Architecture Seq2Seq et attention

Michel.RIVEILL@univ-cotedazur.fr

2024-2025

Sommaire

- Architecture Seq2Seq
  - Construction
  - Entrainement
  - Inférence
- Le mécanisme d'attention
- Les attentions en Keras

Vecteurs d'embedding

- J'ai bien aimé vos propositions
  - Paella + France – Espagne = Cassoulet
  - Pasta + Japon - France = noddle
  - Steak + Vegetable – Meat = Salad
  - Pancake + France – Canada = Croissant
  - Shawarma + Turkish – Lebanon = Kebab

  - Rome + Romania – Italie = Bucharest
  - Paris + Duch – Berlin = French
  - Germany + Paris – France = Berlin

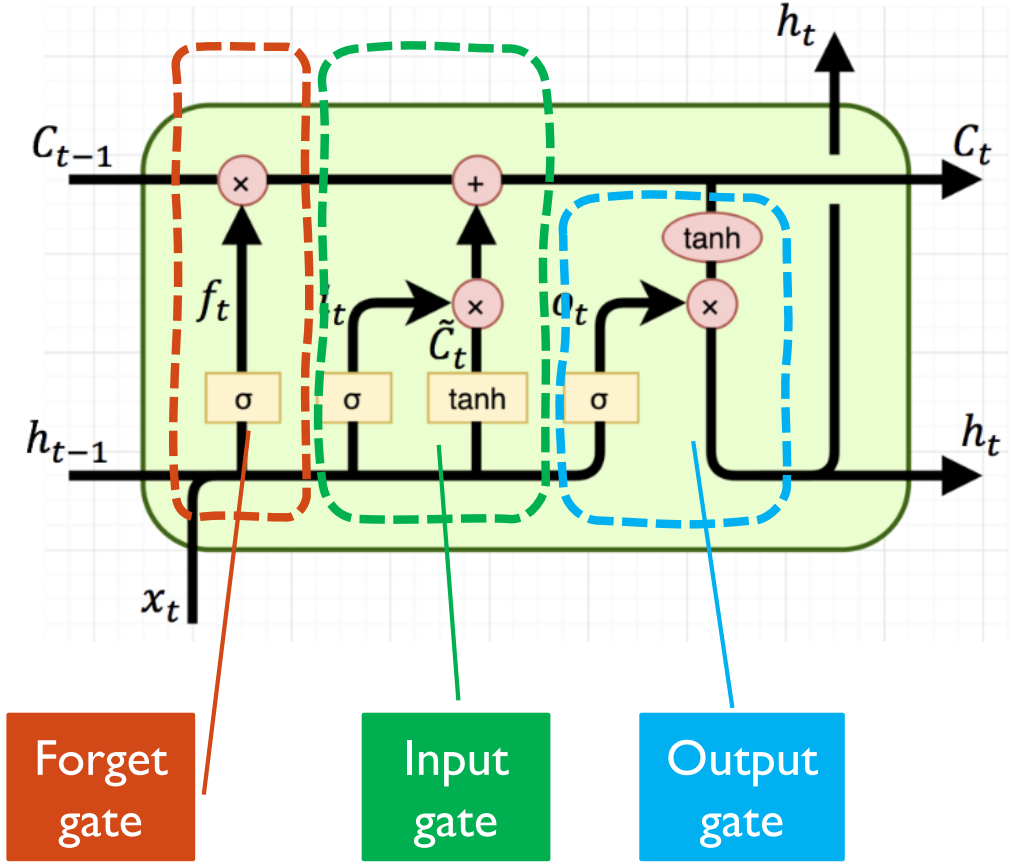
  - Boy + mother – father = girl

  - Planet + dry – water = martian

  - Music + jazz – blues = classical
- J'espère que cela vous aura permis de bien comprendre 'l'importance d'un bon embedding'

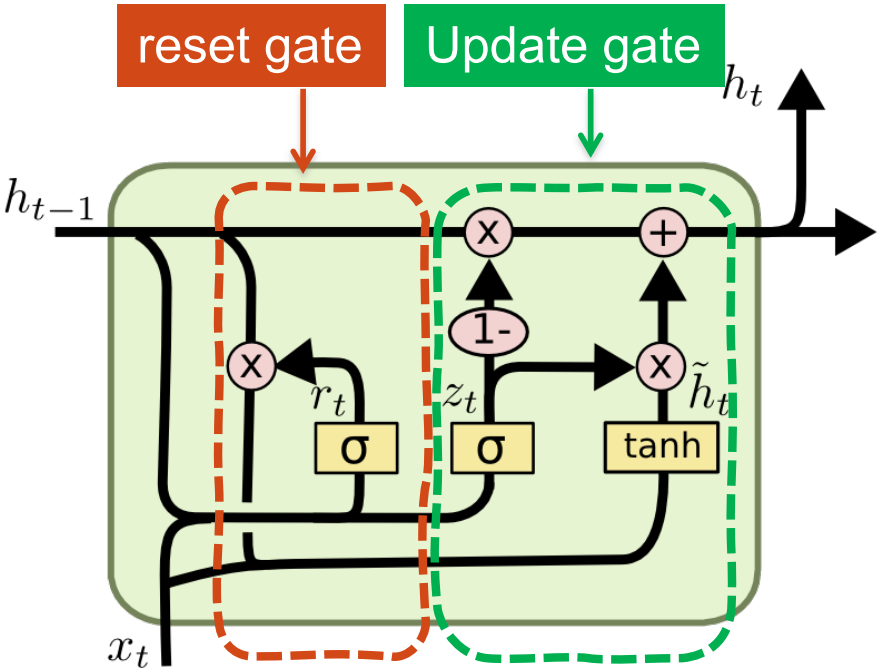
LSTM cell

- Cellule composée de trois « portes » : ce sont des zones de calcul qui régulent le flux d'informations :
  - Forget gate (porte d'oubli)
  - Input gate (porte d'entrée)
  - Output gate (porte de sortie)
- Cell state (état de la cellule)
  - Même function que 'residual'
  - Code une sorte d'agrégation des données de tous les pas de temps précédents qui ont été traités
- Hidden state (état caché)
  - N'a rien de cache… puisque c'est celui qui l'on observe
  - Code une sorte de caractérisation des données du pas de temps précédent.



GRU – gated recurrent unit

- GRU est plus simple que le LSTM.



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

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

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

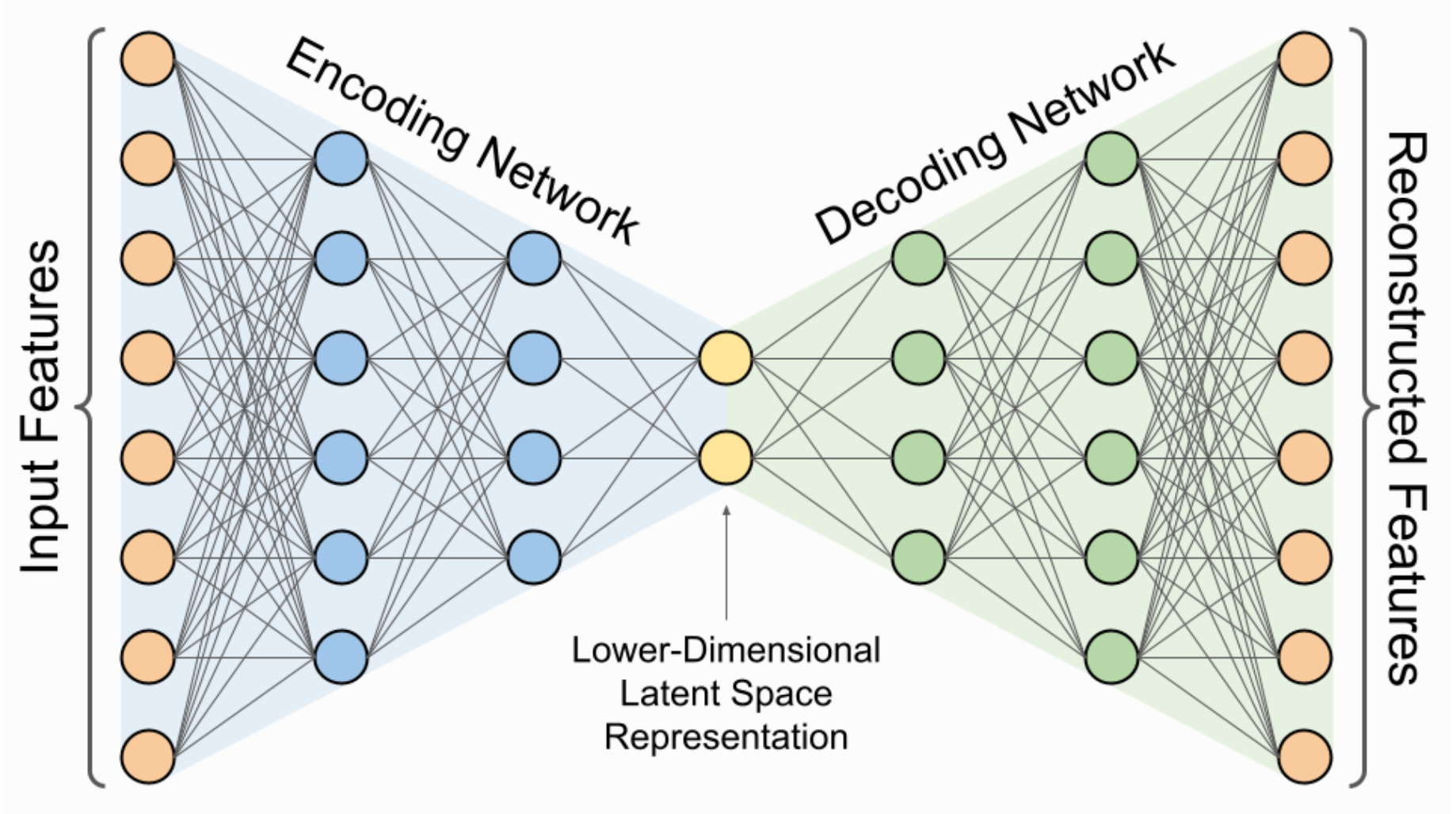
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- GRU combine l'oubli et l'entrée en une seule porte de mise à jour.
- GRU fusionne également l'état de la cellule et l'état caché.

