Seq2seq model
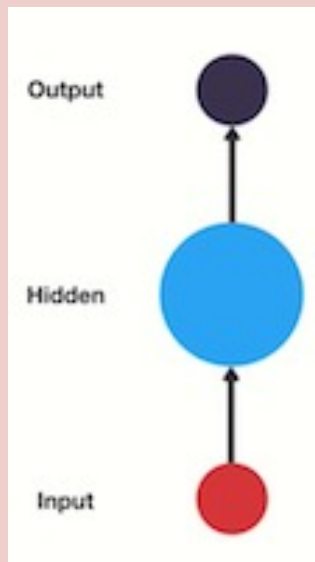# (Encoder-Decoder model)
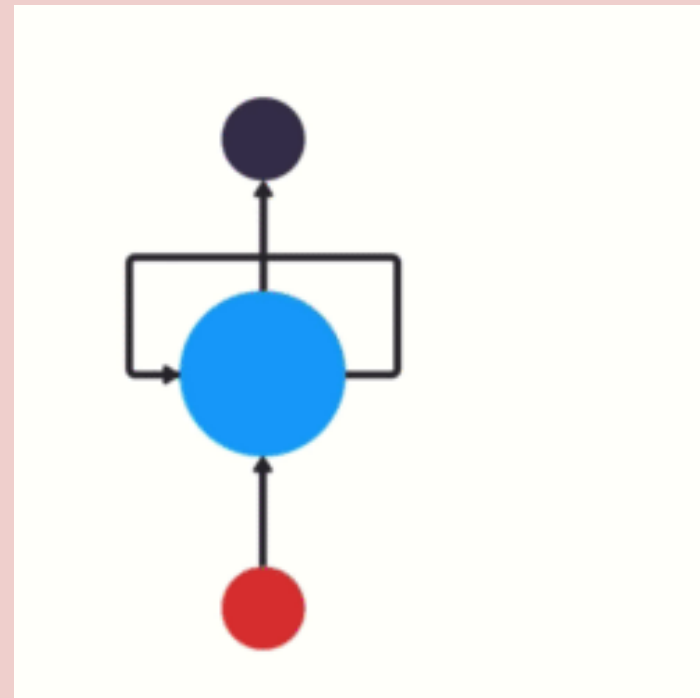
Michel RIVEILL

michel.riveill@univ-cotedazur.fr

Remember
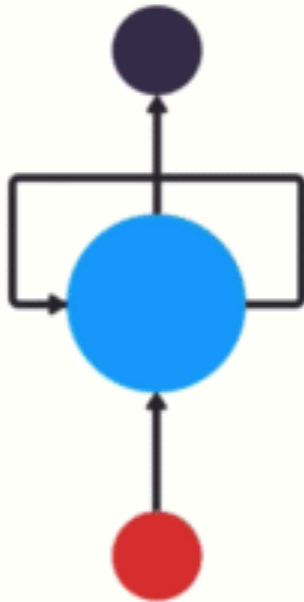
From	To
Output Hidden Input	

Remember

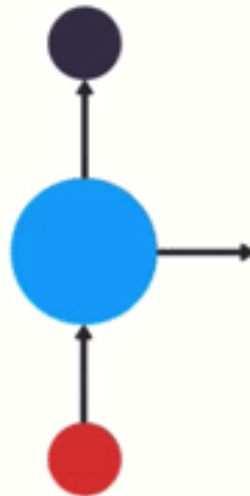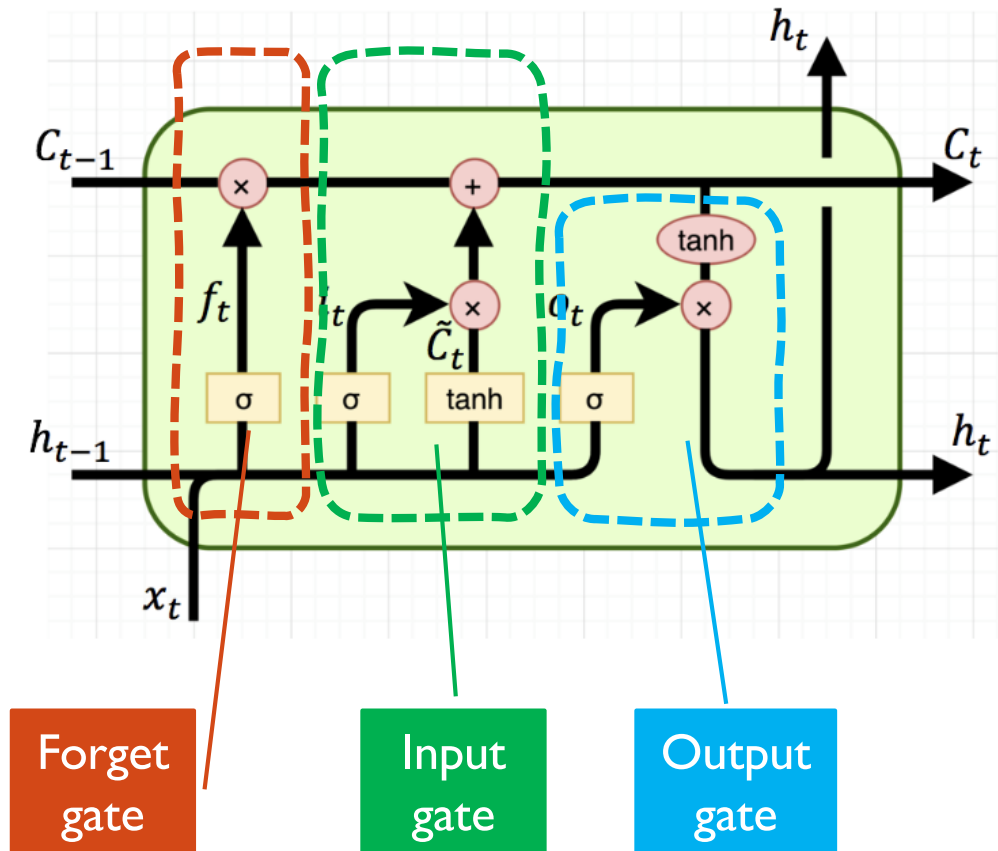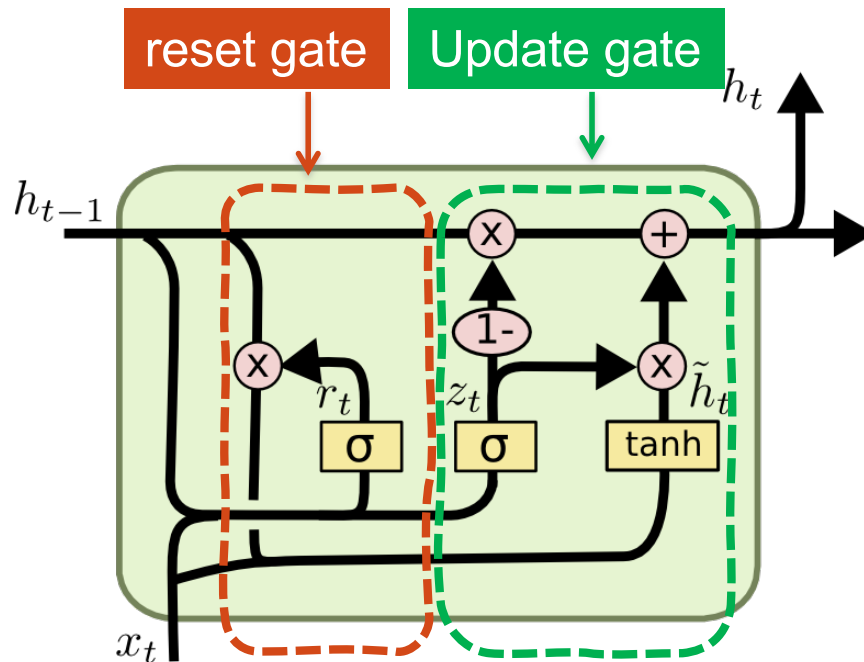| From	To

RNN in action

LSTM cell

▸ Cell made up of three "gates": these are calculation zones which regulate the flow of information (by carrying out specific actions).

  ▸ Forget gate (porte d'oubli)

  ▸ Input gate (porte d'entrée)

  ▸ Output gate (porte de sorti

▸ Cell state (état de la cellu

  ▸ Like residual

▸ Hidden state (état caché)

GRU – gated recurrent unit

▸ GRU = a light LSTM Cell

$$z_t = \sigma \left(W_z \cdot [h_{t-1}, x_t] \right)$$

$$r_t = \sigma \left(W_r \cdot [h_{t-1}, x_t] \right)$$

$$\tilde{h}_t = \tanh \left(W \cdot [r_t * h_{t-1}, x_t] \right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- It combines the forget and input into a single update gate.
- It also merges the cell state and hidden state.
→ This is simpler than LSTM.

