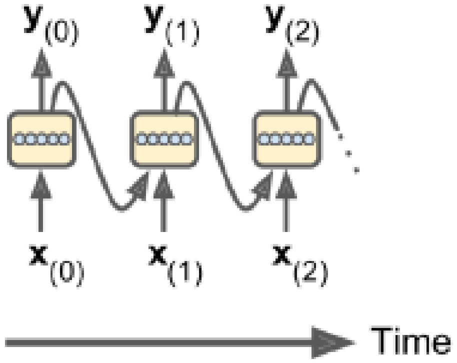
A recurrent neuron



Recurrent neuron unrolled through time



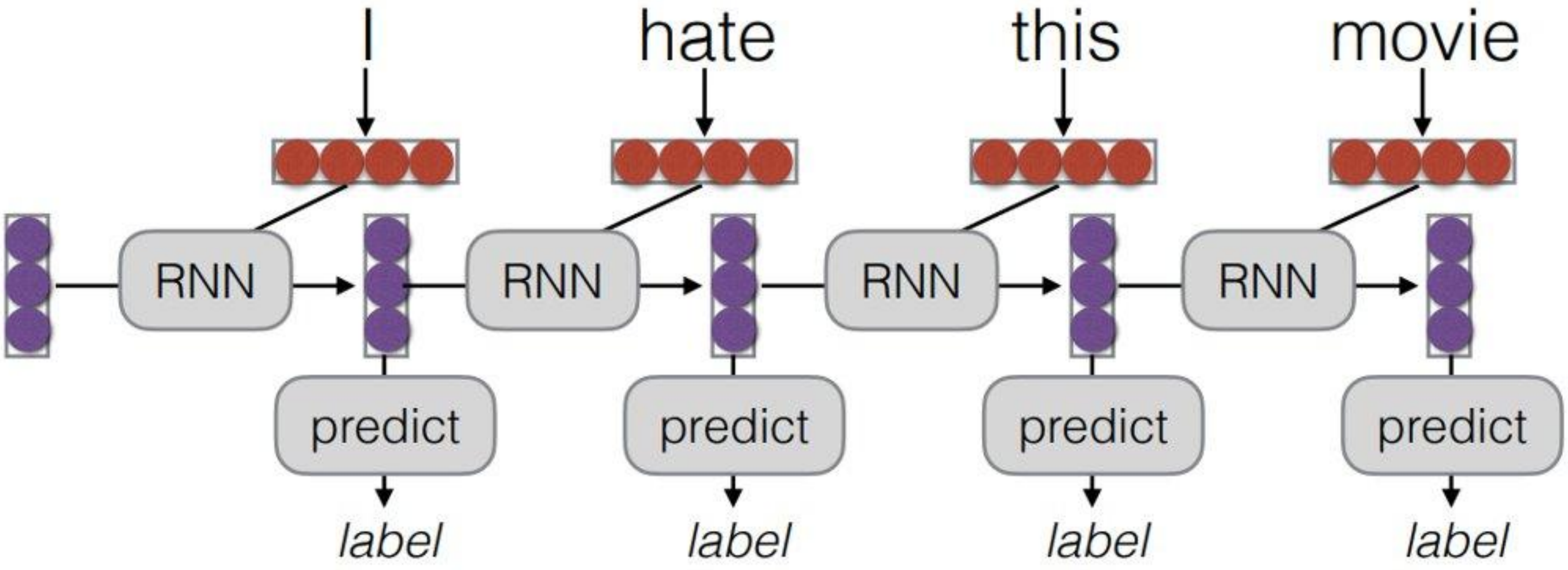
A layer of recurrent neurons



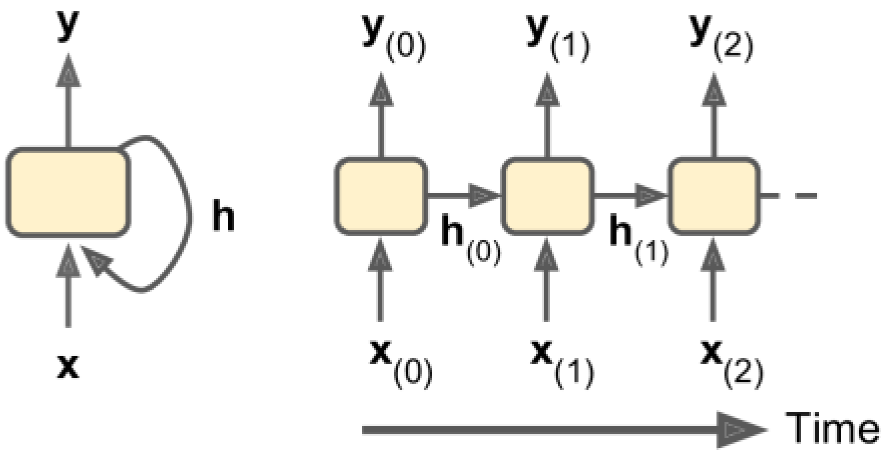
A layer of recurrent neurons unrolled through time