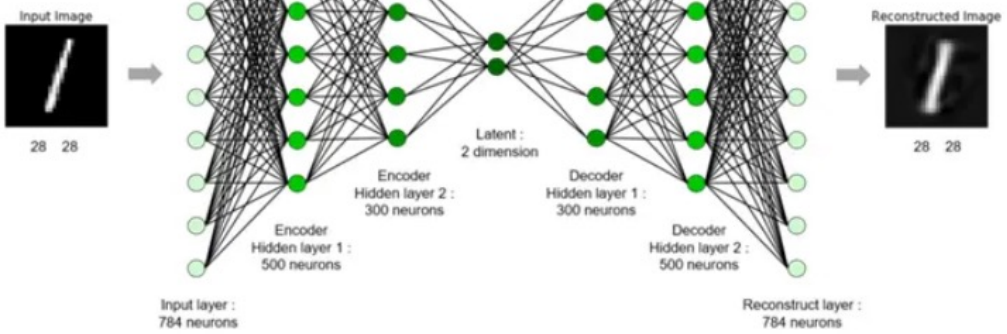
Architecture Seq2Seq

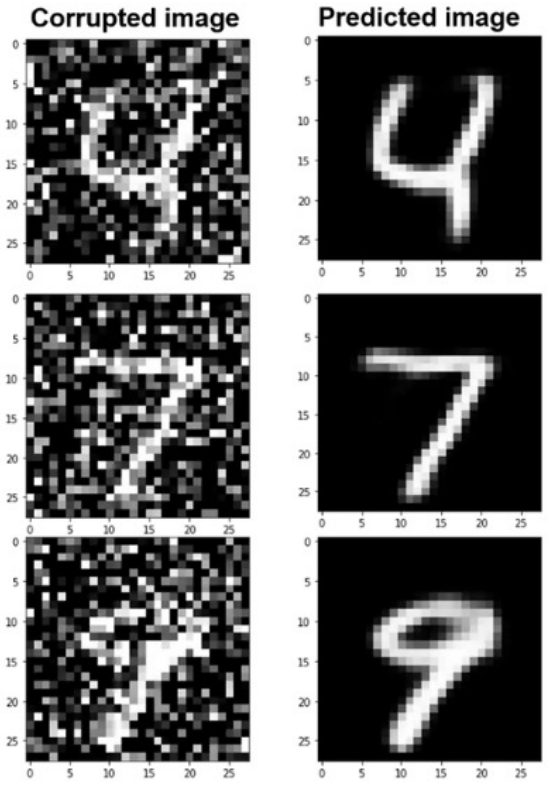
Auto-encodeur avec cellule récurrente

Architecture des auto-encodeurs

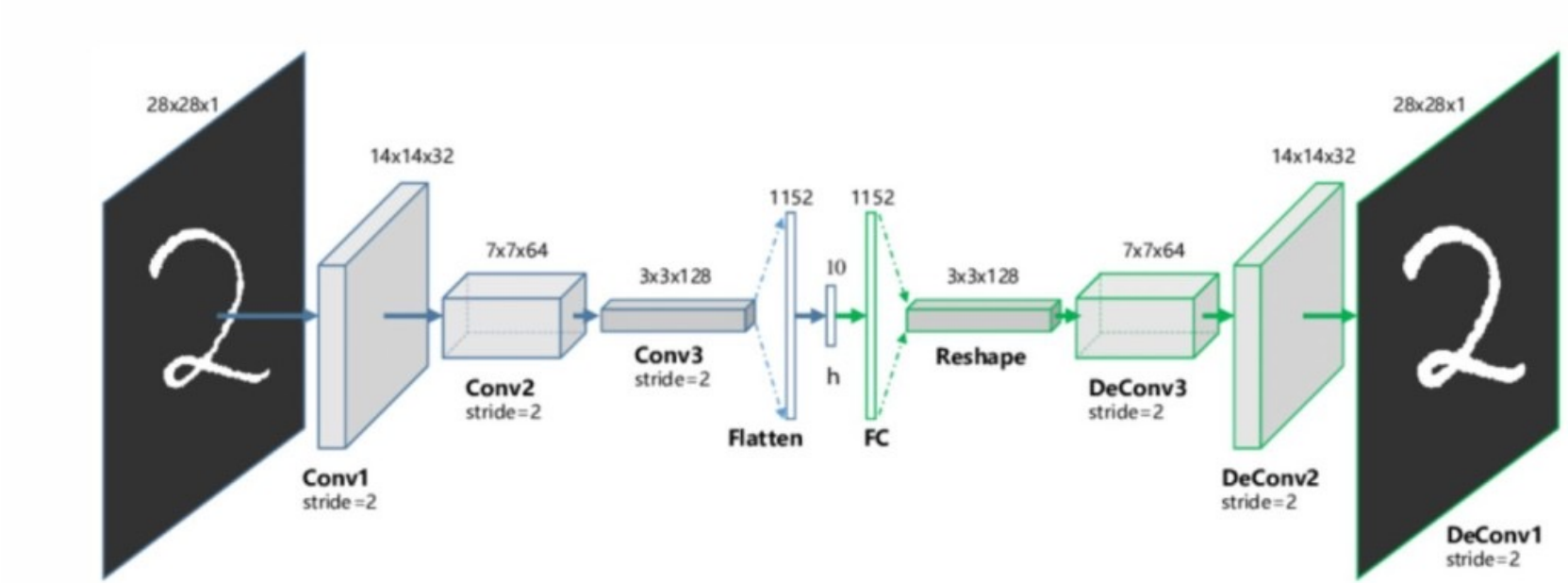


Exemple d'utilisation des auto-encodeurs



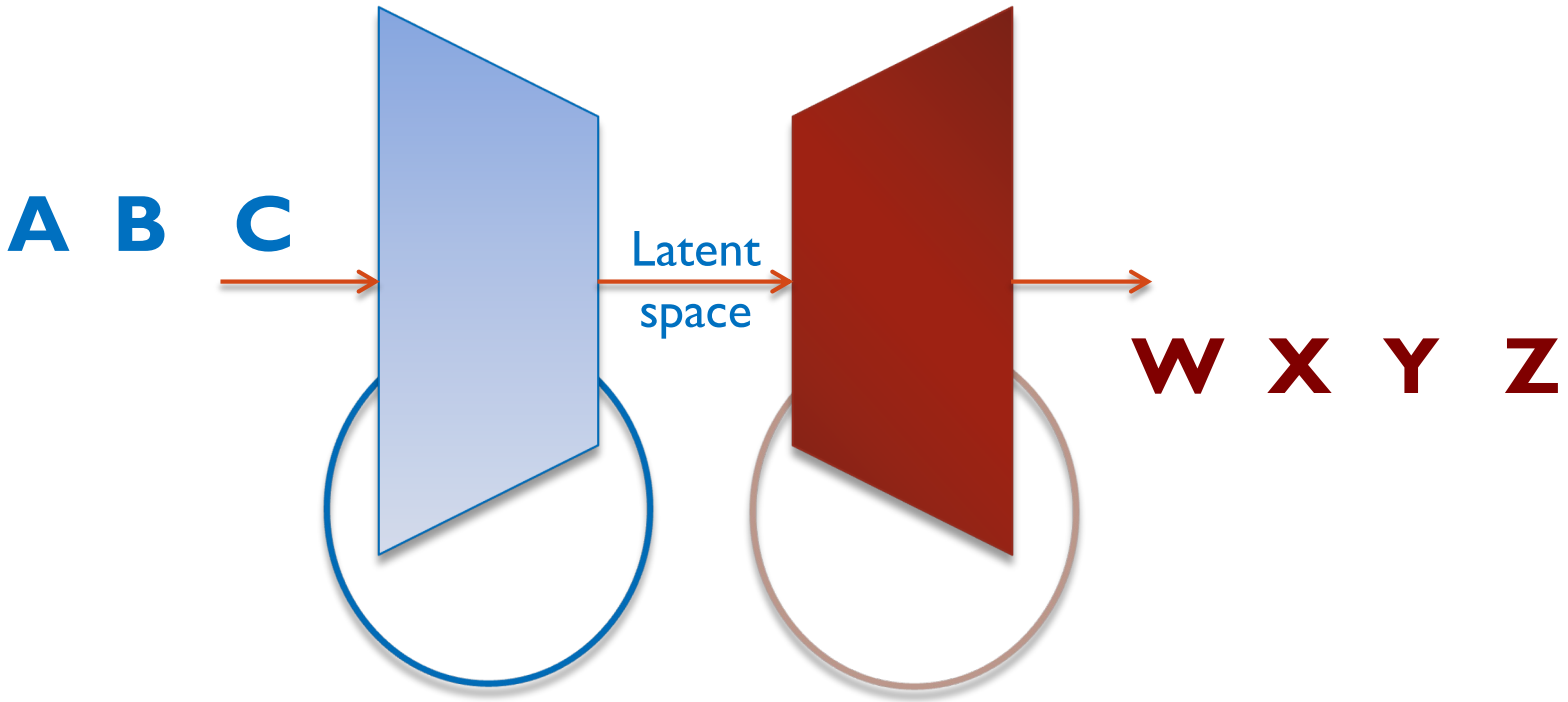


Auto-encodeurs avec des CNN



Auto-encoder avec des cellules récurrentes

- Étendre l'architecture du codeur-décodeur
  - à une séquence de données
  - afin de développer une architecture capable de générer des séquences de sortie de longueur arbitraire adaptées au contexte



Seq2Seq model

- Applications with text
  - Machine translation
  - Text summarization
  - Question answering
  - Dialogue modeling
- But also : forecasting
- Il est aussi possible de remplacer l'encoder par un au autre latent space :
  - Par exemple : Image captioning

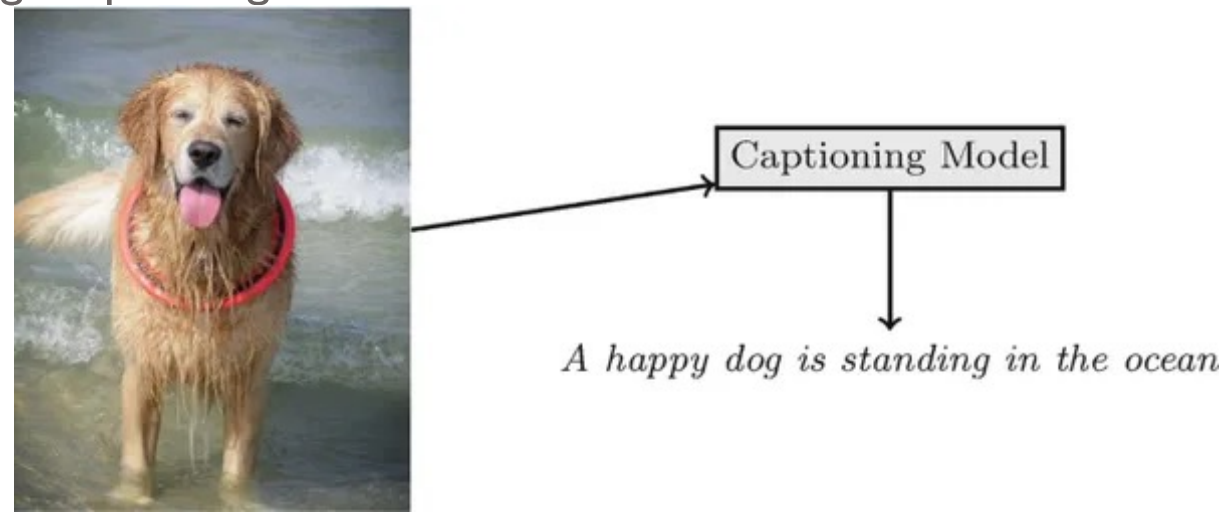
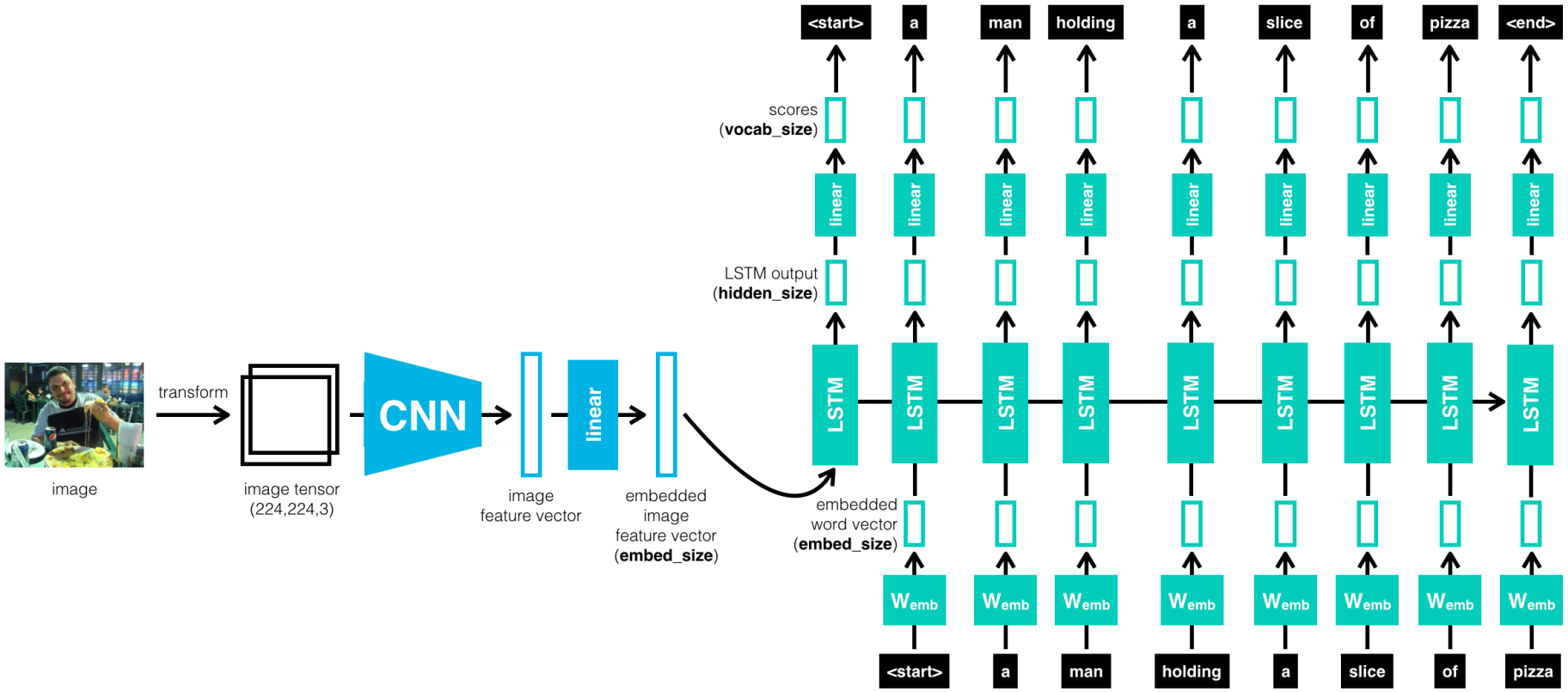
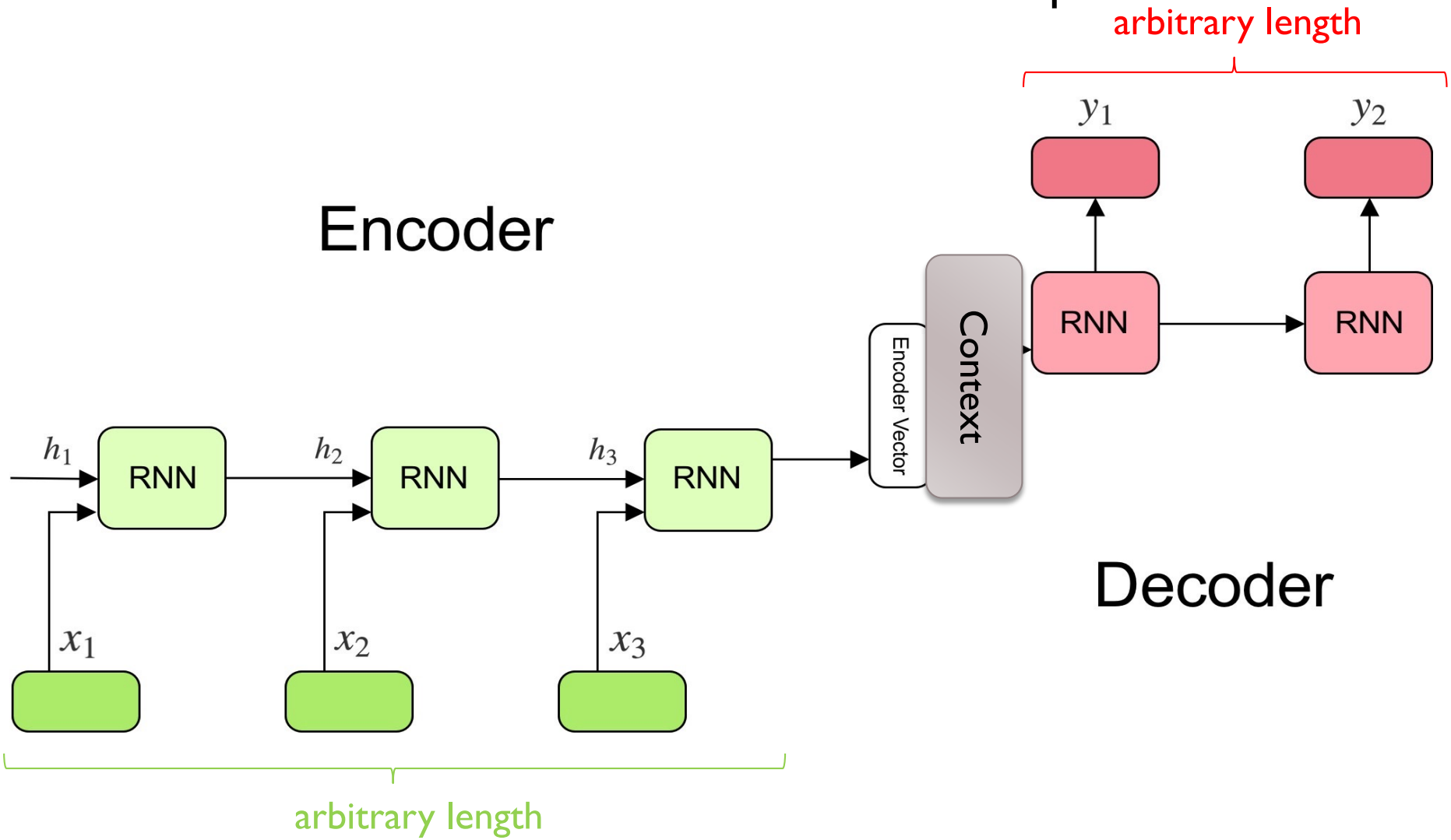