Bi-directional RNNs

▸ RNNs can process the input sequence in forward and in the reverse direction
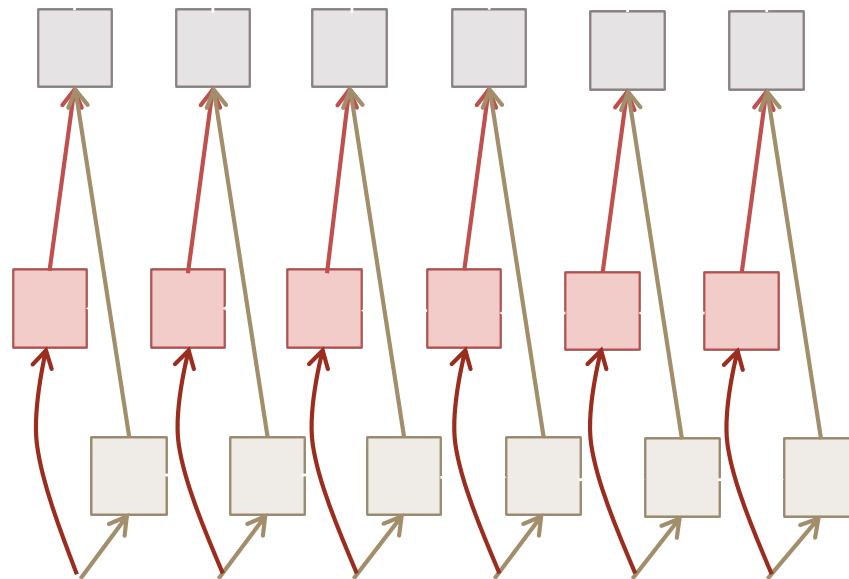
Popular in speech recognition used also with text

Different RNN architecture

Last week

one to one one to many many to one many to many many to many

This week
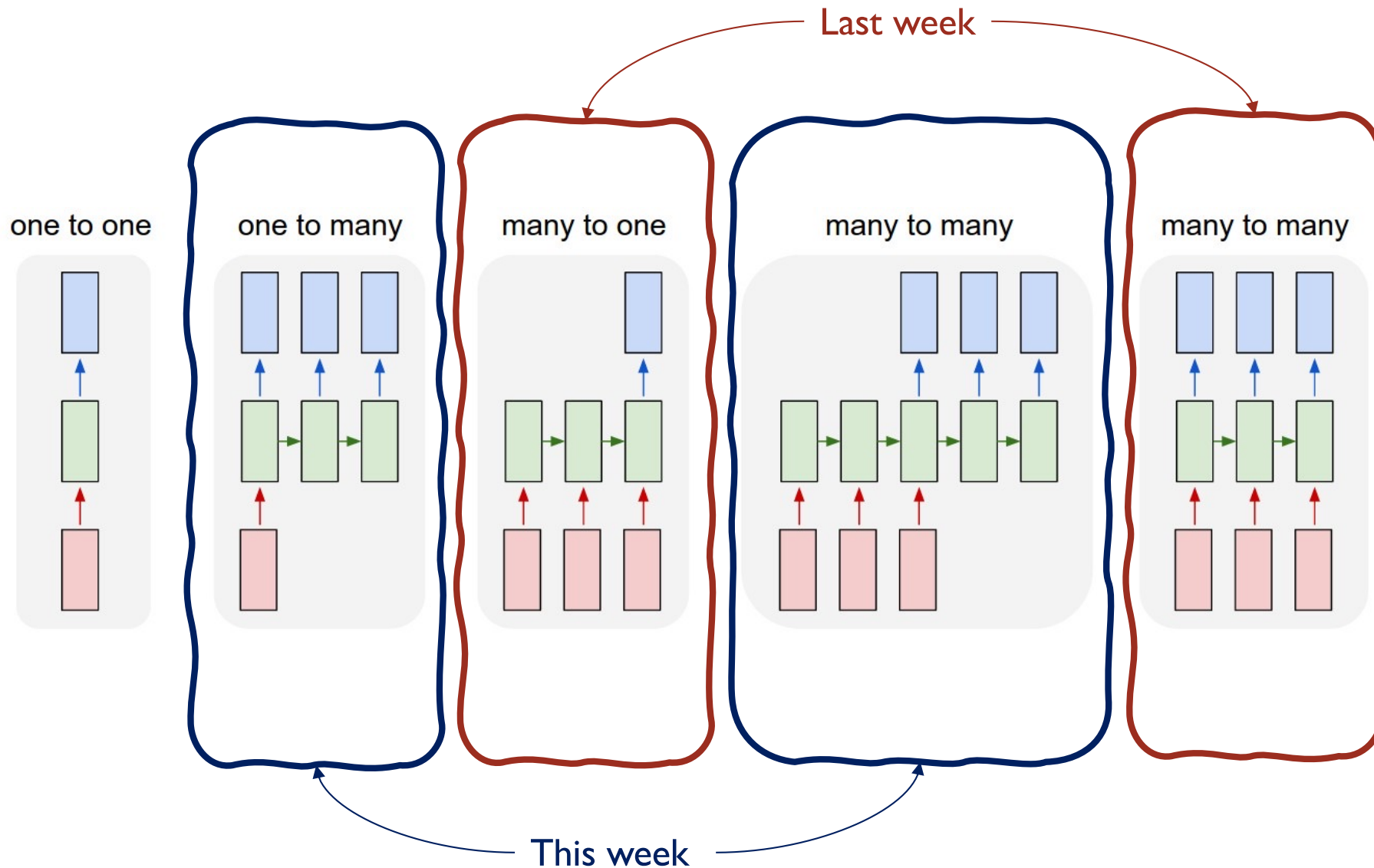
Seq2seq model
(Encoder-Decoder model)

len(input) != len(output)
And generally, the two lengths are unknow

Auto-encoder architure

Input Features

Encoding Network

Decoding Network

Reconstructed Features

Lower-Dimensional Latent Space Representation
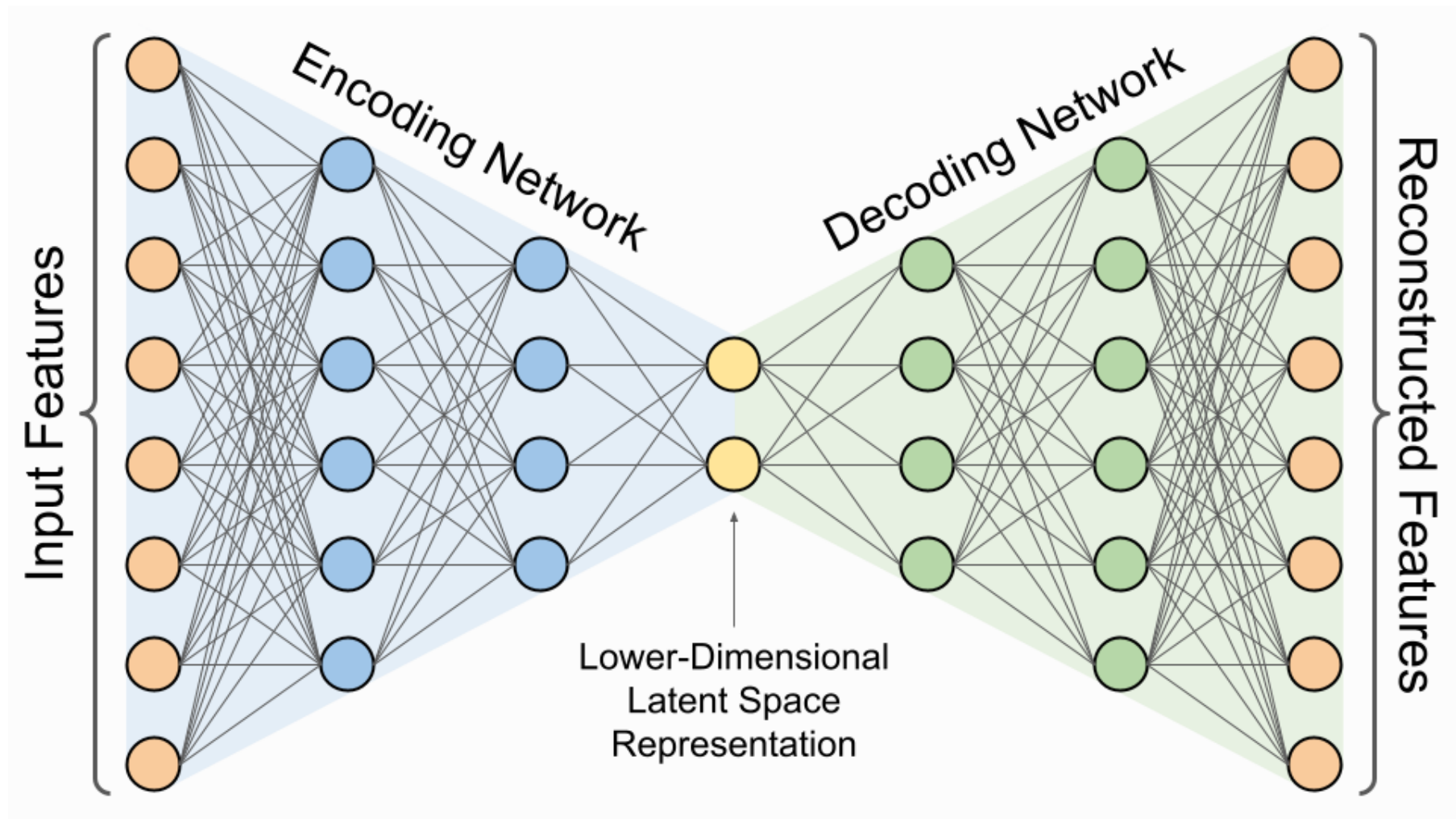
Seq2seq – original approach

- Extend encoder-decoder architecture
  - to a sequence data
  - in order to develop an architecture capable of generating contextually appropriate, arbitrary length, output sequences