https://www.tensorflow.org/guide/keras/working_with_rnns

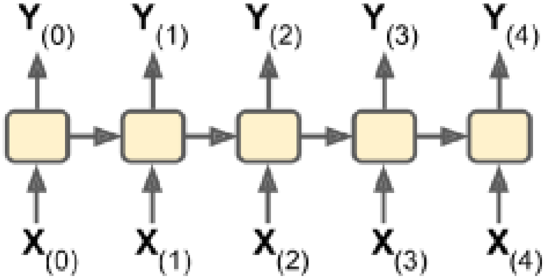
Recurrent neurons and layers



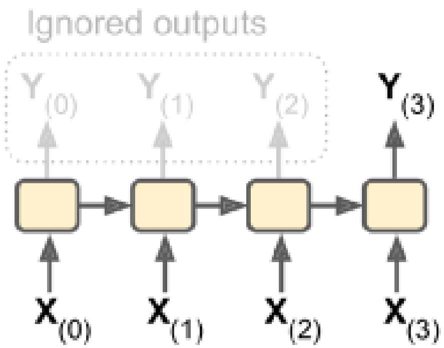
Like memory cells



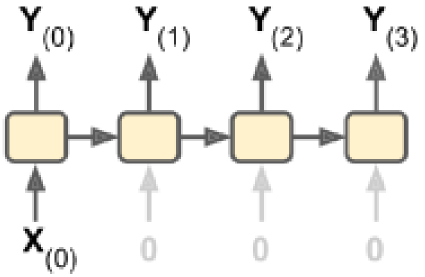
h: hidden state



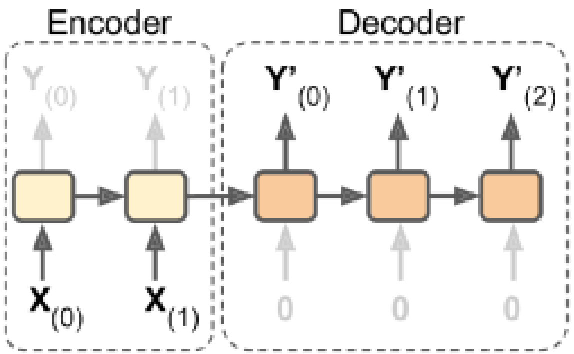
seq-to-seq



seq-to-vector

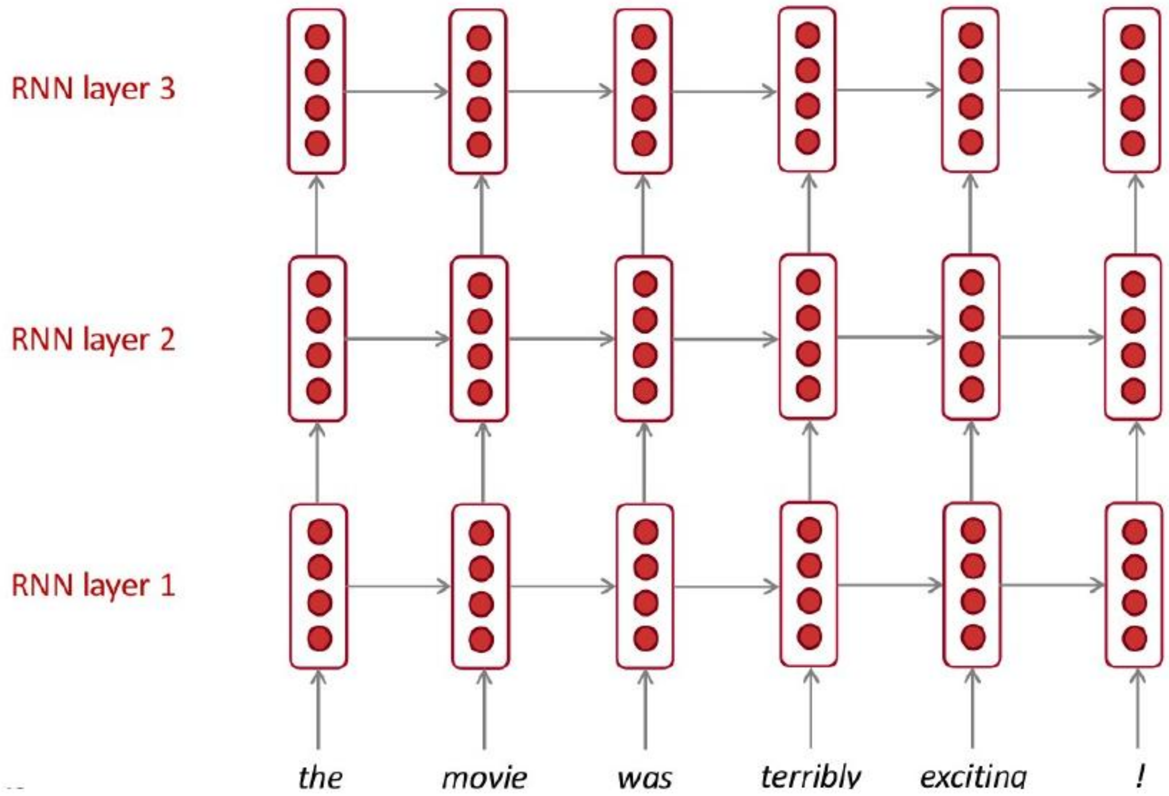


vector-to-seq



Endocer-Decoder networks

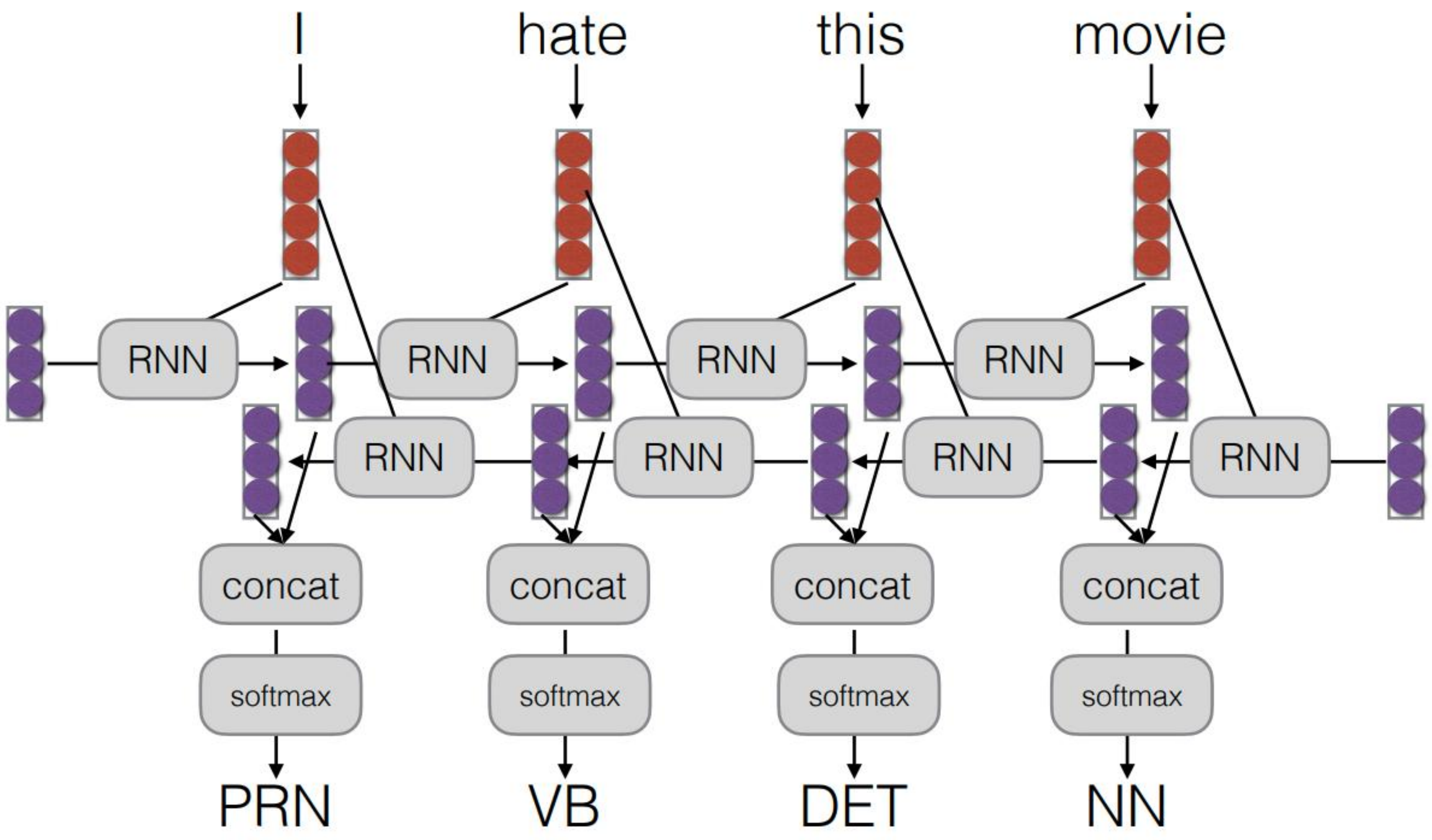
Deep RNNs




```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None,
1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])
```