Image captioning model



Seq2seq – original approach

- Extend encoder-decoder architecture to a sequence data



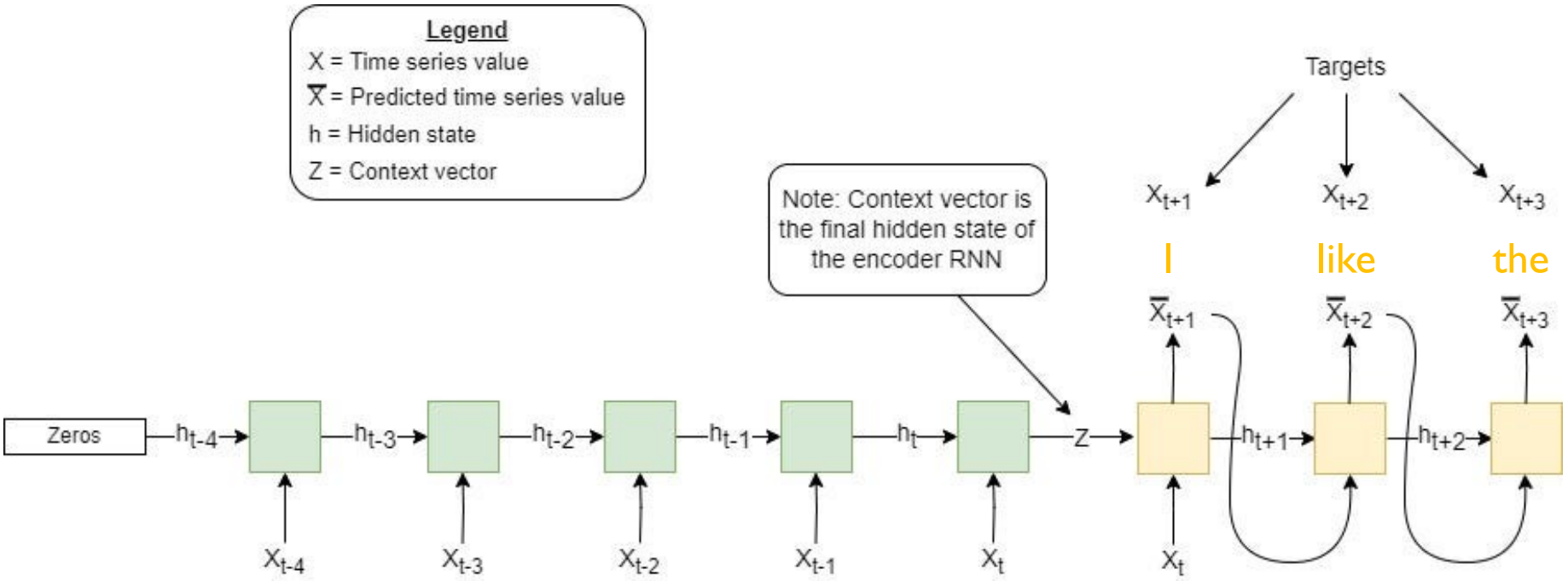
Sequence-to Sequence Architectures

- 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

Comment construire une architecture Seq2Seq

- Encodeur
- Context
- Décodeur : 2 approches possibles
  - Sans teacher : plus difficile à construire et plus difficile à faire converger
  - Avec teacher : approche aujourd'hui adoptée lors du TD

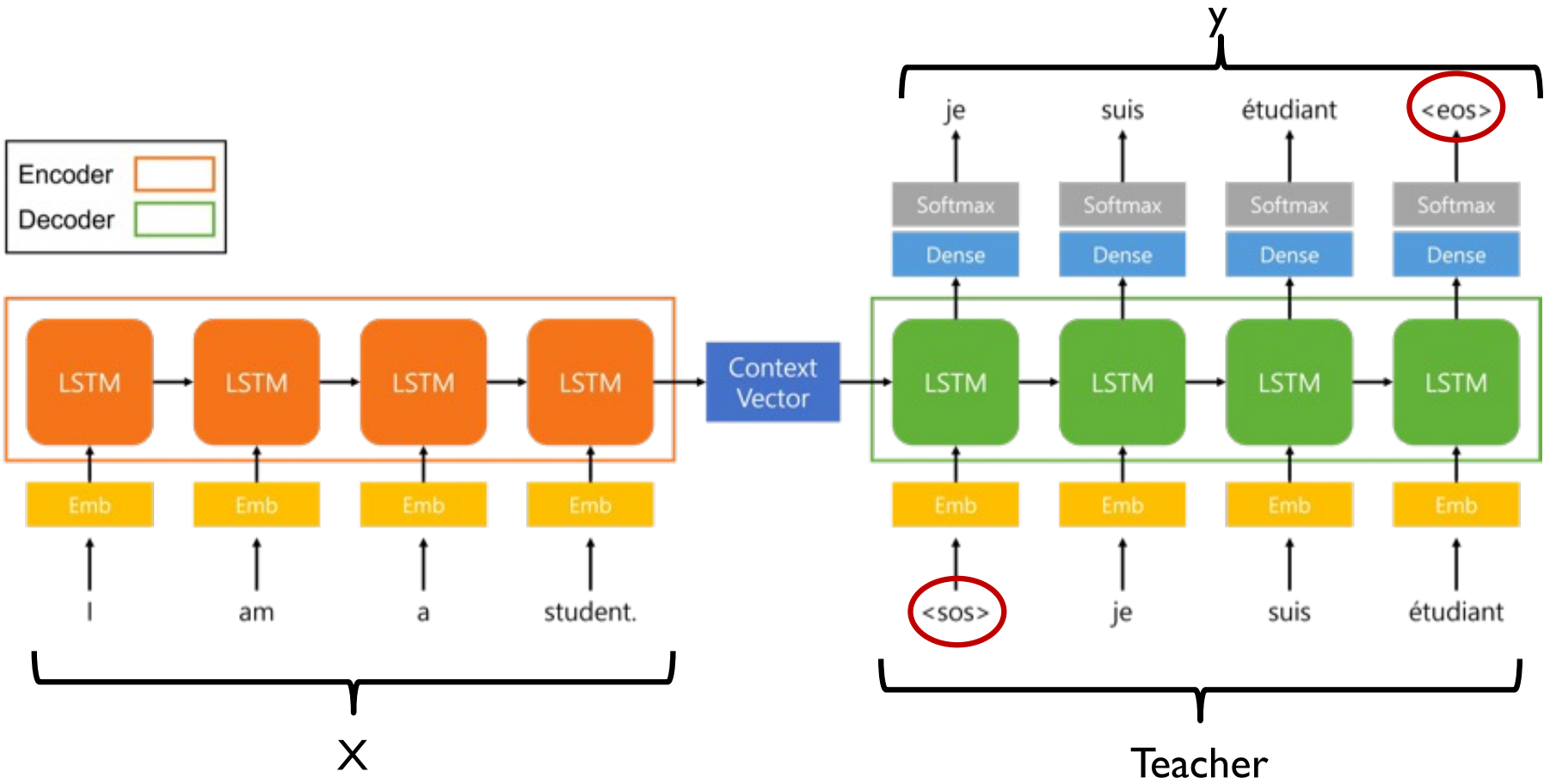
Seq2Seq sans teacher



si $\bar{X}_t \neq X_t$

→ l'erreur est propagée

Seq2Seq avec teacher



model.fit ([x, Teacher], y)

Le 'teacher' est construit à partir de y

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 literature.