A B C → → W X Y Z
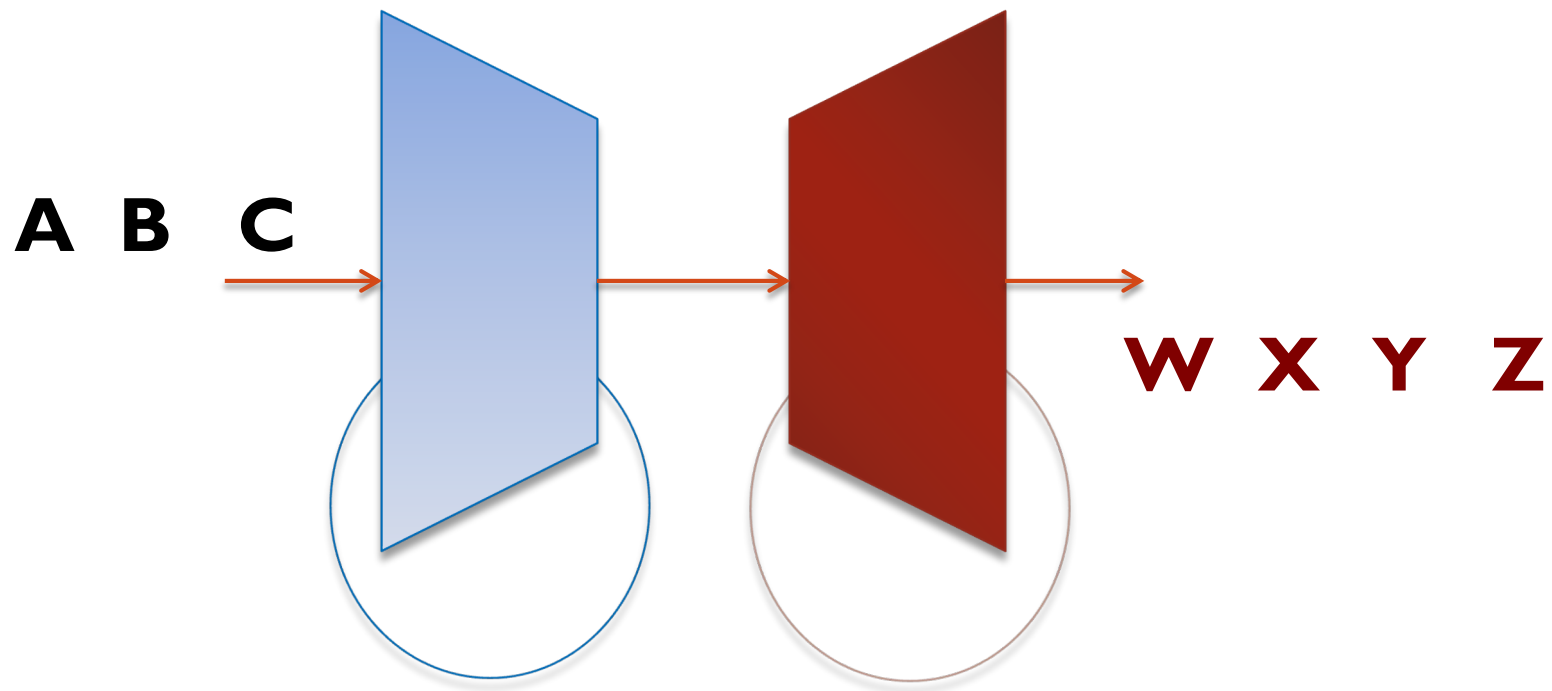
Seq2Seq model

- Applications with text
  - Machine translation
  - Text summarization
  - Question answering
  - Dialogue modeling
- But also
  - Forecasting
  - Image captioning

A happy dog is standing in the ocean
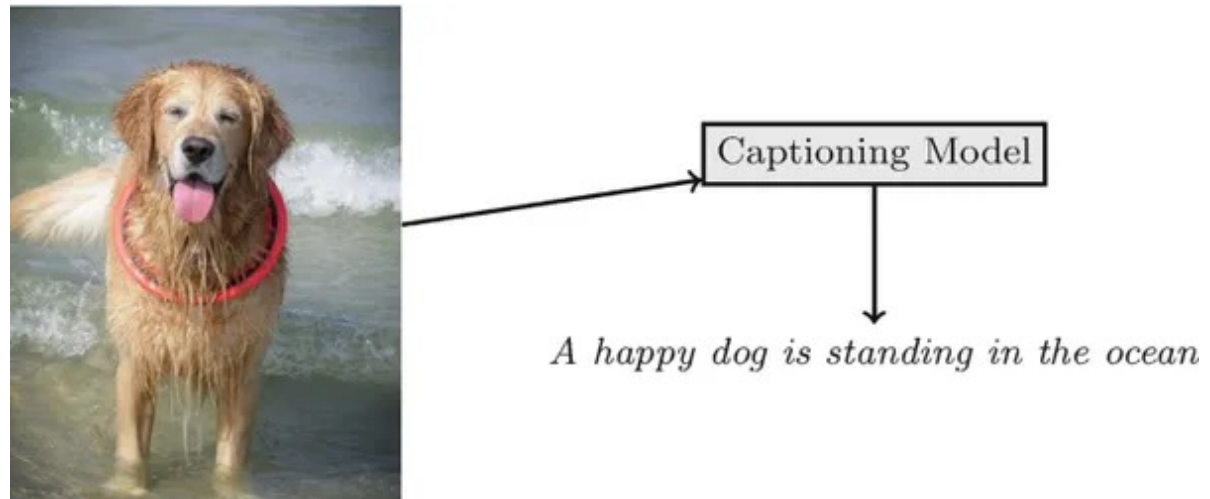
Image captioning model
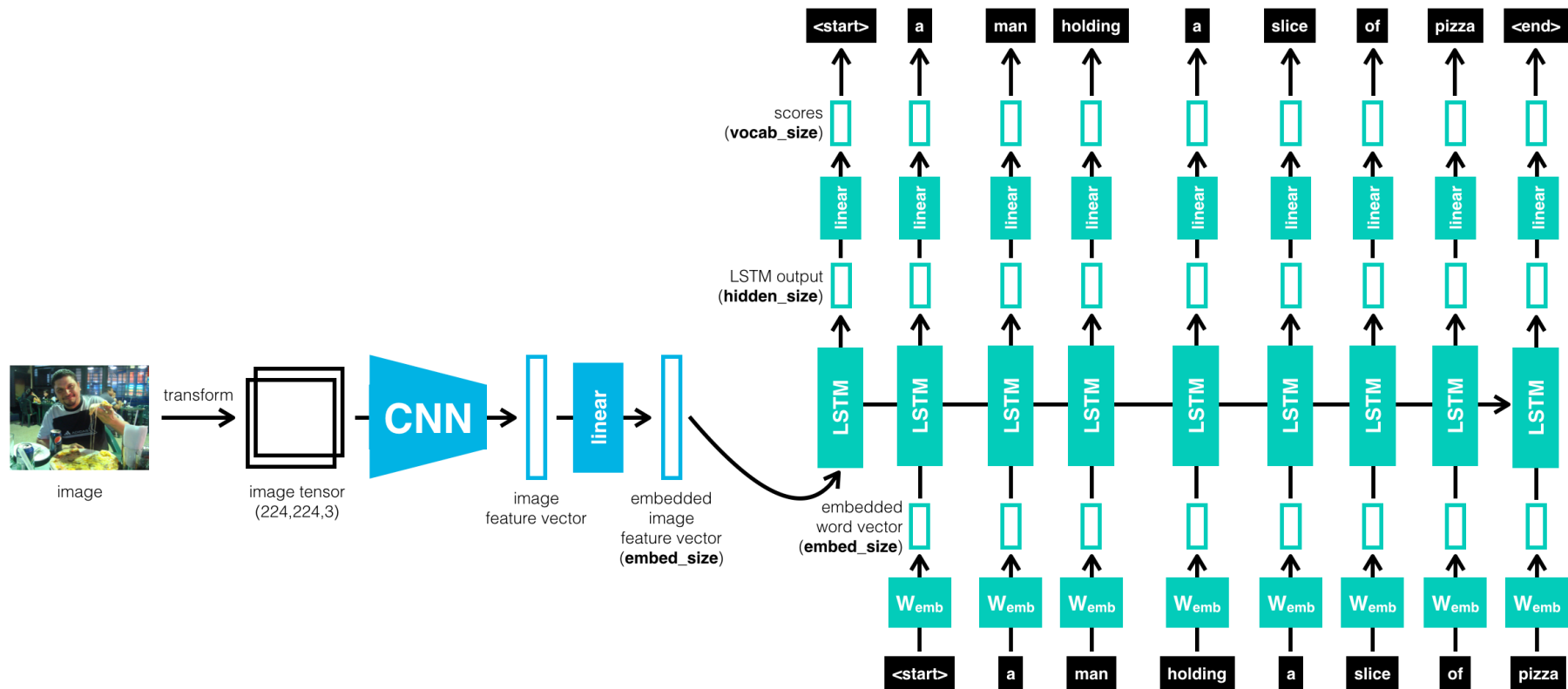
Seq2seq – original approach

▸ Extend encoder-decoder architecture to a sequence data
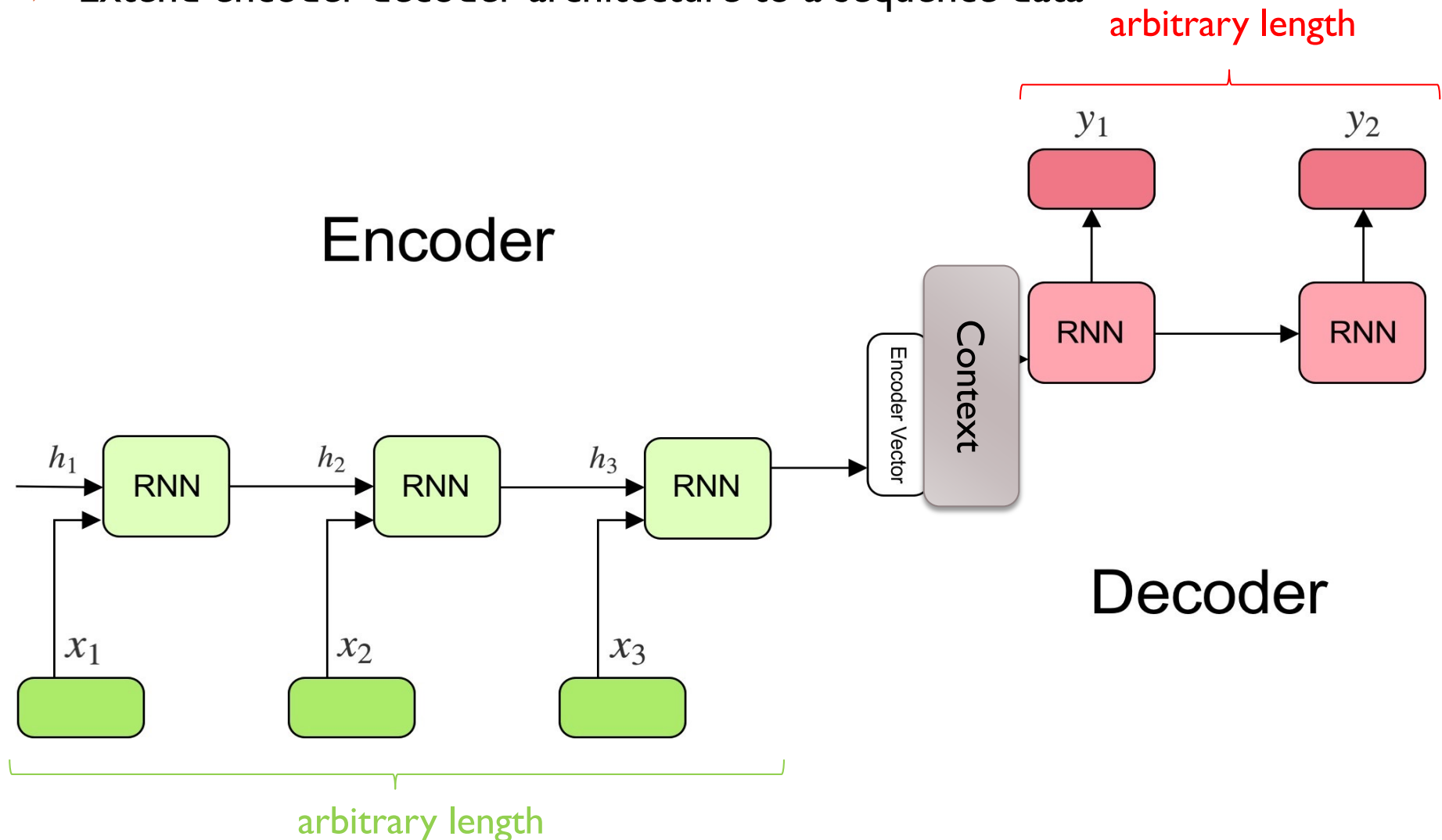
Sequence-to Sequence Architectures