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None,
1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)
])
```

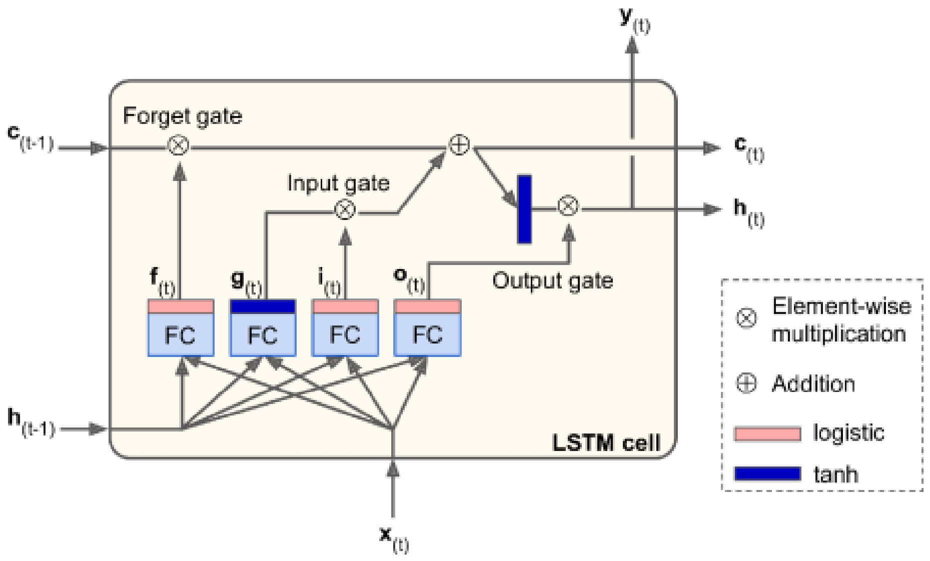
Bidirectional RNNs



LSTM cells

LSTM as a black box:

- used like any other basic memory cell
- much better performance and faster convergence
- detecting long-term dependencies in the data (successful at capturing long-term patterns)



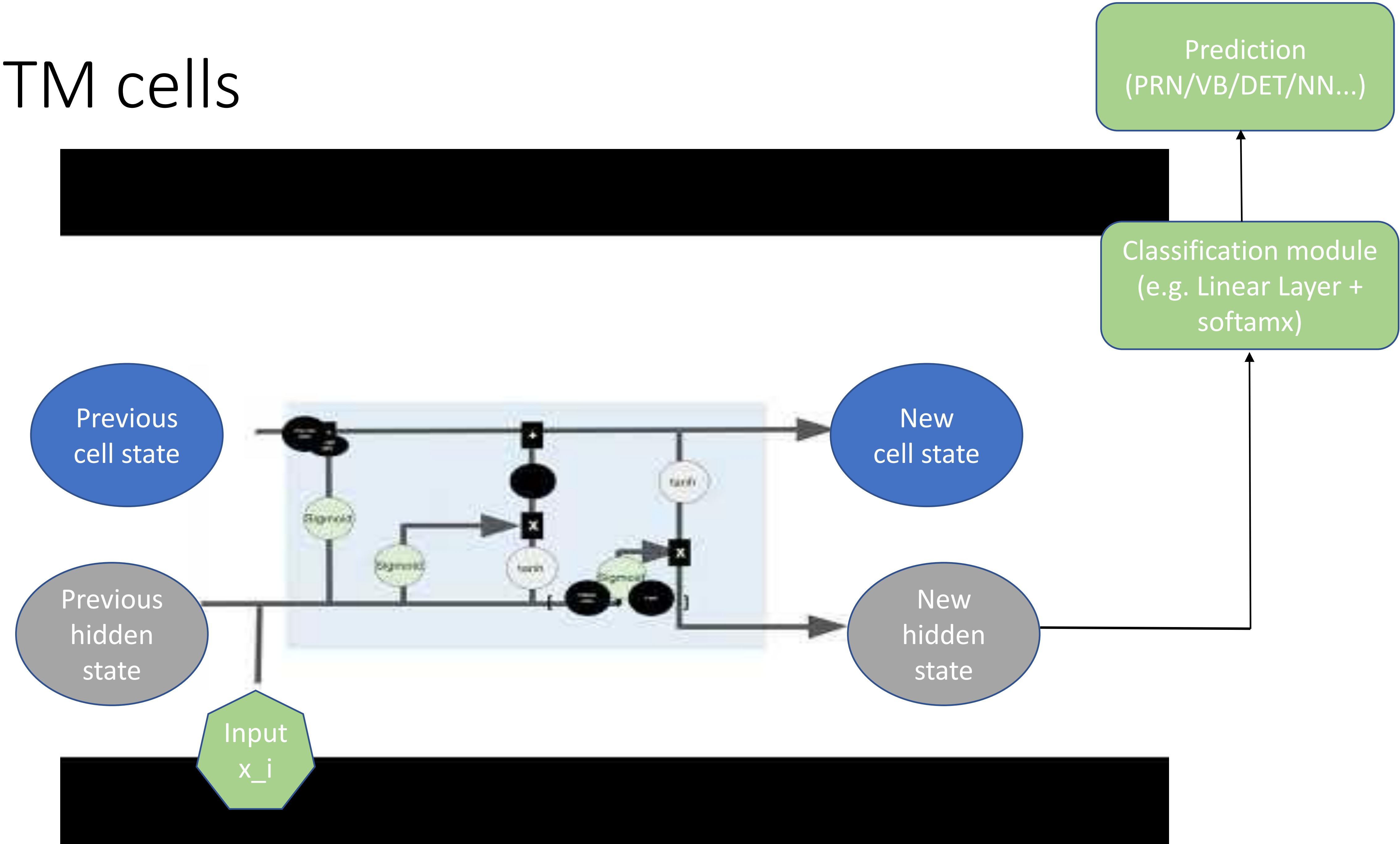

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None,
1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