Comment construire un modèle Seq2Seq avec 'teacher'

- Encoder

```
def build_encoder():
    inputs = layers.Input(shape=(_INPUT_LENGTH,), name="encInput")
    h = layers.Embedding(input_dim=_VOCAB_SIZE,
                         output_dim=_EMBEDDING_SIZE)(inputs)
    encoder_outputs, memory_state, carry_state = layers.LSTM(_RNN_SIZE,
                         return_sequences=True, return_state=True,
                         dropout=_DROPOUT_RATE, recurrent_dropout=_DROPOUT_RATE)(h)
    encoder_context = [memory_state, carry_state]

    return models.Model(inputs, [encoder_outputs, encoder_context], name="encoder")
```

Comment construire un modèle Seq2Seq avec 'teacher'

- Decoder

```
def build_decoder():
    teacher_inputs = layers.Input(shape=(_OUTPUT_LENGTH,), name="decInput")
    ctx__input = [layers.Input(shape=(_RNN_SIZE,), name="ctxInputH"),
                                    layers.Input(shape=(_RNN_SIZE,), name="ctxInputC")]
    h = layers.Embedding(input_dim=_VOCAB_SIZE,
                         output_dim=_EMBEDDING_SIZE)(teacher_inputs)
    decoder_outputs, memory_state, carry_state = layers.LSTM(_RNN_SIZE,
                         return_sequences=True, return_state=True,
                         dropout=_DROPOUT_RATE,
                         recurrent_dropout=_DROPOUT_RATE)(h, initial_state=ctx_input)
    context = [memory_state, carry_state]
    outputs = layers.Dense(_VOCAB_SIZE, activation='softmax')(decoder_outputs)

    return models.Model([teacher_inputs, context_input], [outputs, context], name="decoder")
```

Comment construire un modèle Seq2Seq avec 'teacher'

- Seq2Seq = Encoder+Decoder

```
def build_seq2seq(encoder, decoder):
    enc_inputs = layers.Input(shape=(_INPUT_LENGTH,), name="encInput")
    _, enc_context = encoder(enc_inputs)

    teacher_inputs = layers.Input(shape=(_OUTPUT_LENGTH,), name="teacherInput")
    dec_sequence_outputs, _ = decoder([teacher_inputs, enc_context])

    return models.Model([enc_inputs, teacher_inputs], [dec_sequence_outputs])
```

Comment prédire avec une architecture Seq2Seq

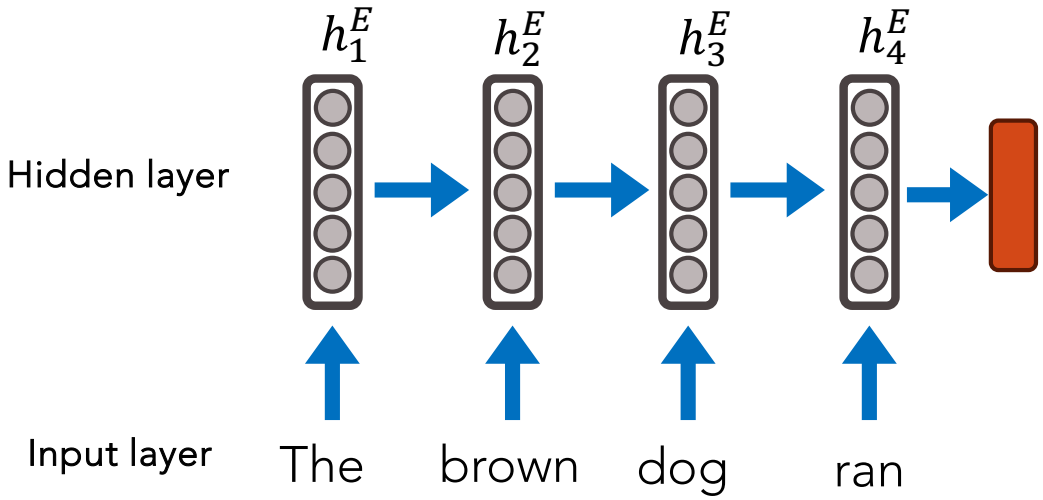
- On est obligé de procéder par étape (approche gourmande)
  - On ne peut pas utiliser de teacher… car on ne connait pas la phrase à décoder

- Le mode d'emploi est donc le suivant :
  1. Encoder la phrase
  2. Itération pour décoder successivement chaque pas de temps, en réutilisant les pas de temps déjà décodés
  3. Arrêt lorsque <fin> ou max_length

- Concrètement, il y a 3 manières de faire
  - De la plus simple et la plus inefficace
    - On fait plusieurs fois la même chose : encodage + decodage à nouveau de la partie déjà décodée
  - A la plus efficace
    - On fait une seule fois chaque étape : encodage / décodage du prochain mot en utilisant le mot que l'on vient de décoder
    - Détails dans les transparents suivants
  - En passant
    - On fait une fois l'encodage, mais on décode plusieurs fois la séquence en utilisant chaque fois, l'ensemble des étapes précédentes

- **A approfondir lors du TD**

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



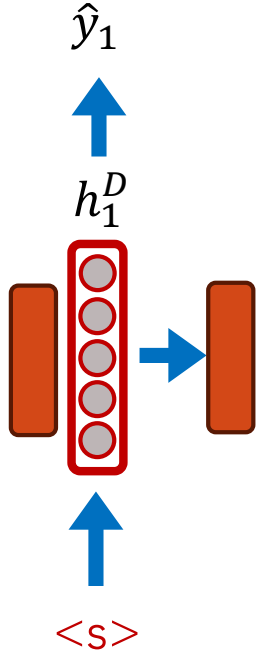
ENCODER RNN

DECODER RNN

Predict with seq2seq model
Step 2 : decode the first output

Decode step by step

Reuse at each step, the previous output

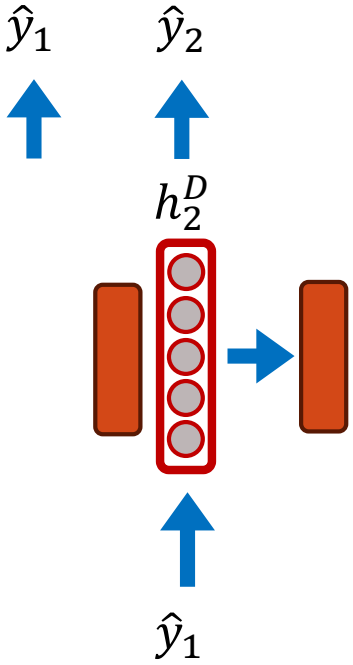


DECODER RNN

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

Decode step by step

Reuse at each step, the previous output



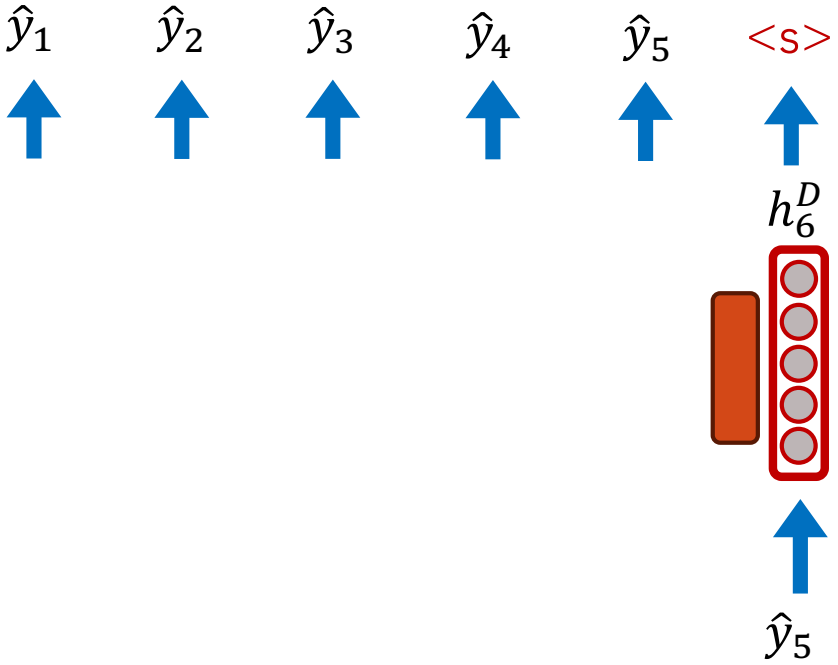
DECODER RNN

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 or max_length



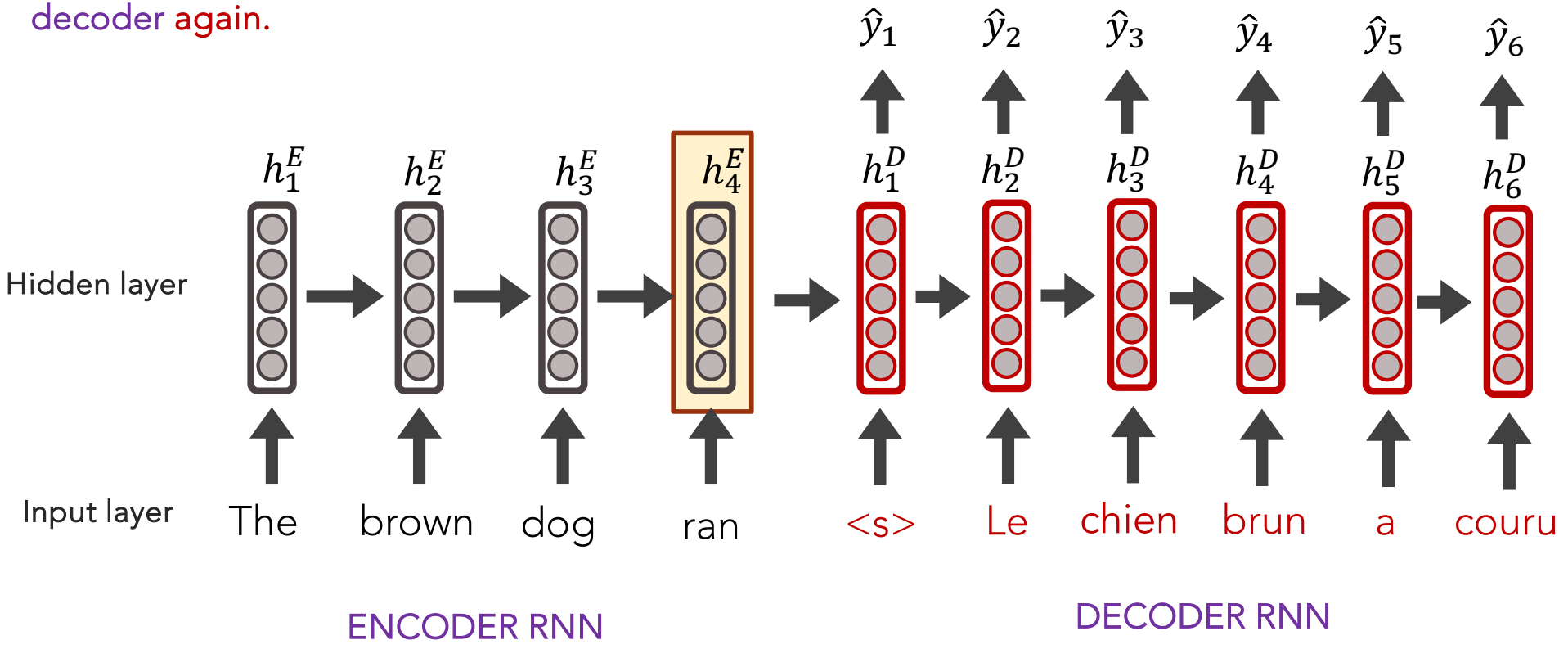
DECODER RNN

On améliore la proposition Seq2Seq

Ajout du mécanisme d'attention

Sequence-to-Sequence (seq2seq)

It's crazy that the entire "meaning" of sentence is expected to be packed into this one embedding, and that the encoder then never interacts w/ the decoder again.