‣ Three main part:
  ‣ Encoder: processes the input sequence (ordinary sequence-to-vector RNN)

  ‣ Context: output of the encoder
    ‣ is usually a simple function of its final hidden state (h + c)
    ‣ aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions

  ‣ Decoder: is conditioned on the context to generate the output sequence
    ‣ the context acts as the initial hidden state of the decoder part of the model
    ‣ produces output at each step

How to train seq2seq architecture

Without teacher forcing

model.fit(input_sent, output_sent)

Decoder part
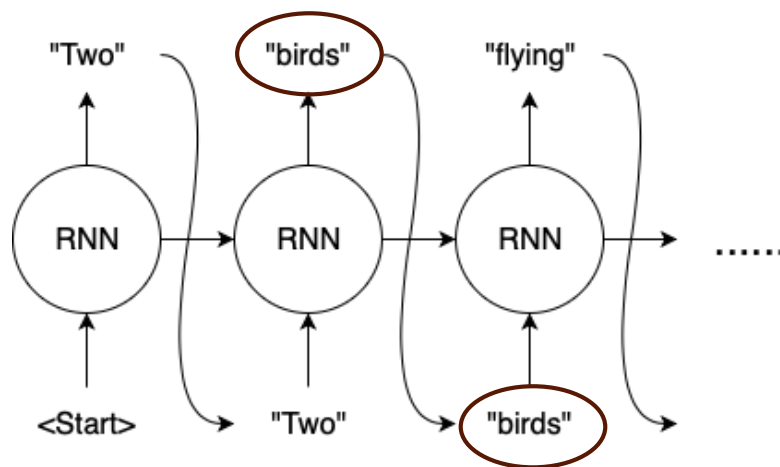
An error is propagated during
the training phase → penalizes it

With teacher forcing

model.fit([input_sent, teacher_sent],
output_sent)

Decoder part
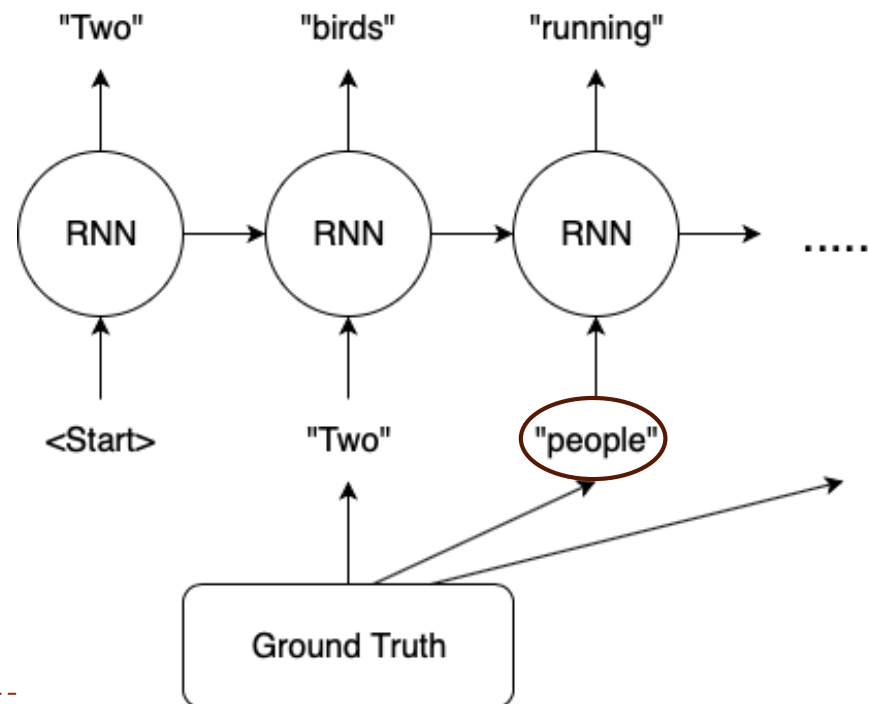
The teacher containt the ground truth

How to infer with seq2seq architecture

▸ Without/ With teacher forcing → no access to a ground Truth

▸ Greedy approach

1. Encode the sentence

2. iterate to successively decode each time step, reusing the time steps already decoded

   • Then reuse step by step the prediction

• Stop when <end> or max_length

Two birds flying in the sky <end>

Deux oiseaux volent dans le ciel

<start> Two birds flying in the sky
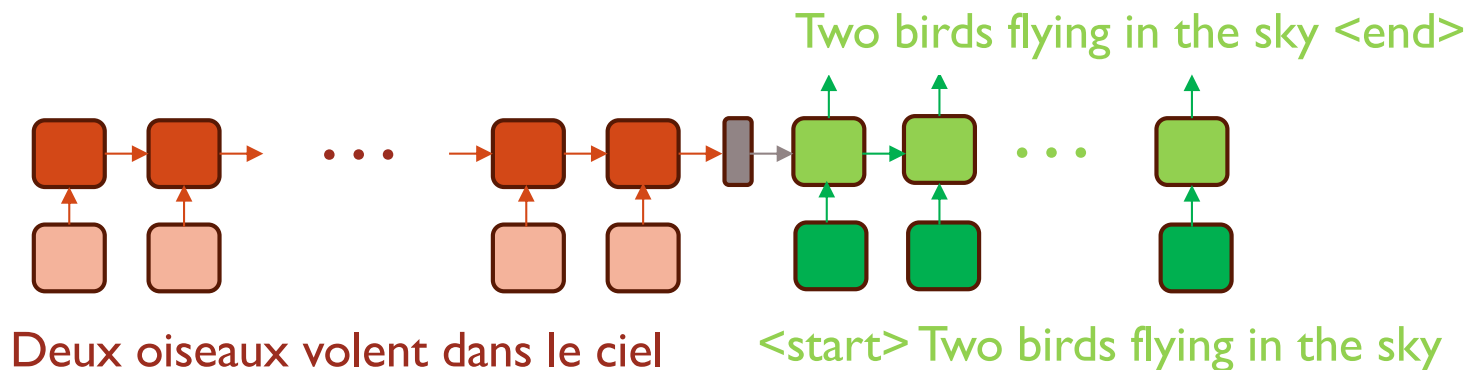
Pros and Cons of Teacher Forcing

- Pros:
  - If we do not use Teacher Forcing
    - the hidden states of the model will be updated by a sequence of wrong predictions
    - **errors will accumulate**
    - and it is difficult for the model to learn from that.
  - **Training with Teacher Forcing converges faster**.

- Cons:
  - **Unfortunately, during inference, there is no ground truth available**
    - the RNN model will have to re-inject its own prediction for the next prediction.
  - There is a difference between
    - learning (no propagation of error)
    - inference (propagation of error),

    which leads to poor performance and model instability.
  - This phenomenon is known as **exposure bias** in the litterature.

Train seq2seq model
Step 1 : encode input sentence

$$h_1^E$$

Hidden layer

Input layer   The   brown   dog   ran
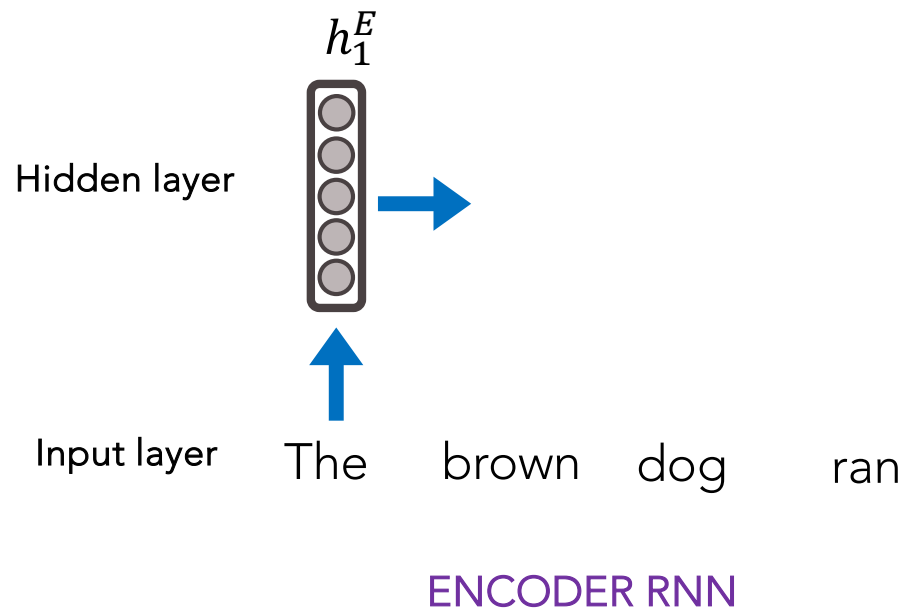
ENCODER RNN