```
model = keras.models.Sequential([
    keras.layers.LSTM(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.LSTM(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

https://keras.io/api/layers/recurrent_layers/lstm/

- https://keras.io/api/layers/recurrent_layers/time_distributed/

LSTM cells


Prediction (PRN/VB/DET/NN...)
Classification module (e.g. Linear Layer + softamx)
Previous cell state
New cell state
Previous hidden state
New hidden state
Input x_i

Defining the neural model in Keras

Creating the input samples

Preprocessing the treebank: Ignore sentences longer than 128 from the training, development and test sets. This means an input to the model will have the shape (max_sentence_length, embedding_size).

Recommendations:

1. Use the Tokenizer or TextVectorizer layer approaches.
2. Implement your models using the functional API.
3. Make sure and extra ID is used for unknown tokens (read the docs for the Tokenizer or the TextVectorizer layer)

Main steps to train and evaluate the word-level model:

1. **Transform the input sentences (strings) to a list of numerical IDs.**
2. **Pad the input sentences,** so they all have the same length (required for Keras models).
3. **Convert the output labels to IDs as well,** so we can train the model with the fit function.

Creating the input samples

texts_to_sequences (with our trained Tokenizer), TextVectorizer or an ad-hoc function

Mary has a cat → $\boxed{17\ \ 56\ \ 28\ \ 100}$

ad-hoc mapping to IDS

PROPN VERB DET NOUN → $\boxed{1\ \ 3\ \ 6\ \ 2}$

Spiderman exists $\boxed{7\ \ 324}$

PROPN VERB $\boxed{1\ \ 3}$

I am real $\boxed{98\ \ 76\ \ 3}$

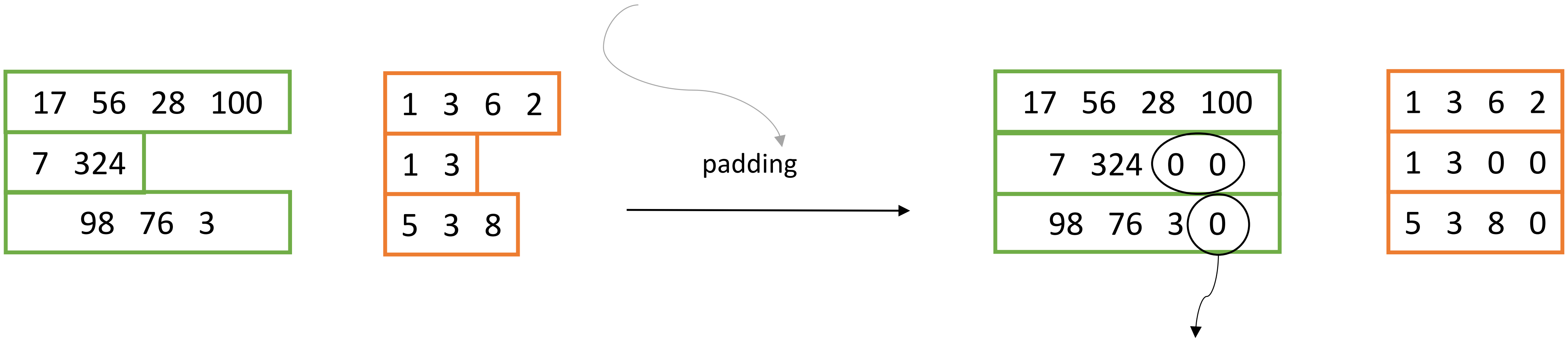
PRONOUN VERB ADJ $\boxed{5\ \ 3\ \ 8}$

Creating the input samples

In a single batch, we have input sentences with different lengths.

All sentences in a batch must have the same length in order to be fed to the keras models.

The solution is to apply padding to the inputs



We can ignore these from evaluation using mask_zero=True
when defining the Embedding layer

Creating the input samples

We are addressing a multi-label classification problem.

We will be optimizing our model with a categorical_crossentropy loss:
https://www.tensorflow.org/api_docs/python/tf/keras/metrics/categorical_crossentropy

To do so, we need to transform the desired output label (initially represented as a string) into a one-hot vector to encode which class the model must predict.

	tf.keras.utils.to_categorical	
1 3 6 2	→	[[0 1 0 0...] [0 0 0 1 0...] [0 0 0 0 0 0 1 0...] [0 0 1 0...] ...]
1 3 0 0	→	[[0 1 0 0...] [0 0 0 1 0...] [1 0...] [1 0...] ...]
5 3 8 0	→	[[0 0 0 0 0 1 0...] [0 0 0 1 0...] [0 0 0 0 0 0 0 0 1 0...] [1 0 0 0...] ...]

Alternatively to a categorical_crossentropy loss we also can use a sparse categorical crossentropy loss (in such case, we do not need to use the to_categorical function):
https://www.tensorflow.org/api_docs/python/tf/keras/losses/SparseCategoricalCrossentropy

Declaring the model – Relevant layers

Input layer: https://www.tensorflow.org/api_docs/python/tf/keras/Input [seen in P0]

Embedding layer: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding [seen in P0]

LSTM layer: https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM [seen in P0]

Bidirectional layer: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Bidirectional [optional]

Dense layer: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense [seen in P0]

TimeDistributed layer: https://www.tensorflow.org/api_docs/python/tf/keras/layers/TimeDistributed [not seen in P0, but relevant for P1]

tf.keras.layers.TimeDistributed

A wrapper to apply a layer to every temporal slice of an input.

The dimension of index one of the input is considered the temporal dimension.