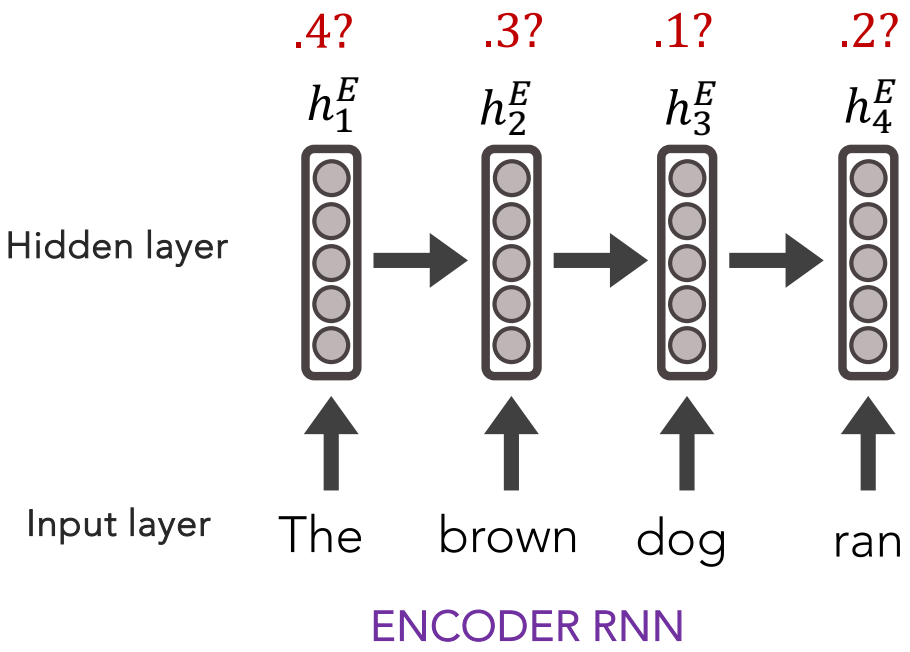


Les limites de l'approche Seq2Seq

- L'encodeur lit la phrase entière **une seule fois**
  - En produit une représentation abstraite (un vecteur)
  - Il doit donc se souvenir de toutes les parties de la phrase
  - Le décodeur utilise cette représentation pour convertir cette phrase
- Pour des phrases longues
  - le codeur ne peut pas se souvenir de la première partie de la phrase, ce qui entraînera une perte d'informations.
- Comment traduisez-vous un texte ?
  - Pensez-vous que l'ensemble de la séquence d'entrée (ou de la phrase) est important à chaque étape de l'encodage ?
  - Pouvons-nous mettre l'accent sur certains mots plutôt que d'accorder la même importance à tous les mots ?
  - Nous, les humains, on traduit chaque mot en nous concentrant principalement sur certains mots de l'entrée.
  - Idéalement, à chaque étape, on ne devrait transmettre que les informations pertinentes
    - Encodages des informations pertinentes
    - Transmission au décodeur pour la traduction
- Le mécanisme d'attention a été développé pour répondre à ces défis.

seq2seq + Attention

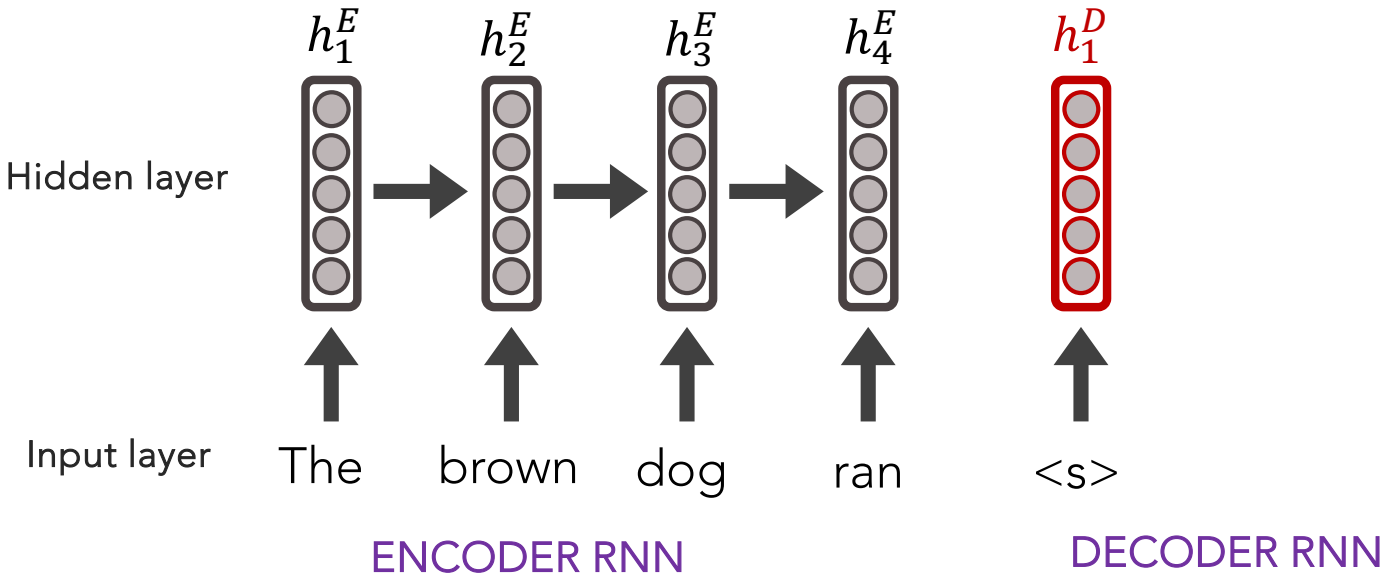
Q: How do we determine how much to pay attention to each of the encoder's hidden layers?



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

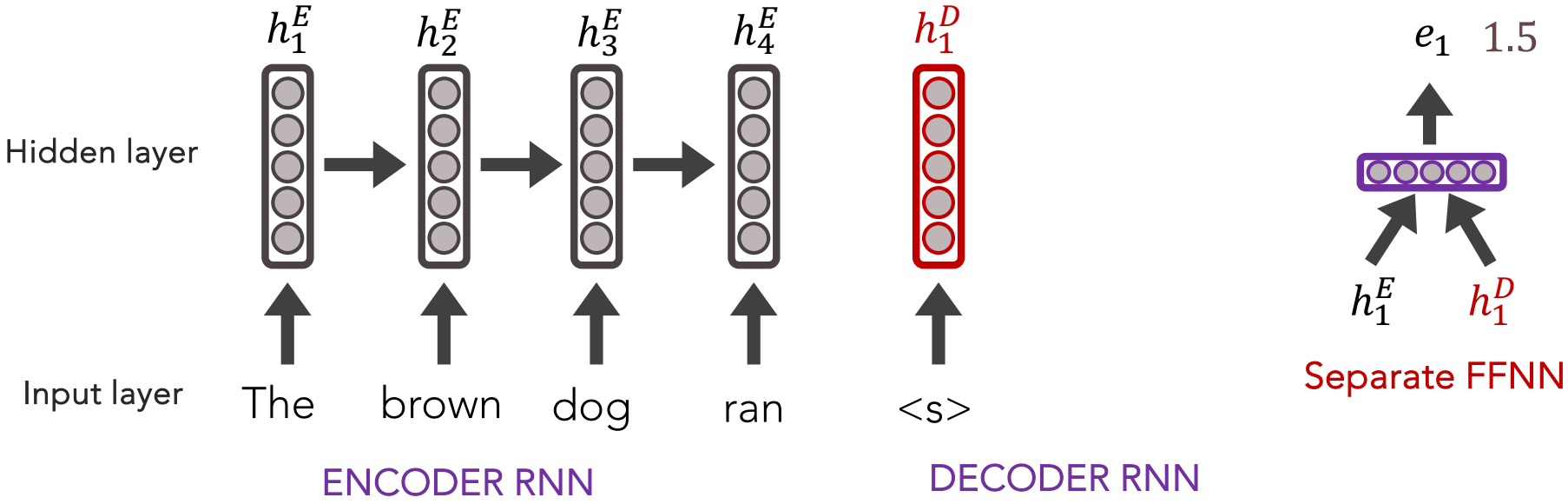
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

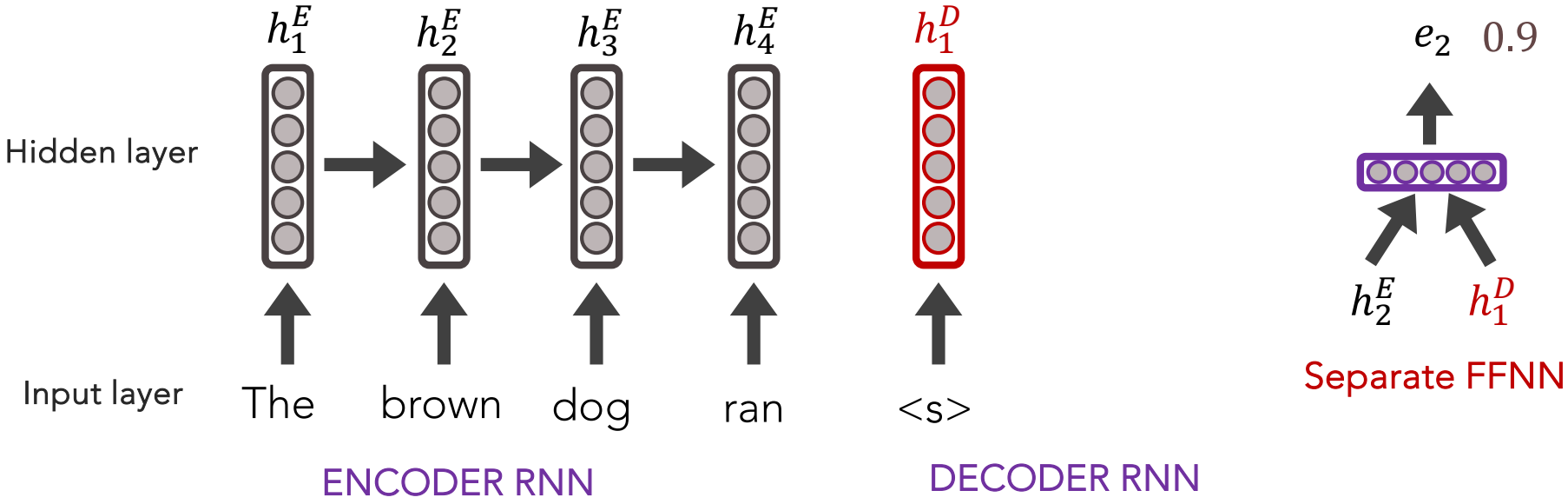
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

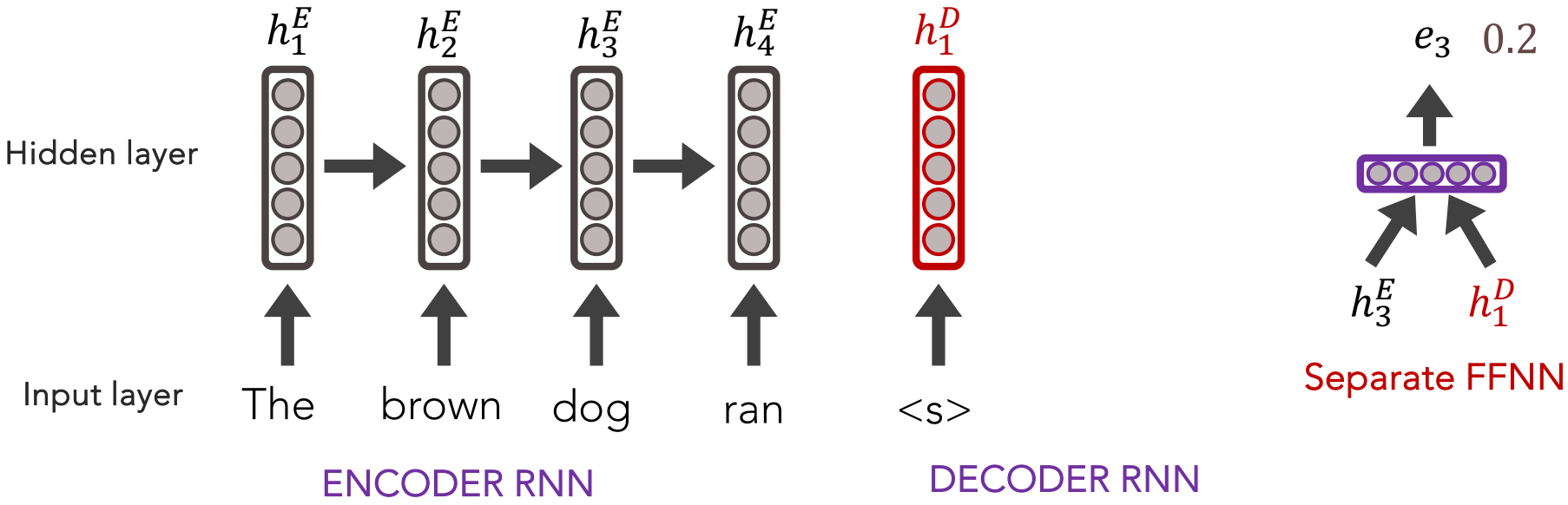
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