Train seq2seq model
Step 1 : encode input sentence

Remember: it's an iterative process until
the end of the input sentence

h_1^E h_2^E

Hidden layer

Input layer   The   brown   dog   ran
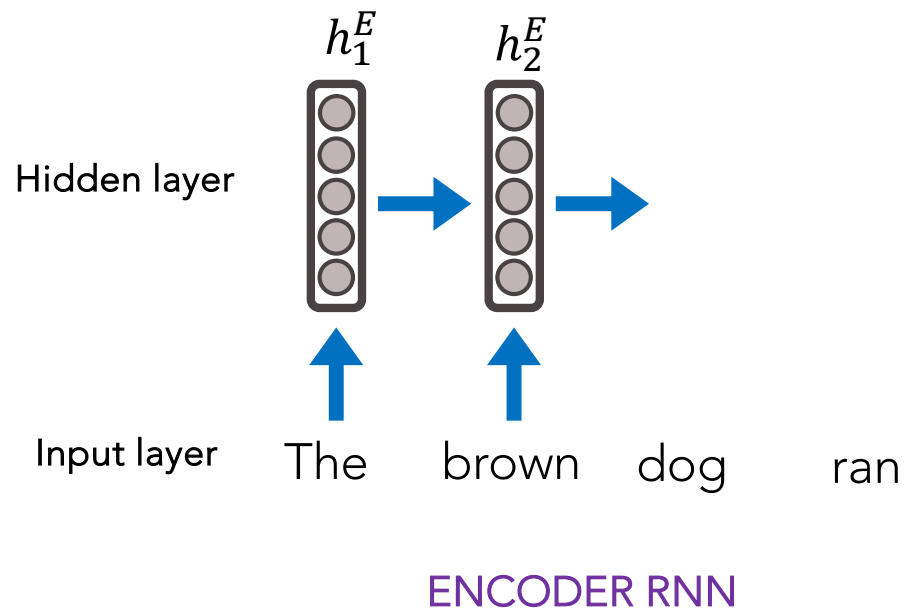
ENCODER RNN

Train seq2seq model
Step 1 : encode input sentence

The final hidden state of the encoder RNN
is the initial state of the decoder RNN

$$h_1^E \quad h_2^E \quad h_3^E \quad h_4^E$$

Hidden layer

Input layer   The   brown   dog   ran

ENCODER RNN
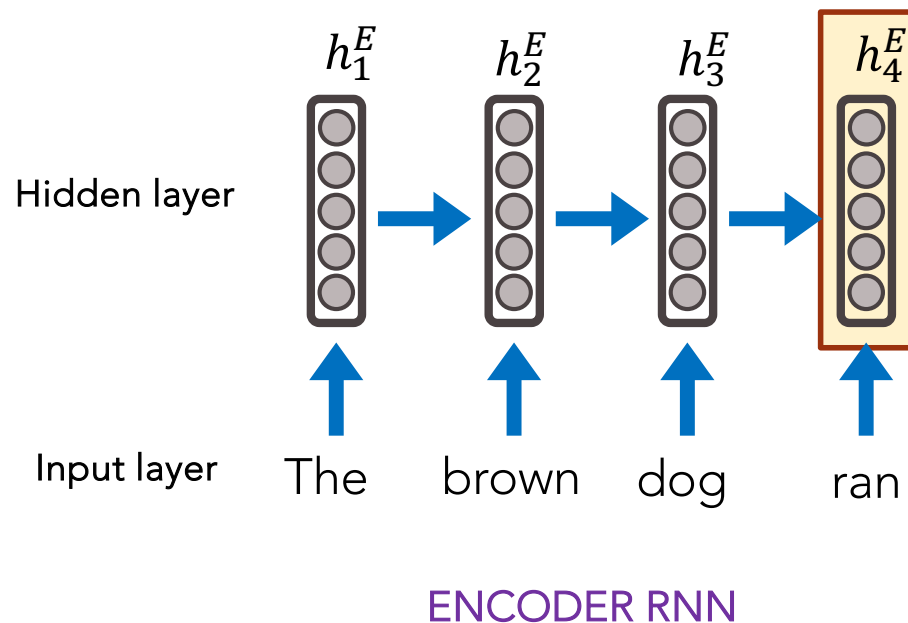
Train seq2seq model
Step 2 : decode the sentence

Remember: teacher, help in this task

$\hat{y}_1$

Hidden layer h_1^E h_2^E h_3^E h_4^E h_1^D

Input layer   The   brown   dog   ran   <s>   Le   chien   brun   a   couru

ENCODER RNN   DECODER RNN
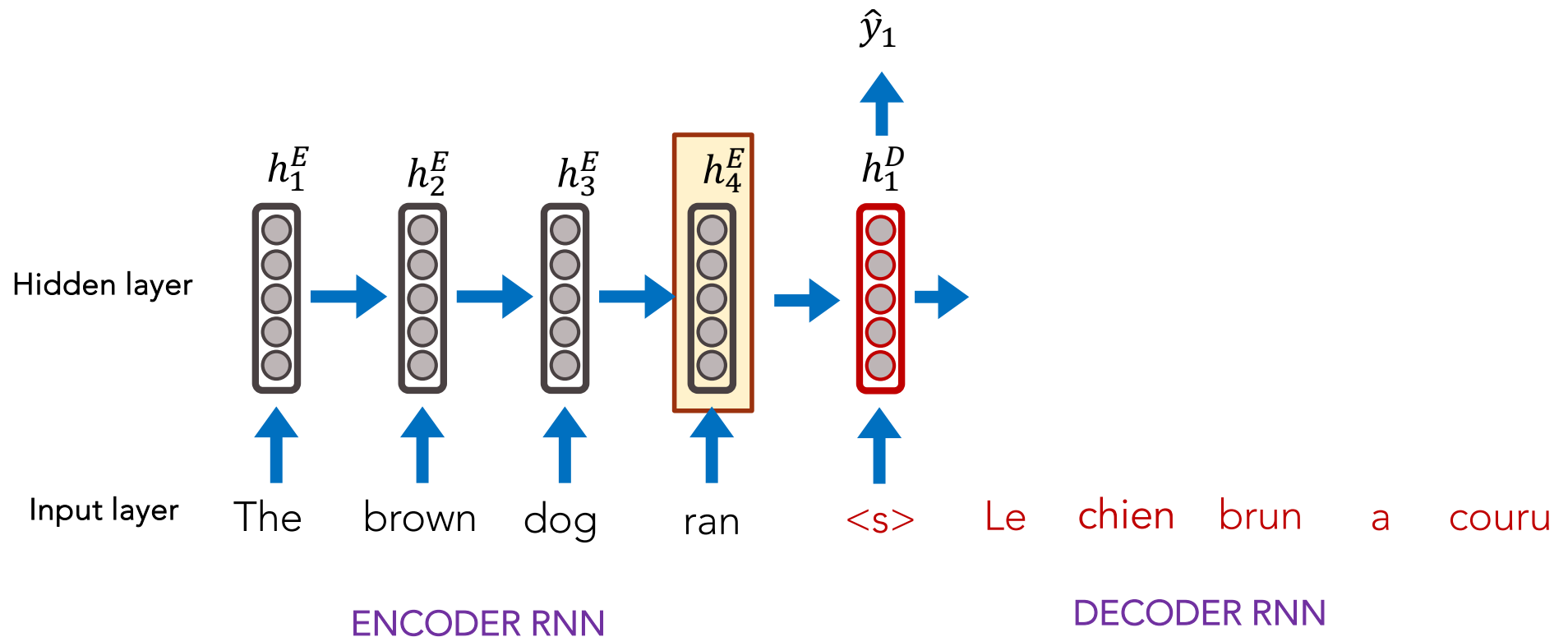
Train seq2seq model
Step 2 : decode the sentence