In our case, the shape of the input for the word-level PoS tagger is (batch_size, max_sentence_length, word_embedding_size).

Given a TimeDistributed layer, it will apply the wrapped layer to every element of the sentence.

tf.keras.layers.TimeDistributed

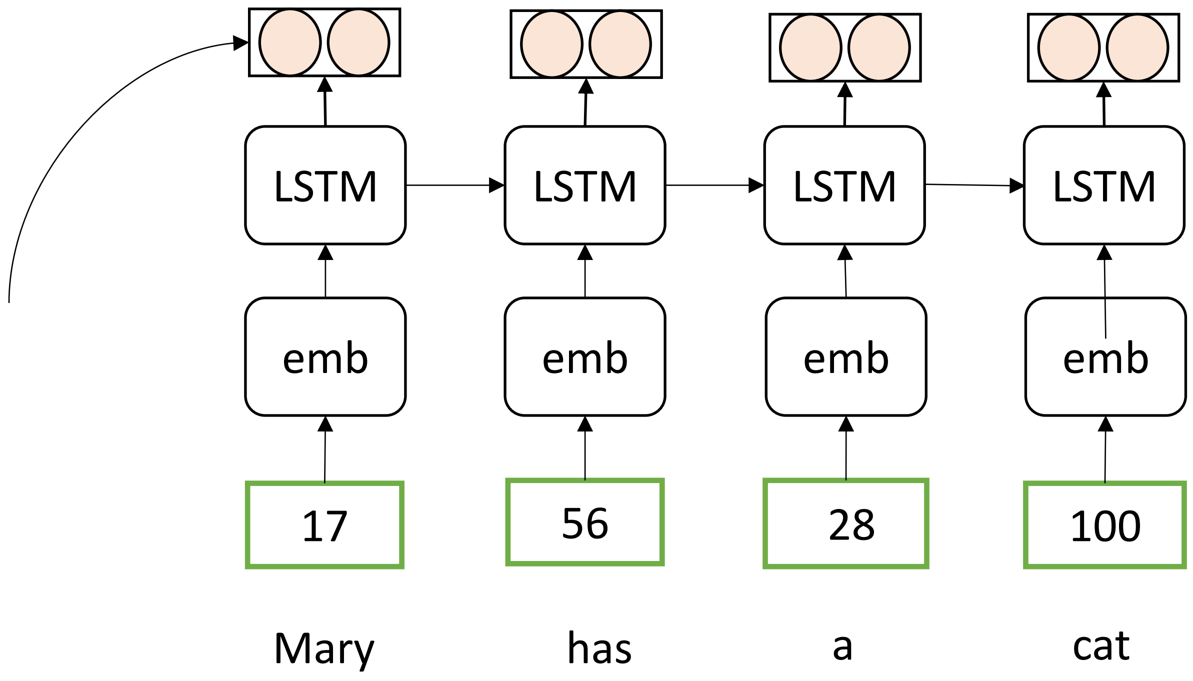
model = …

…

We add the embedding layer
We add the LSTM layer

…

The output of the LSTM must be a vector for each word.



tf.keras.layers.TimeDistributed

model = …

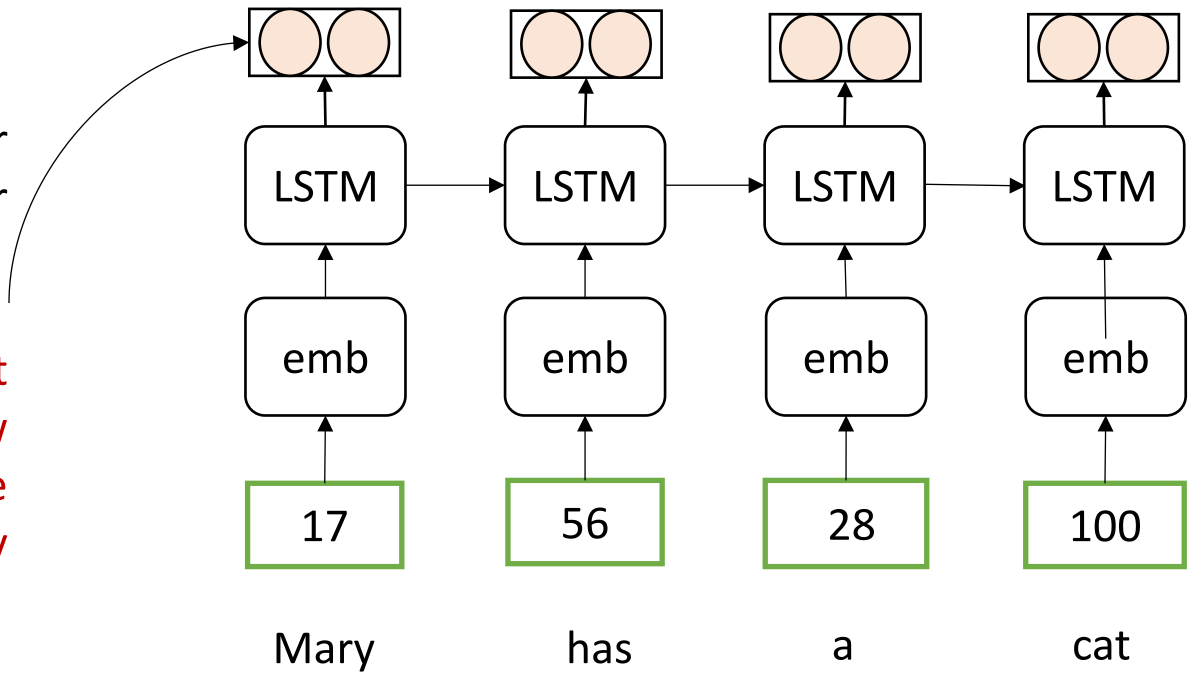
…

We add the embedding layer
We add the LSTM layer

…

The output of the LSTM must be a vector for each word, i.e. the hyperparameter return_sequences=True for the LSTM instance

However, the output layer (a Dense layer) is not intended to be applied over sequences, how can we deal with that? Using the TimeDistributed wrapper, which does exactly that.