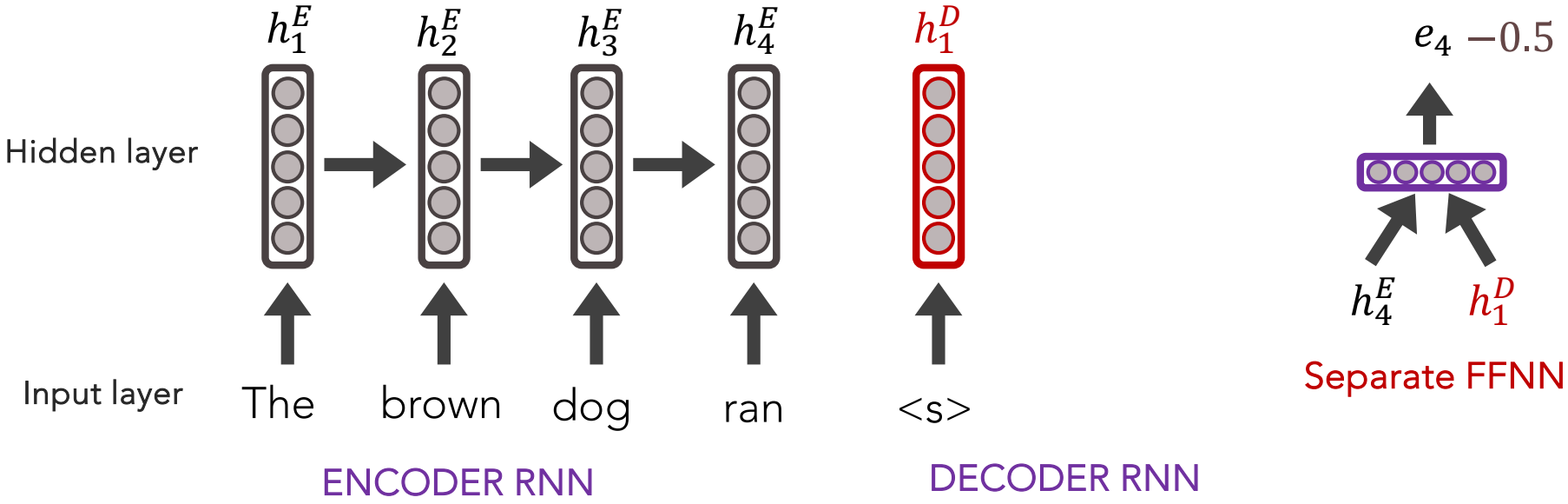
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

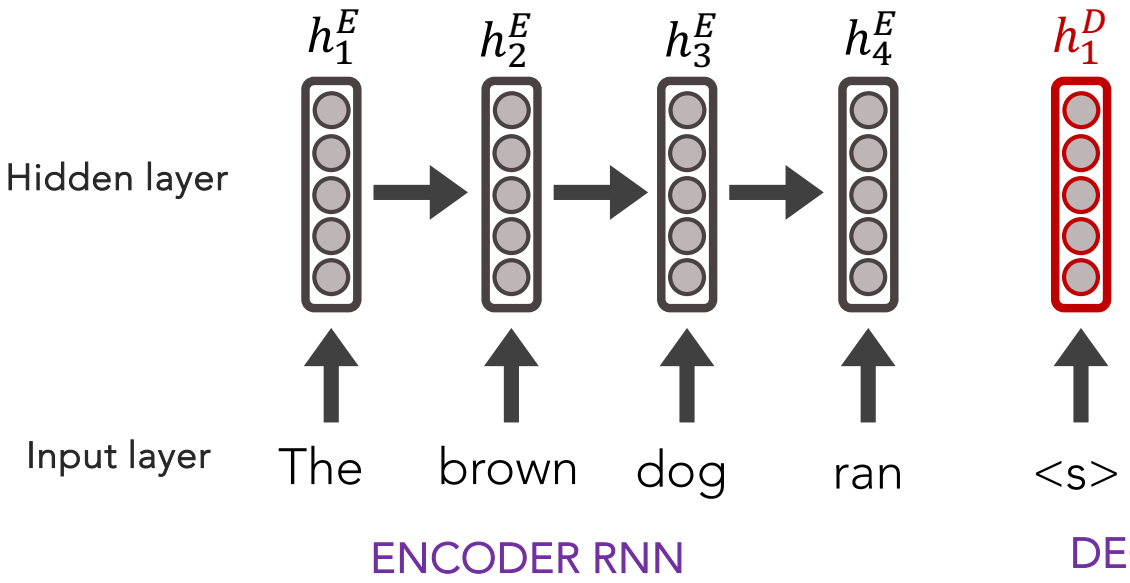
A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



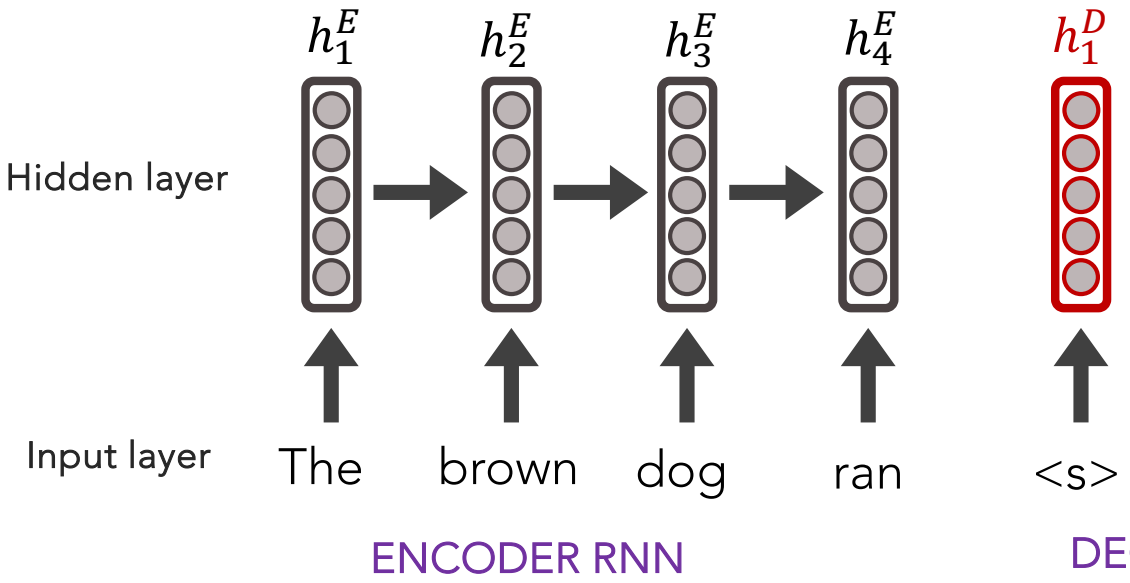
Attention (raw scores)

e_1 1.5
e_2 0.9
e_3 0.2
e_4 −0.5

seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



Attention (raw scores)

$$e_1 \quad 1.5$$
$$e_2 \quad 0.9$$
$$e_3 \quad 0.2$$
$$e_4 \quad -0.5$$

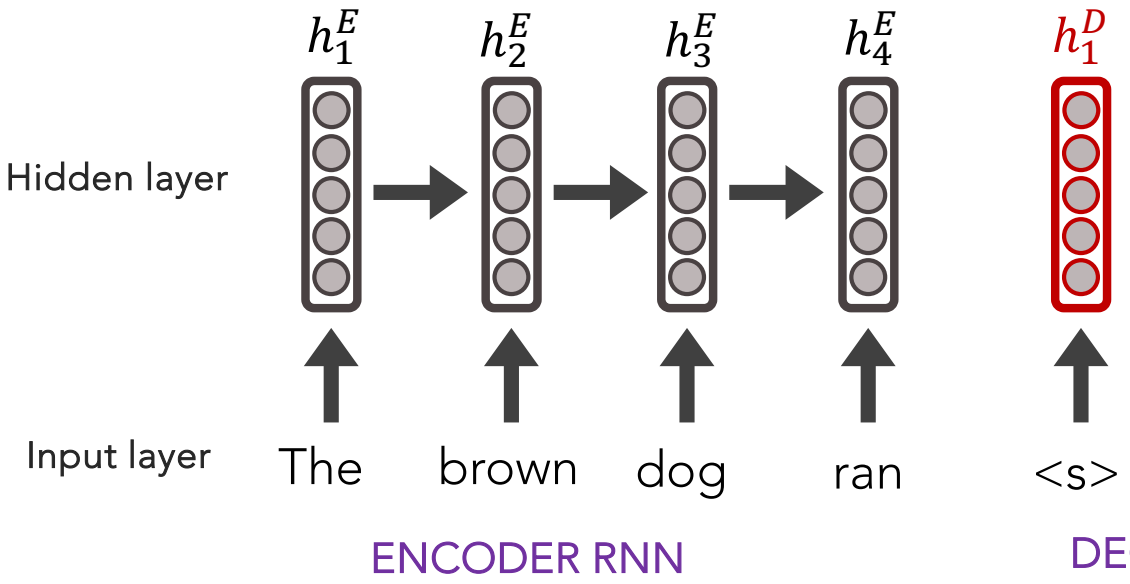
Attention (softmax'd)

$$a_i^1 = \frac{\exp(e_i)}{\sum_i^N \exp(e_1)}$$

seq2seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's current hidden state (our current representation of meaning) and all of the encoder's hidden layers!



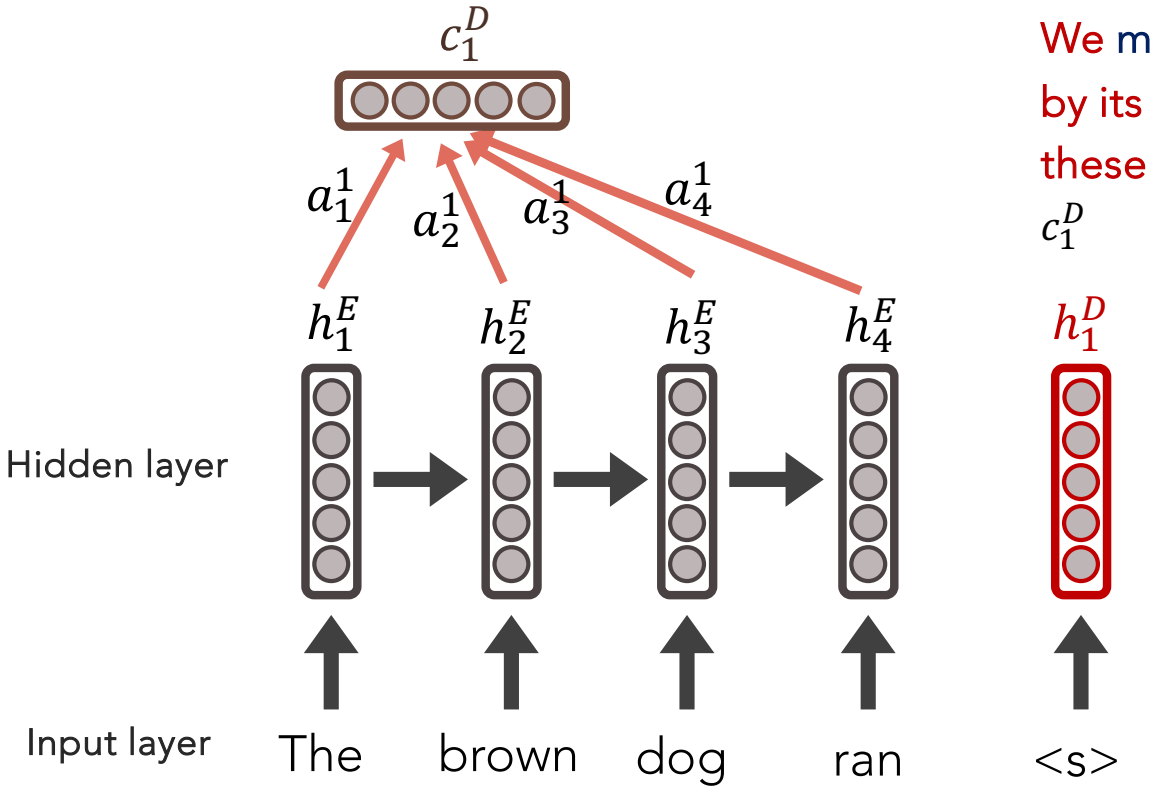
Attention (raw scores)

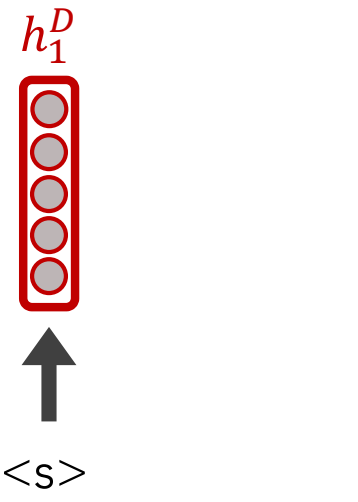
e_1 1.5
e_2 0.9
e_3 0.2
e_4 −0.5

Attention (softmax'd)

$a_1^1 = 0.51$
$a_2^1 = 0.28$
$a_3^1 = 0.14$
$a_3^1 = 0.07$

seq2seq + Attention





We multiply each encoder's hidden layer by its a_i^1 attention weights and sum all these vectors to create a context vector c_1^D