Remember: it's also an iterative process
until the end of the teacher sentence

ENCODER RNN   DECODER RNN

Hidden layer

Input layer

h_1^E h_2^E h_3^E h_4^E h_1^D h_2^D h_3^D h_4^D h_5^D h_6^D

$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$

The   brown   dog   ran   <s>   Le   chien   brun   a   couru

ENCODER RNN

DECODER RNN
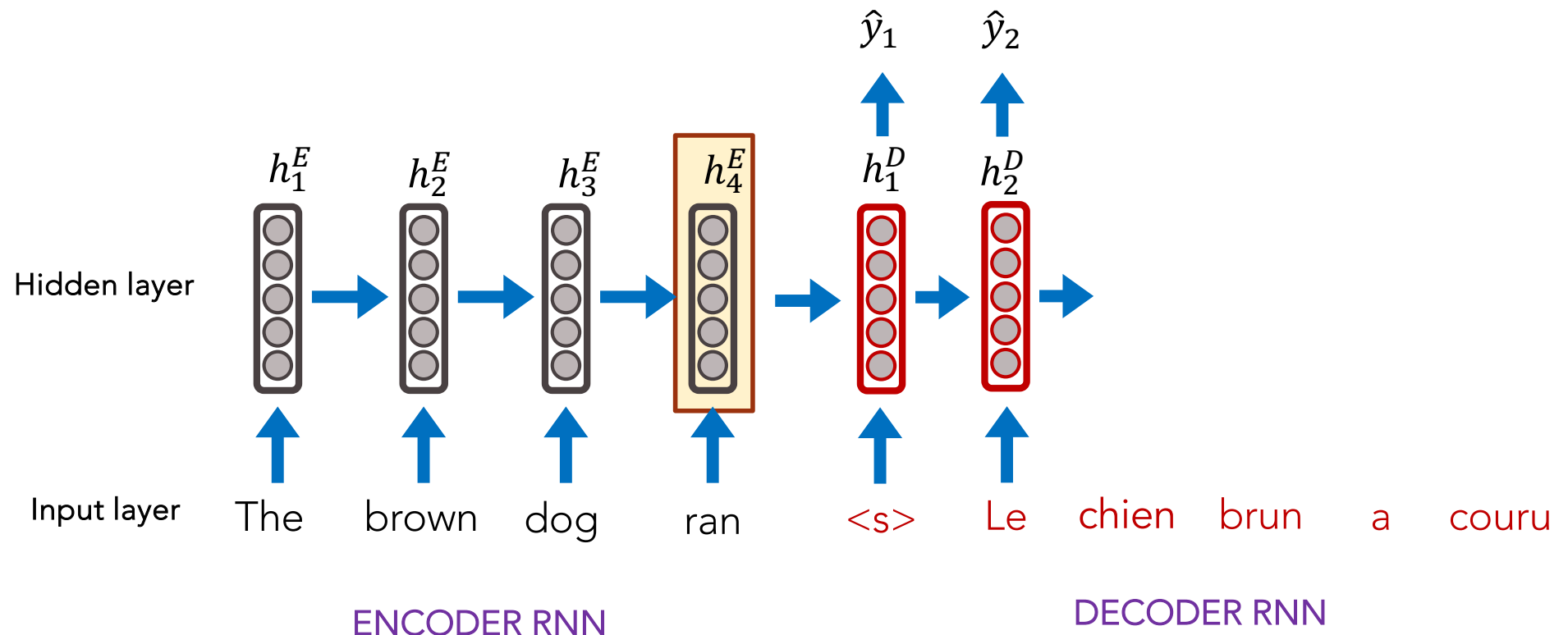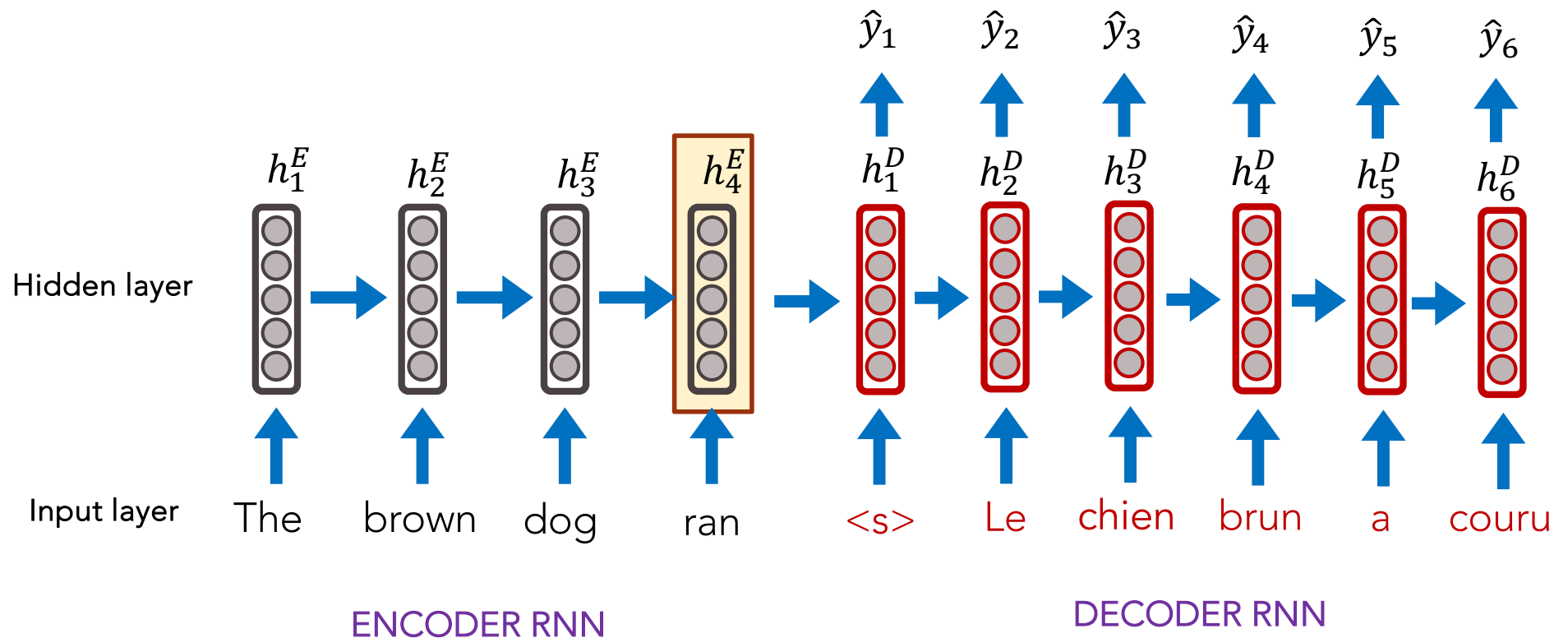
How to build Teacher Seq2Seq model

▸ For example

  ▸ Translation from Spanish to English

▸ Input sequence = Spanish = (None, None, in_features)

  ▸ in_features: spanish_vocab_size=521
  ▸ Use spanish_vectorizer

▸ Output sequence = English = (None, None, out_features)

  ▸ out_features: english_vocab_size = 262
  ▸ Use english_vectorizer

▸ Embedding dim → as usual (**50**, 100, 150, 300)

▸ Latent dim

  ▸ Represent the size of the latent space (**64**, 128, 256 or more)
  ▸ Latent space = 2*latent_dim for LSTM / latent_dim for GRU

How to build Teacher Seq2Seq model

Define encoder

```
enc_inputs = Input(shape=(None,), name="spanish_input")
# Why input_shape=(None, None) ?
enc = Embedding(sp_vocab_size, emb_dim, name="sp_embedding")(enc_inputs)
_, enc_state_h, enc_state_c = LSTM(latent_dim, return_sequences=False,
                                    return_state=True)(enc)
```

Define context

```
context = [enc_state_h, enc_state_c]
```

How to build Teacher Seq2Seq model

```
# Define decoder layers
    layer_embedding = Embedding(en_vocab_size, emb_dim, name="en_embedding")
    layer_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
                                              # We use return states in inference.
    layer_dense = Dense(en_vocab_size, activation='softmax')


# Define decoder
    dec_inputs = Input (shape=(None,))
    # Why input_shape=(None, None) ?
    dec = layer_embedding (dec_inputs)
    dec, _, _ = layer_lstm(dec, initial_state=context)
    dec_outputs = layer_dense(dec)


# Define the Encoder_Decoder model
    model = Model ([enc_inputs, dec_inputs], dec_outputs)
```

Seq2Seq model with teacher

spanish_input	input:	[(None, None)]
InputLayer	output:	[(None, None)]

sp_embedding	input:	(None, None)
Embedding	output:	(None, None, 50)

input_15	input:	[(None, None)]
InputLayer	output:	[(None, None)]

lstm_12	input:	(None, None, 50)
LSTM	output:	[(None, 64), (None, 64), (None, 64)]

en_embedding	input:	(None, None)
Embedding	output:	(None, None, 50)

lstm_13	input:	[(None, None, 50), (None, 64), (None, 64)]
LSTM	output:	[(None, None, 64), (None, 64), (None, 64)]

dense_22	input:	(None, None, 64)
Dense	output:	(None, None, 262)