tf.keras.layers.TimeDistributed

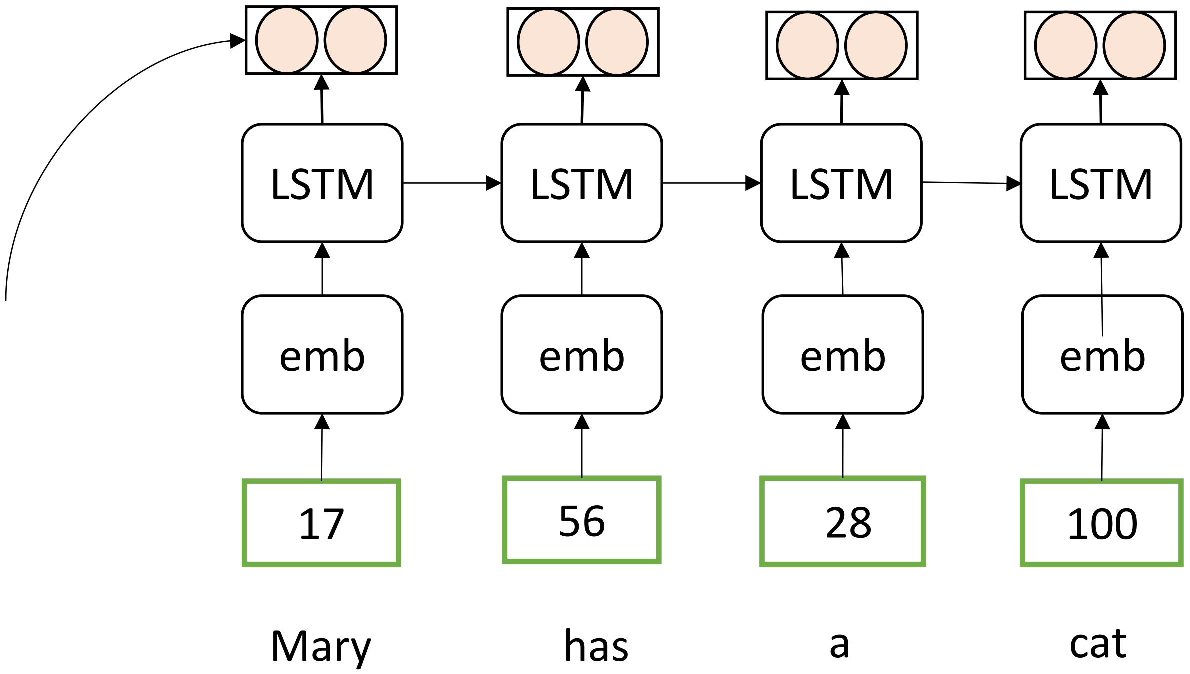
model = …

…

We add the embedding layer
We add the LSTM layer

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

The output of the LSTM must be a vector for each word.
…
model.fit(…)



tf.keras.layers.TimeDistributed

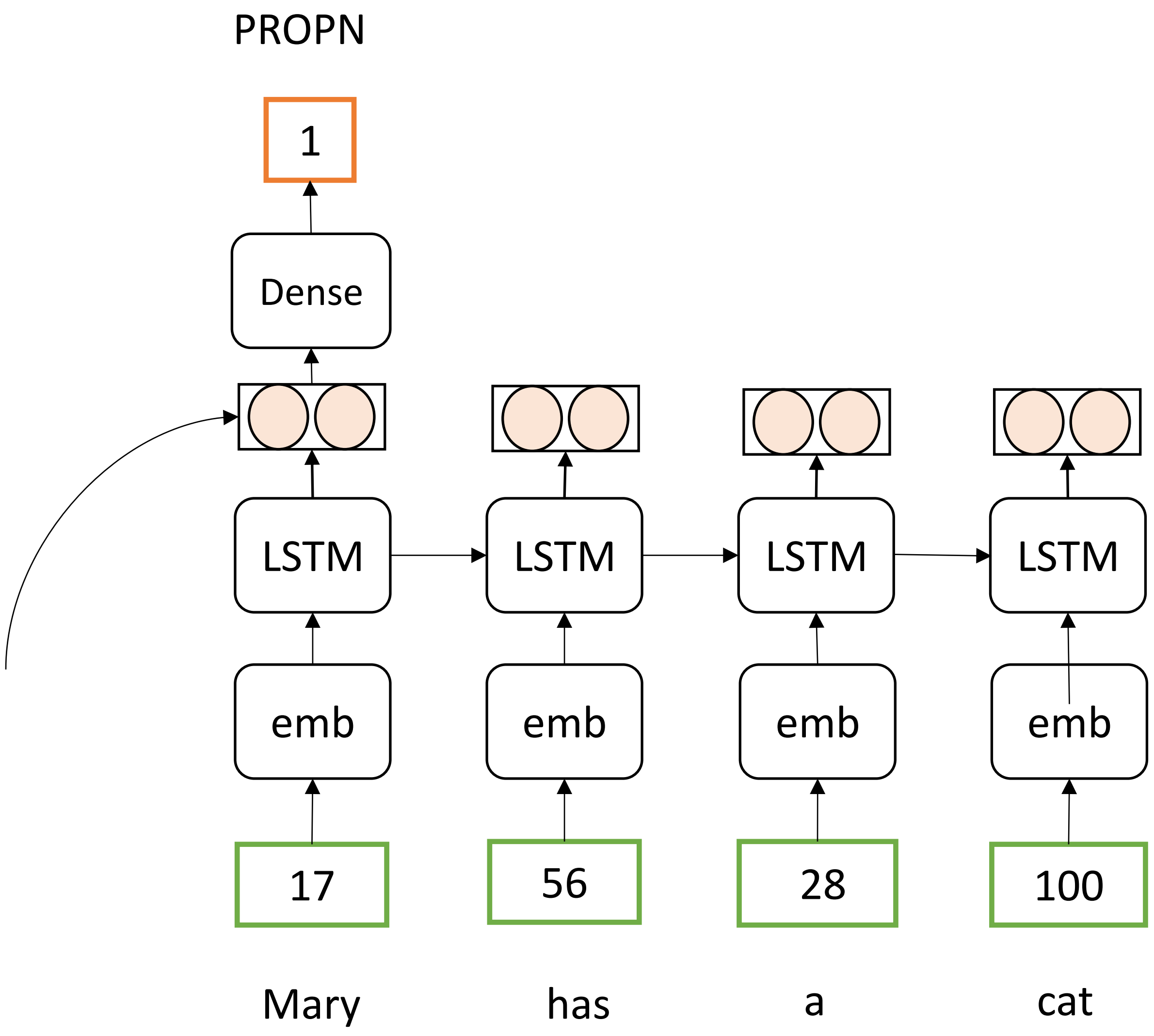
model = …

…

We add the embedding layer
We add the LSTM layer

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

The output of the LSTM must be a vector for each word.
…
model.fit(…)