Attention (softmax'd)

$a_1^1 = 0.51$

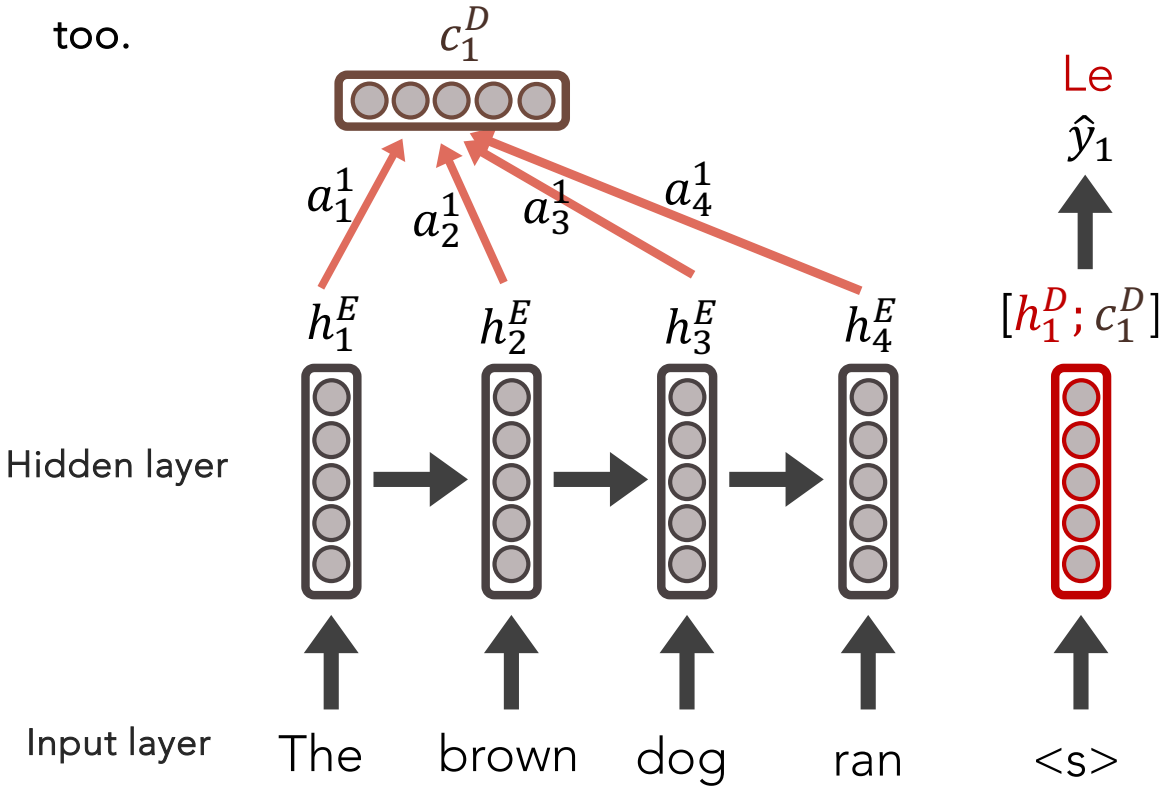
$a_2^1 = 0.28$

$a_3^1 = 0.14$

$a_3^1 = 0.07$

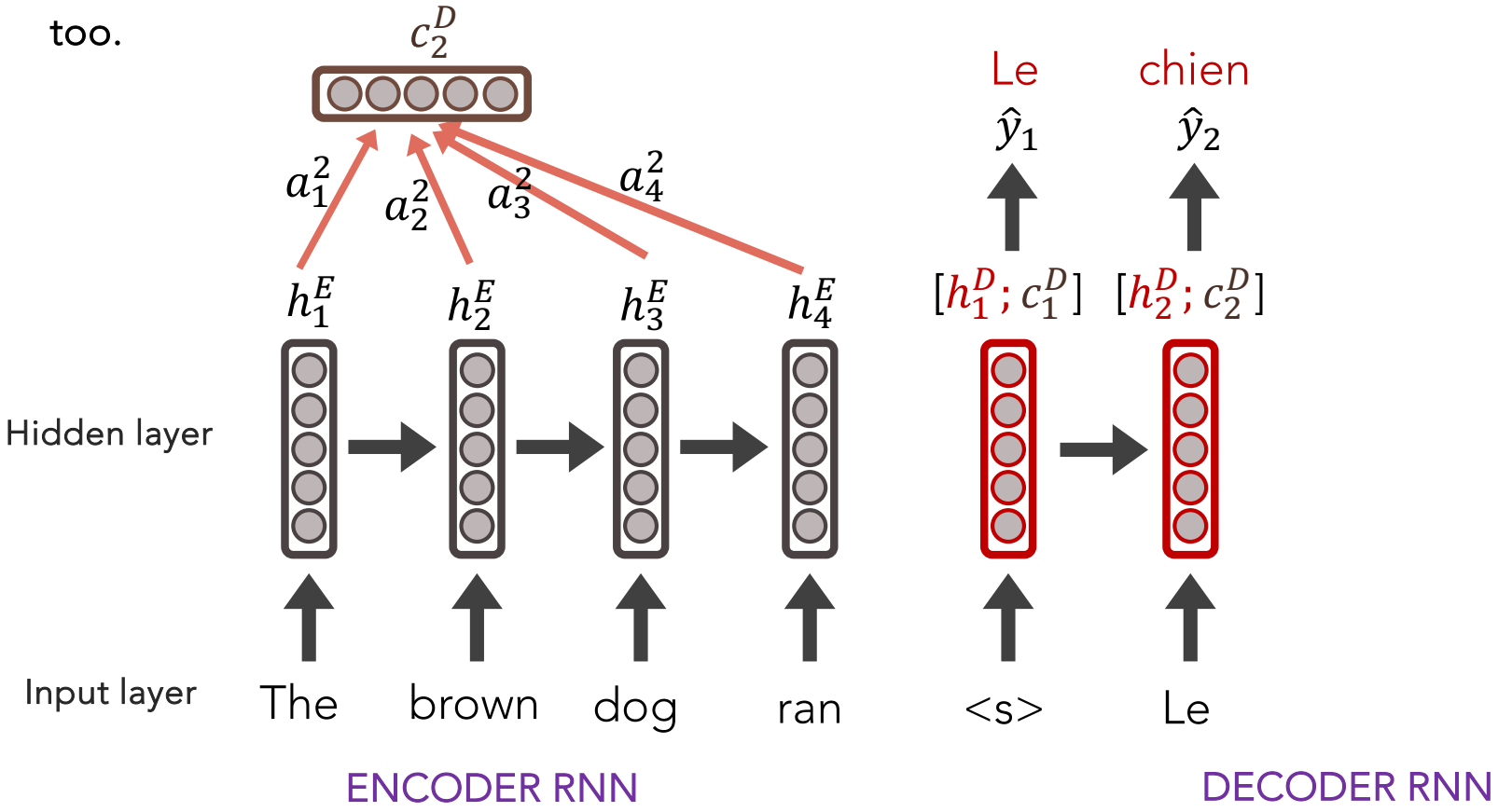
seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



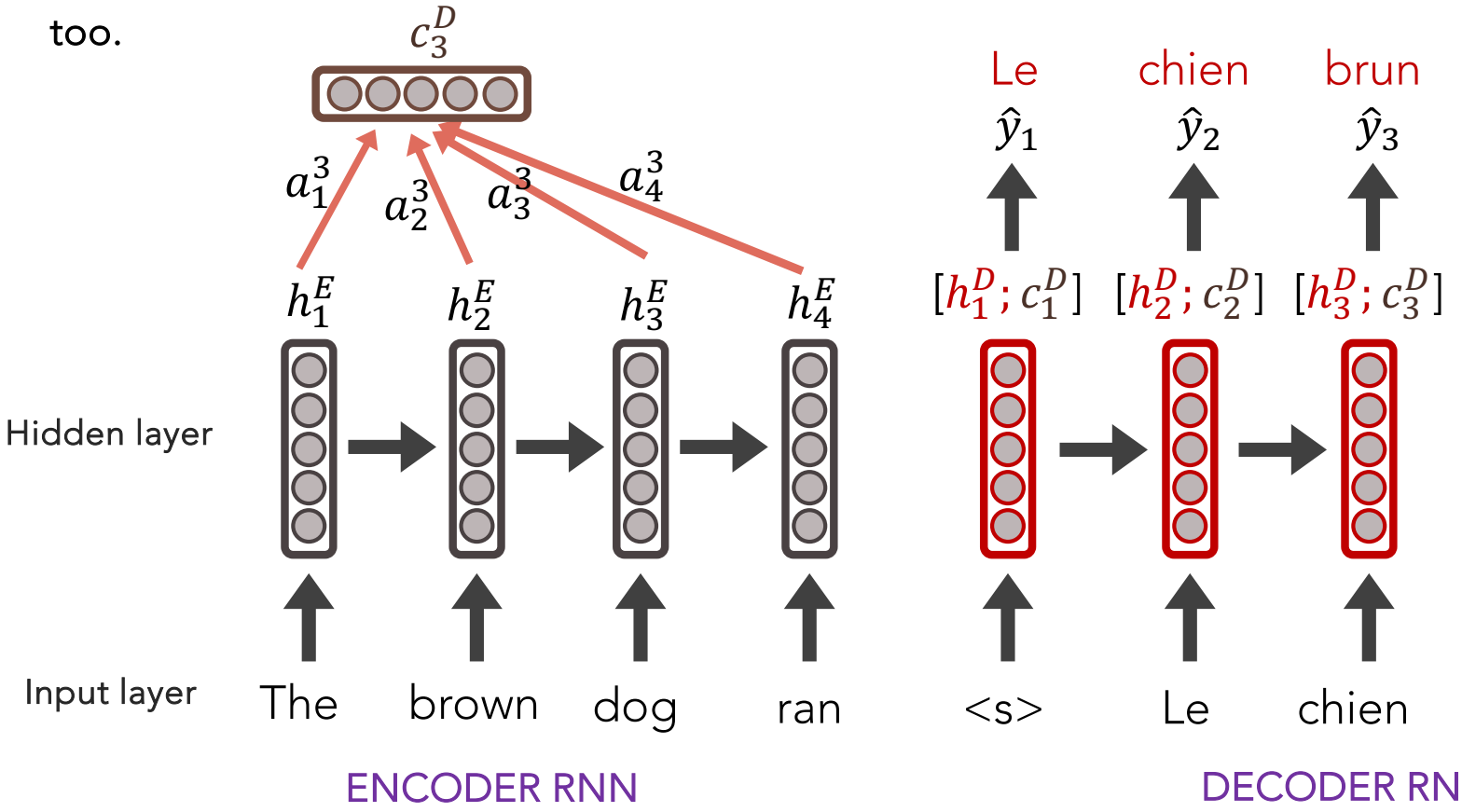
seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