Predict with seq2seq model
Step 1 : Use encode to define context

Hidden layer

Input layer

The   brown   dog   ran

ENCODER RNN   DECODER RNN
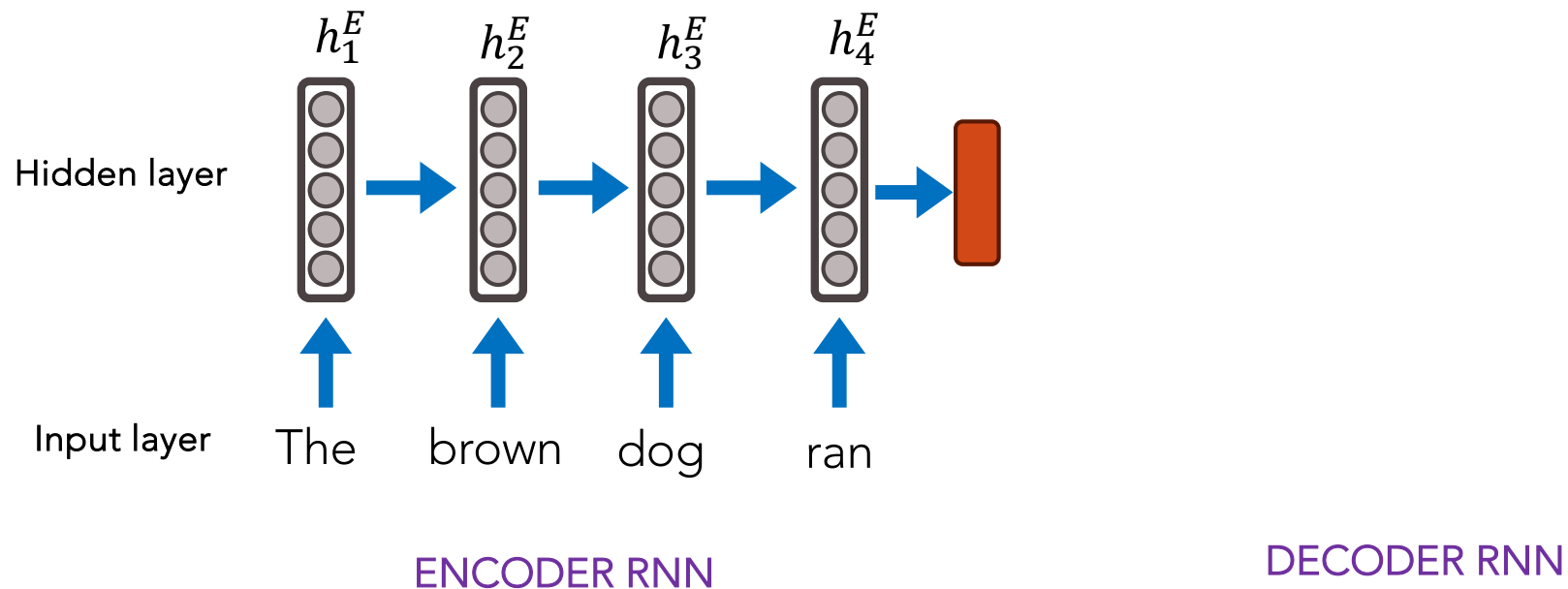
Predict with seq2seq model
Step 2 : decode the first output
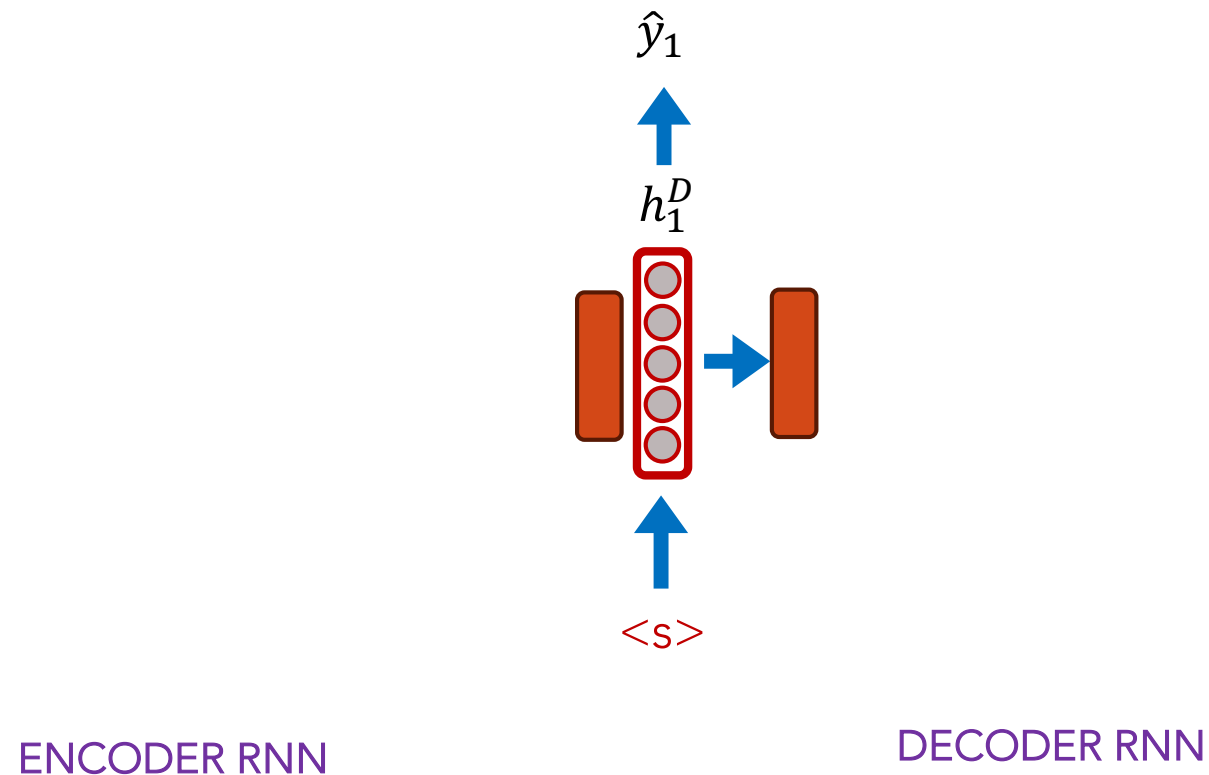
$$\hat{y}_1$$

$$h_1^D$$

<s>

ENCODER RNN

DECODER RNN

Predict with seq2seq model
Step 2 : step by step... decode sentence

Decode step by step

Reuse at each step, the previous output

$$\hat{y}_1 \qquad \hat{y}_2$$

$$h_2^D$$

$$\hat{y}_1$$

ENCODER RNN   DECODER RNN

Predict with seq2seq model
Step 2 : step by step... decode sentence

Decode step by step

Reuse at each step, the previous output

Stop, when generate "stop" label

$\hat{y}_1 \qquad \hat{y}_2 \qquad \hat{y}_3 \qquad \hat{y}_4 \qquad \hat{y}_5 \qquad \text{<s>}$
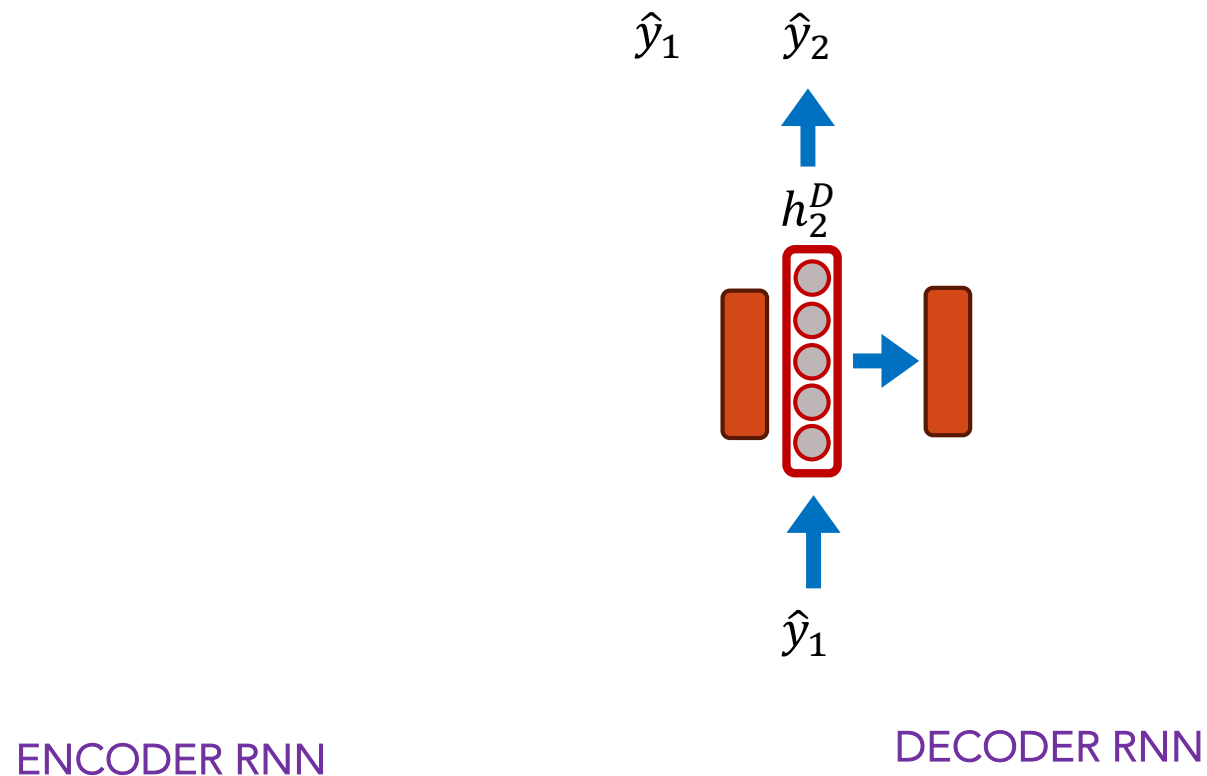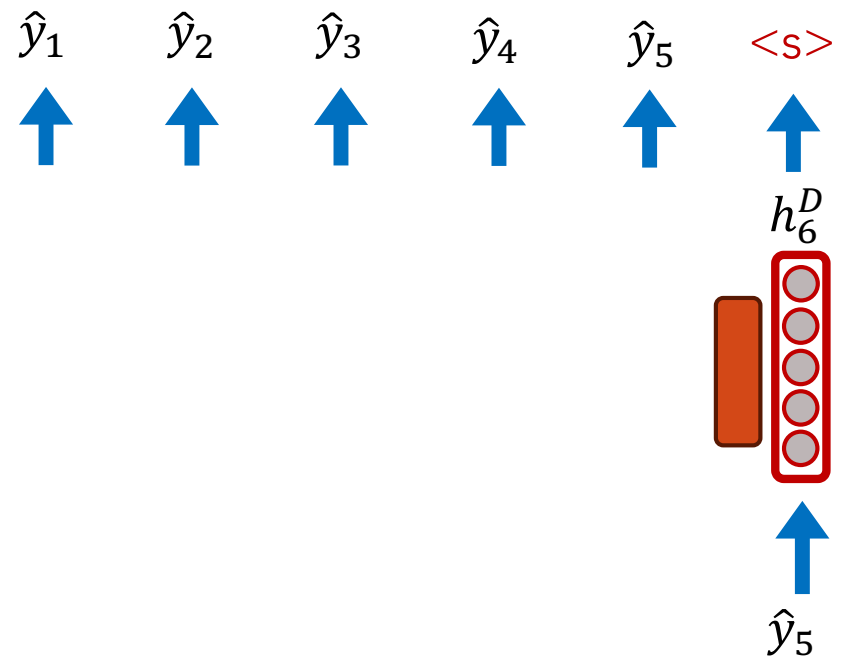
h_6^D

$\hat{y}_5$

ENCODER RNN

DECODER RNN

How to predict ?