tf.keras.layers.TimeDistributed

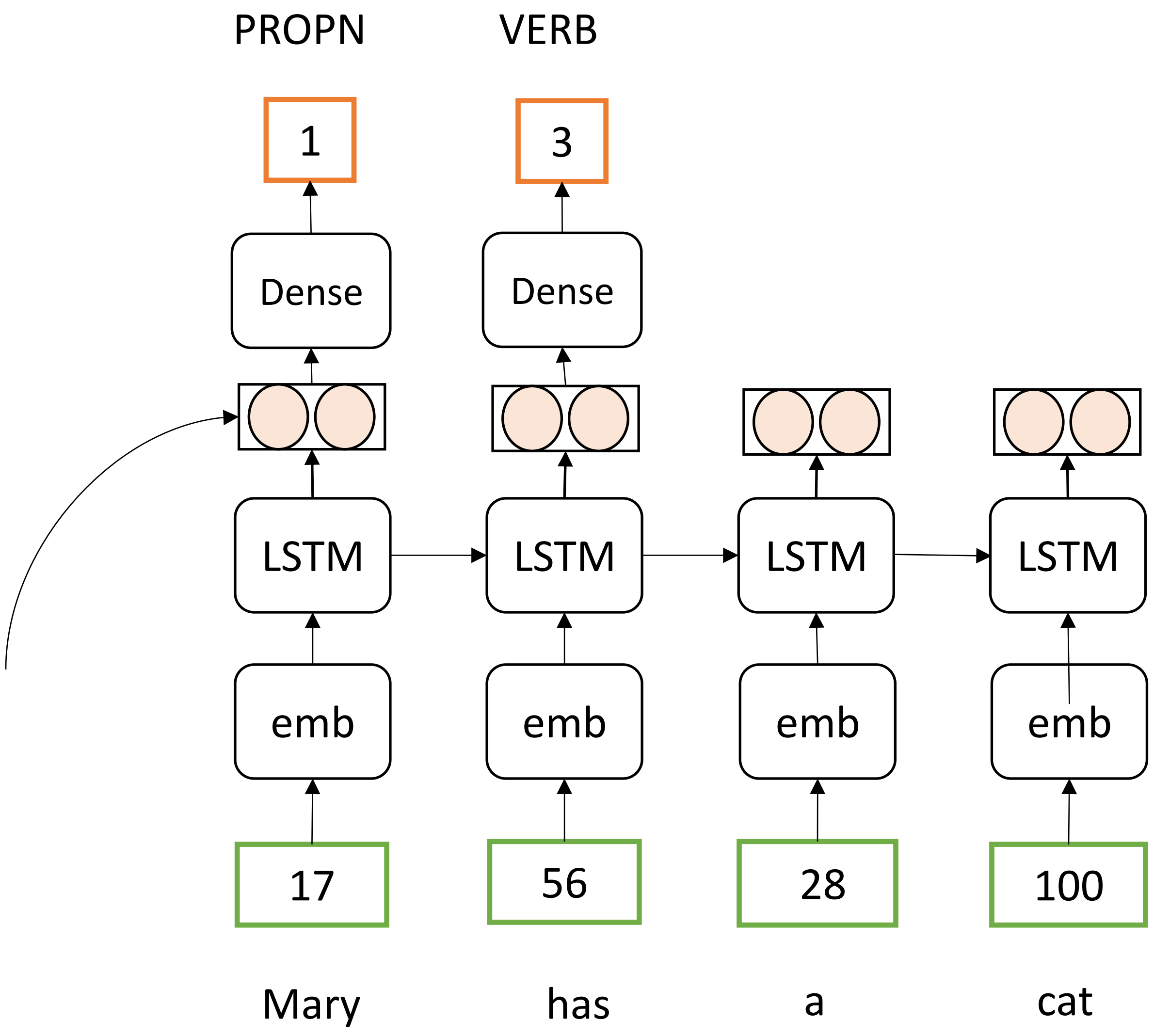
model = …

…

We add the embedding layer
We add the LSTM layer

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

The output of the LSTM must be a vector for each word.
…
model.fit(…)