Build encoder and decoder model

encoder_model = Model(enc_inputs, context)

dec_inputs = Input(shape=(None,)
dec_input_h = Input(shape=(latent_dim,))
dec_input_c = Input(shape=(latent_dim,))

dec = **layer_embedding**(dec_inputs)   # Same cell as previously
dec, dec_h, dec_c = **layer_lstm**(dec_inputs,   # Same cell as previously
                initial_state=[dec_input_h, dec_input_c])
decoder_outputs = **layer_dense**(dec)   # Same cell as previously

decoder_model = Model([dec_inputs, dec_input_h, dec_input_c],
                [decoder_outputs, dec_h, dec_c])

Encoder model

spanish_input	input:	[(None, None)]
InputLayer	output:	[(None, None)]

sp_embedding	input:	(None, None)
Embedding	output:	(None, None, 50)

lstm_14	input:	(None, None, 50)
LSTM	output:	[(None, 64), (None, 64), (None, 64)]
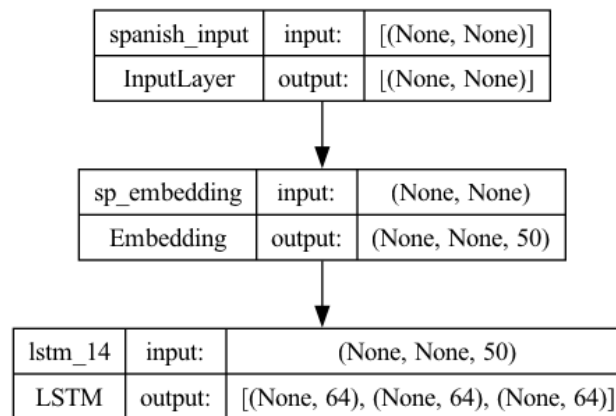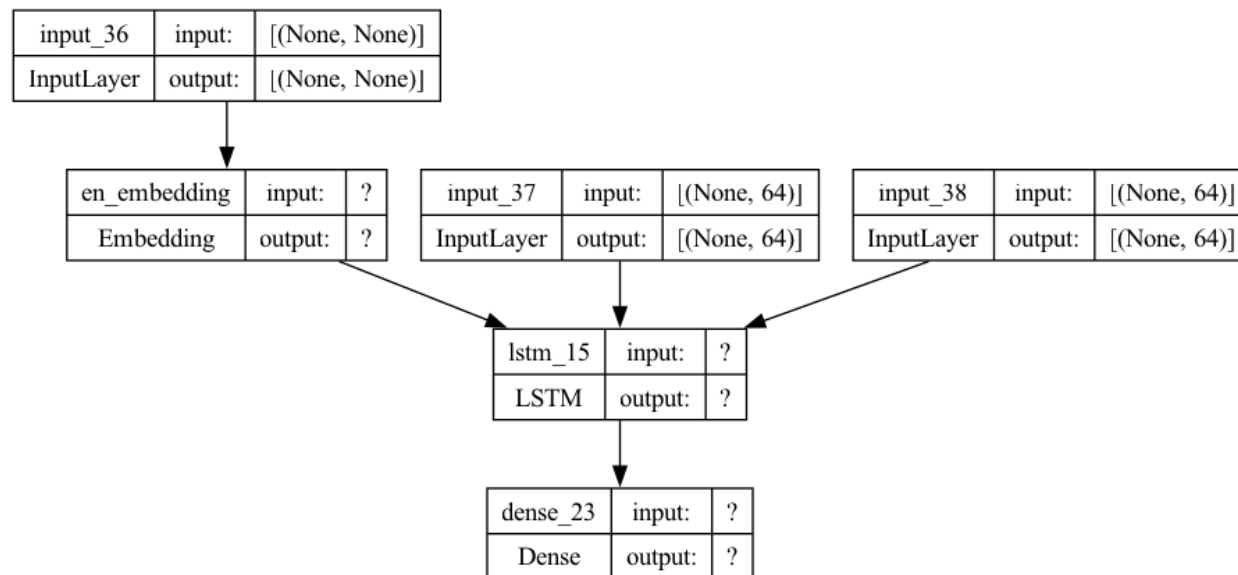
Decoder model

How to predict ?

```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)

    # Iterate over decoded sentence. Target_seq is the input of the decoder
    target_seq = np.zeros((len(input_seq), 1))
    target_seq[:, 0] = « initialize the first input »

    output_sequence = []  # Output_sequence is the output of the decoder
    For _ in range(max_output_length):
        output_value, h, c = decoder_model.predict( [target_seq] + states_value)
        # Update the target sequence (of length 1) and state
        target_seq[:, 0] = decode(output_value)
        states_value = [h, c]
        # extend output sequence
        output_sequence += [target_seq]

    return output_sequence # eventually format it
```
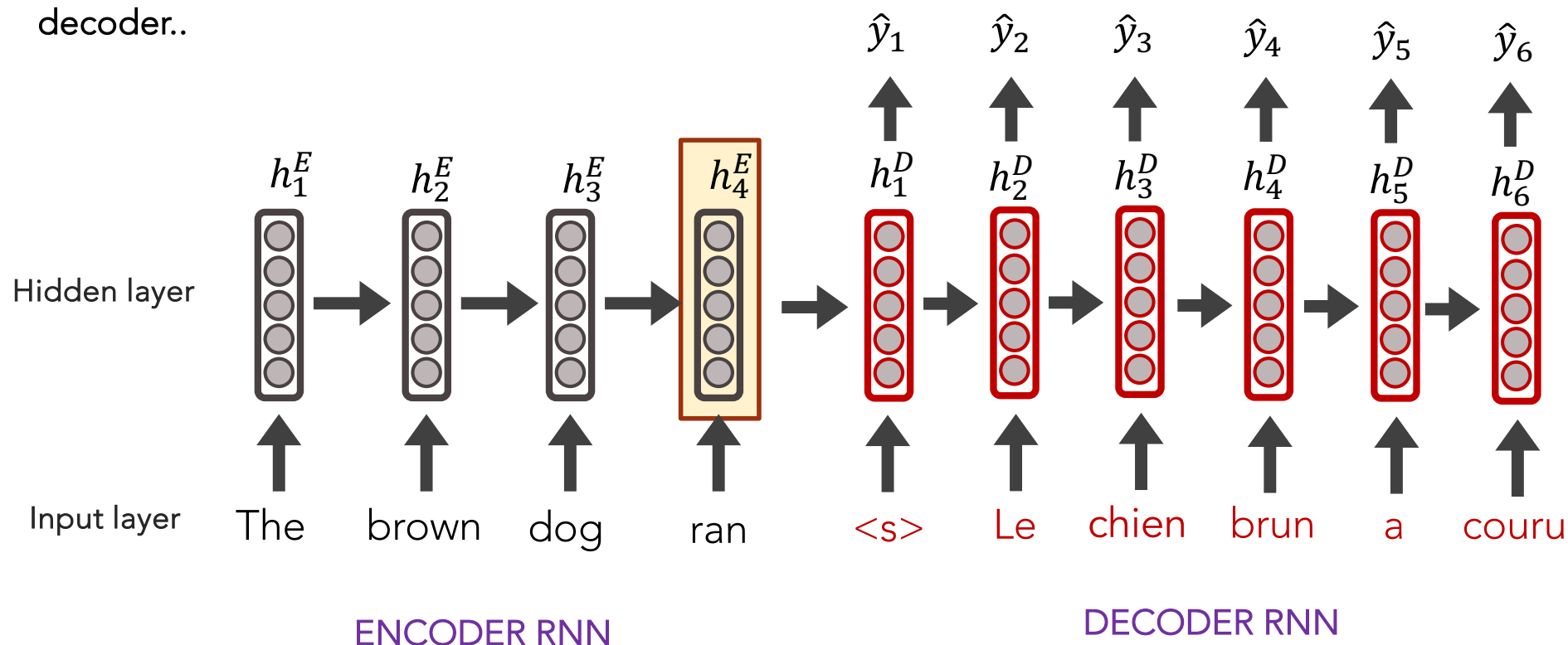
Must necessarily be adapted, as must the architecture of the network according to the problem :
- presence or absence of the embedding layer
- binary classification, categorical classification or mono or multi regression

Sequence-to-Sequence (seq2seq)

With a Seq2Seq model, we assume that the entire input sequence can be represented by a vector that is the only interaction between the encoder and the decoder..

Hidden layer

h_1^E h_2^E h_3^E h_4^E h_1^D h_2^D h_3^D h_4^D h_5^D h_6^D

$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$

Input layer

The brown dog ran <s> Le chien brun a couru

ENCODER RNN

DECODER RNN

Lab – Build a Deep Learning Translator

▶ Must necessarily be **finished before next week** as we will continue adding attentions to this model

▶ **Dataset:** download your own pair of language and prepare the dataset (code next slide)
  ▸ https://www.manythings.org/anki/
  ▸ For good performance, it is necessary to have a large dataset
  ▸ But unfortunately, training a recurrent network is time consuming
  ▸ We will therefore work with a reduced number of sentences
▶ Build a seq2seq neural network
  ▸ 2 possibilities
    ▸ At character level
    ▸ At word level (preferable)
  ▸ Over-fit your network (very low error rate) with a teacher
    ▸ We use only a training set (we predict on test)
    ▸ A very small validation split in order to visualize the overfitting
    ▸ No EarlyStopping
▶ Build model for inference and predict

Data preparation

```python
def step1(sent): # sent = on sentence in a language
    def unicode_to_ascii(s): # In order to reduce the possibility
        return ''.join(c for c in unicodedata.normalize('NFD', s) if
unicodedata.category(c) != 'Mn')
    sent = unicode_to_ascii(sent.lower().strip()) # Only lower charater


    # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",",
"¿")
    sent = re.sub(r"[^a-zA-Z?.!,¿]+", " ", sent)     # To be adapted
according to the languages chosen


    # creating a space between a word and the punctuation following it.
E.g. "he is a boy." => "he is a boy ."
    # Reference:- https://stackoverflow.com/questions/3645931/python-
padding-punctuation-with-white-spaces-keeping-punctuation
    sent = re.sub(r"([?.!,¿])", r" \1 ", sent)             # To be adapted
according to the languages chosen


    return '<start> ' + sent.strip() + ' <end>'      # Suppress extra space
```

Data preparation

```python
# Loading data
def read_data(path, num_examples):
    # path : path to spa-eng.txt file
    # num_examples : Limit the total number of training example for faster
training
    lines = io.open(path, encoding='UTF-8').read().strip().split('\n')
    print(lines[0])
    sentences1, sentences2= zip(*[[step1(sent) for sent in
l.split('\t')[:2]] for l in lines[:num_examples]])

    return np.array(sentences1), np.array(sentences2)
```

Data preparation

```python
# Search vocabulary and max_length for each language
def voc(lang):
    # a list of sentences in the same language
    lengths = [len(txt.split()) for txt in lang]
    vocab = set([w for txt in lang for w in txt.split()])

    return max(lengths), list(vocab), len(vocab)+2 # for padding and OOV


max_length1, vocab1, vocab_size1 = voc(sentences1)


# Build vectorizer layer
vectorizer1 = layers.TextVectorization(standardize=None,
                                       output_mode='int',
                                       vocabulary=vocab1,
                                       name= "language1")


# Do the same for language 2
```