tf.keras.layers.TimeDistributed

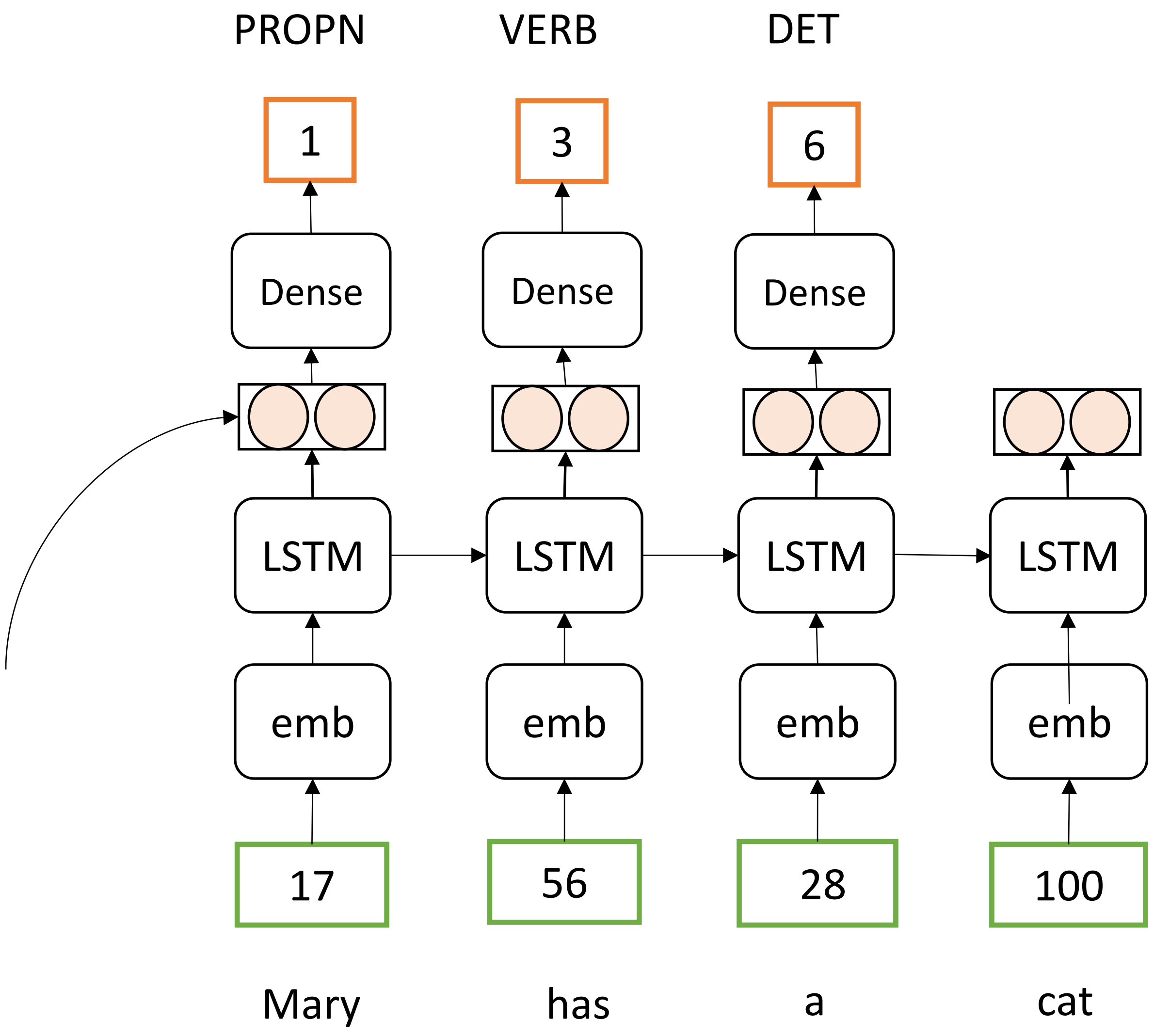
model = …

…

We add the embedding layer
We add the LSTM layer

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

The output of the LSTM must be a vector for each word.
…
model.fit(…)



tf.keras.layers.TimeDistributed

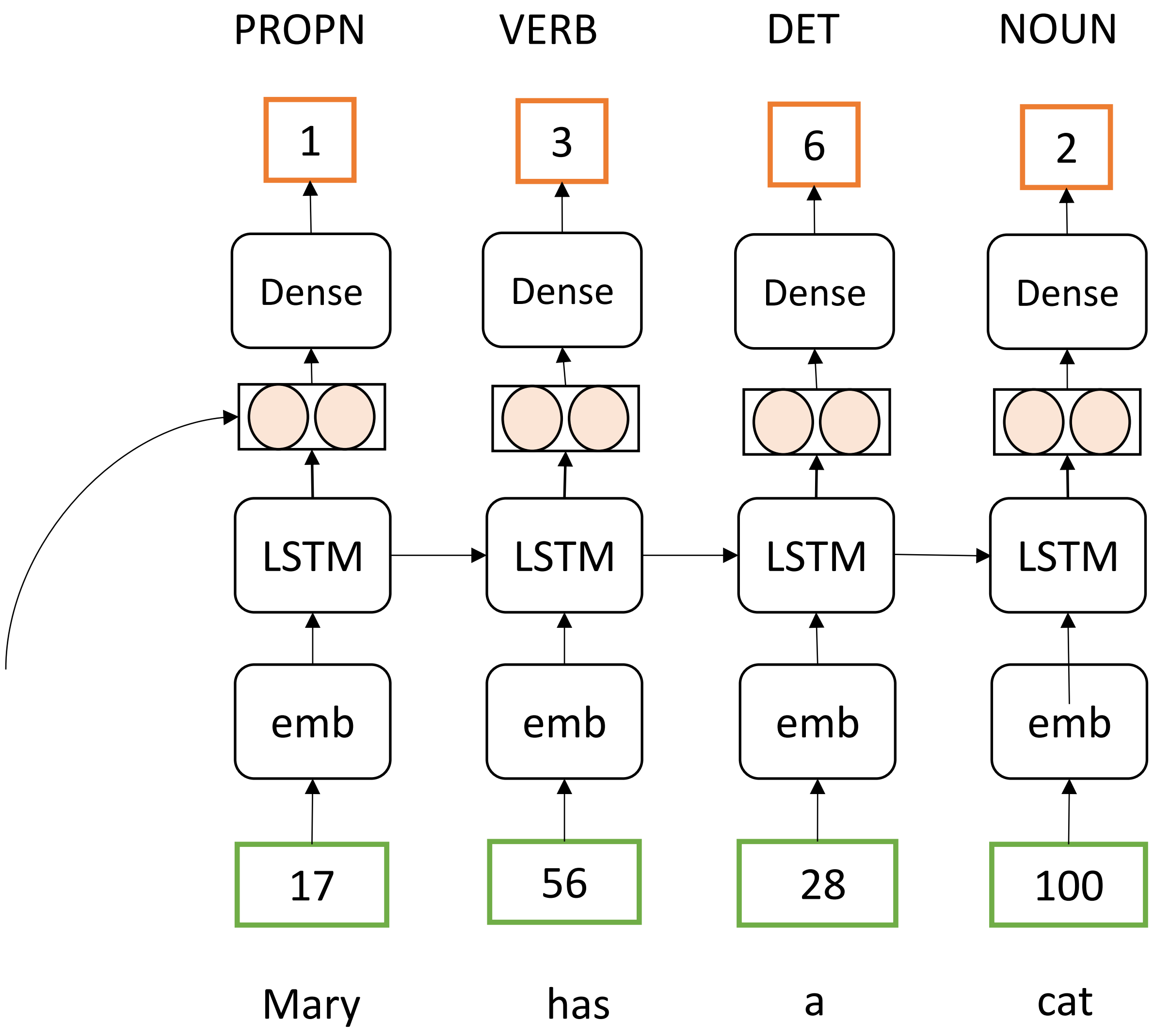
model = …

…

We add the embedding layer
We add the LSTM layer

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

The output of the LSTM must be a vector for each word.
…
model.fit(…)



Proposed planning

Week	Minimum amount of tasks that you should have completed at such point
September 29- October 3	Read the assigment and check materials (starts on October 1)
October 6 – 10	Implement functions to process the UD English treebank, remove multiword and empty lines.
October 13 – 17	Implement functions for tokenization and mapping samples to ids
October 20 – 24	Implement functions for training and evaluation of the model. Consider some variants of the network modifying the parameters.
October 27 – 31	Implement generation of labels for unseen sentences. Test other non-English treebanks.
Until November 3 (Monday, 15.30!!)	Writing the report and submission