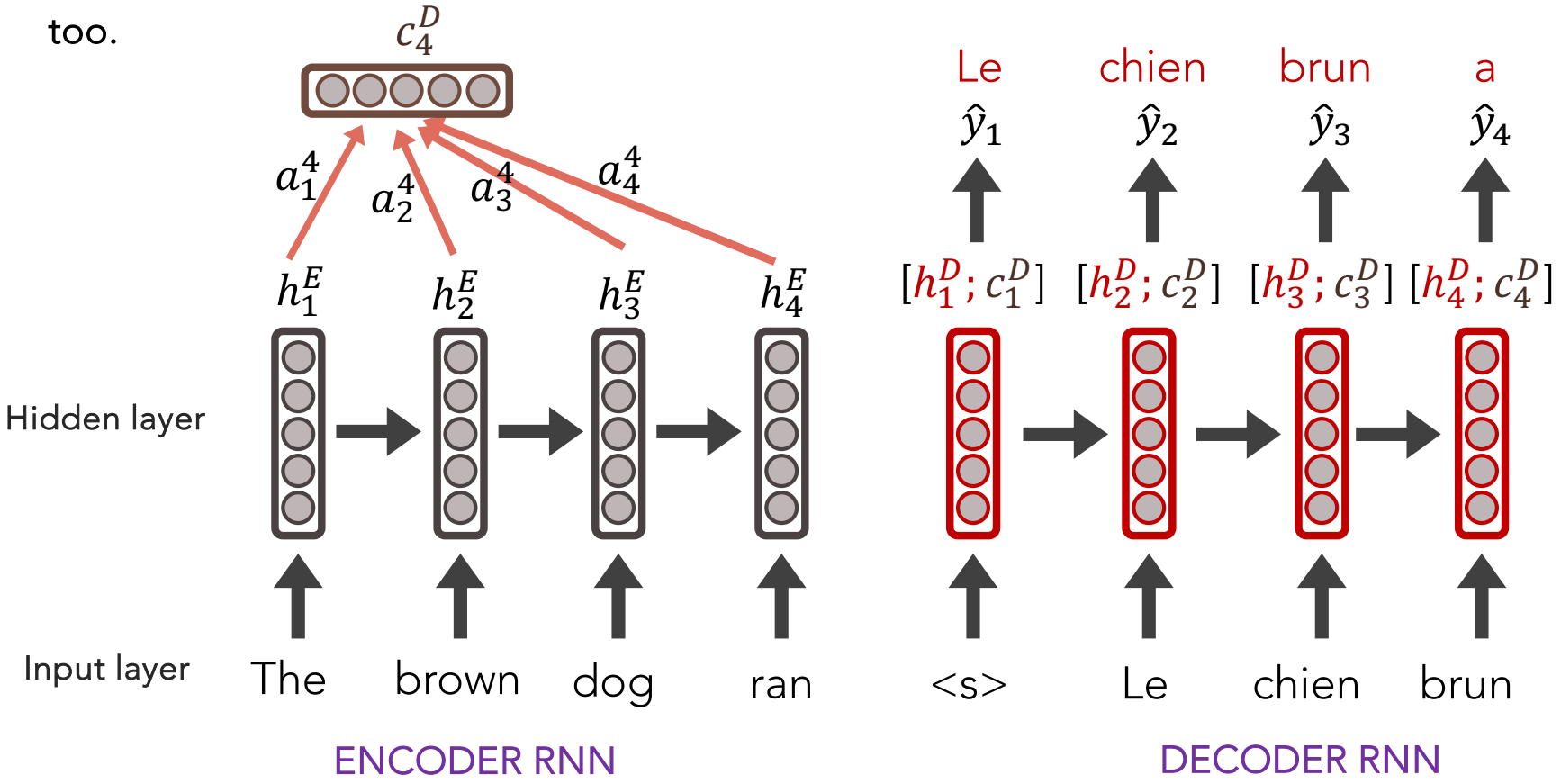
seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



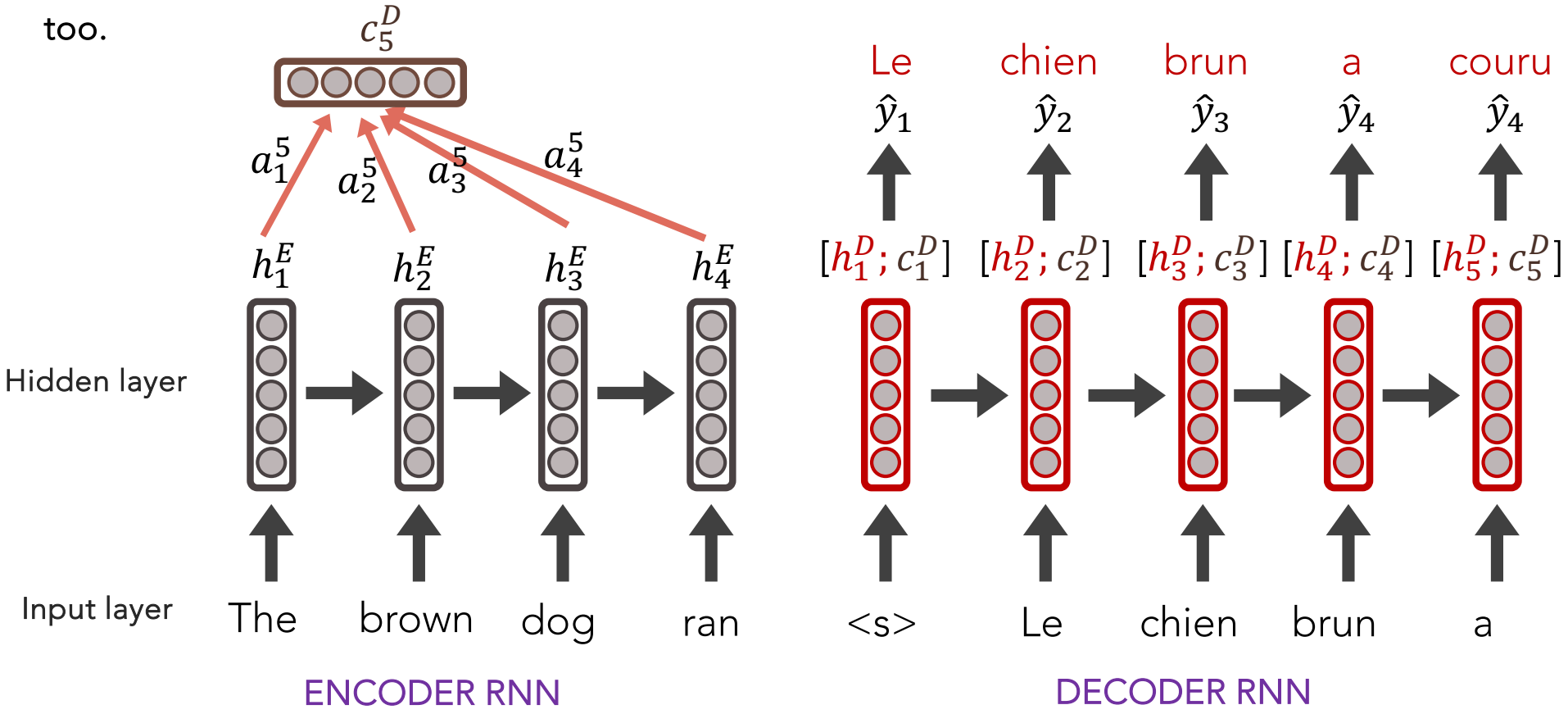
seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



seq2seq + Attention

REMEMBER: each attention weight a_i^j is based on the decoder's current hidden state, too.



For convenience, Attention calculation summarized on 1 slide

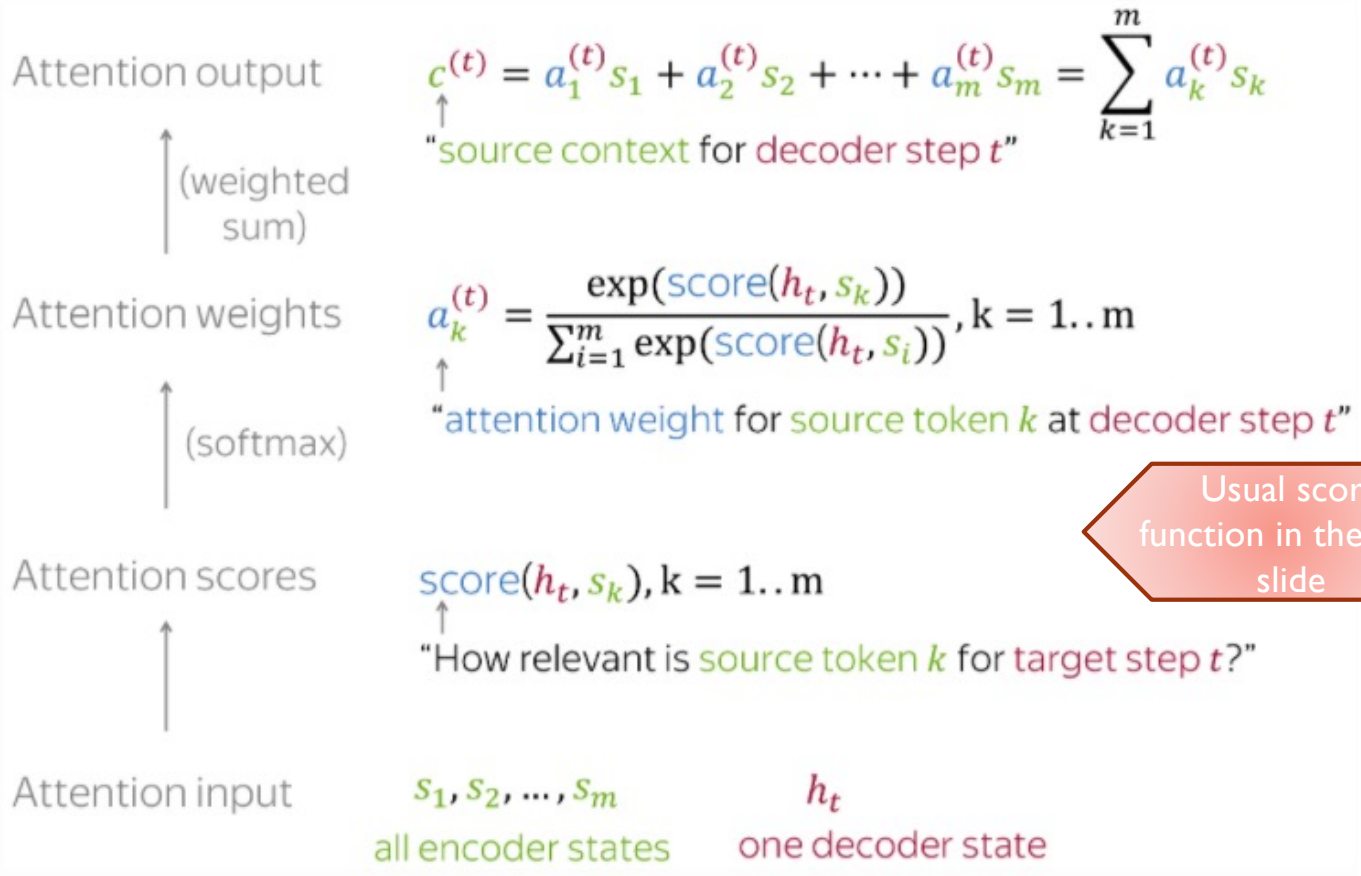


Photo credit: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Les principales fonctions 'Score'

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Il nous reste à construire le réseau

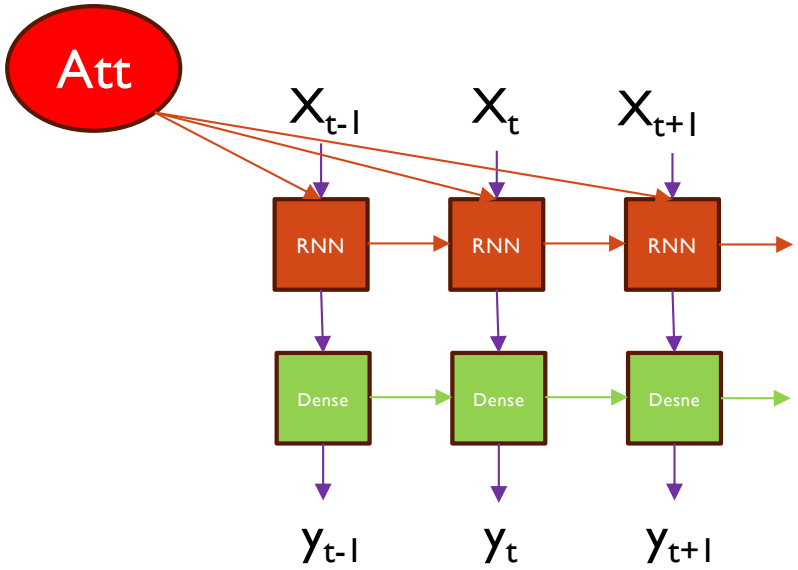
- Encodeur → inchangé
- Decodeur → il faut calculer l'attention

```
def build_decoder_attention():
    teacher_inputs = layers.Input(shape=(_OUTPUT_LENGTH,), name="decInput")
    ctx_input = [layers.Input(shape=(_RNN_SIZE,), name="ctxInputH"),
                 layers.Input(shape=(_RNN_SIZE,), name="ctxInputC")]
    encoder_outputs = layers.Input(shape=(_INPUT_LENGTH, _RNN_SIZE))

    h = layers.Embedding(input_dim=_VOCAB_SIZE, output_dim=_EMBEDDING_SIZE)(teacher_inputs)
    decoder_outputs, memory_state, carry_state = layers.LSTM(_RNN_SIZE, return_sequences=True,
            return_state=True,
            dropout=_DROPOUT_RATE, recurrent_dropout=_DROPOUT_RATE)(h, initial_state=ctx_input)
    decoder_context = [memory_state, carry_state]

    attention_context, attention_scores = attention(decoder_outputs, encoder_outputs)
    decoder_outputs = layers.concatenate([attention_context, decoder_outputs])

    outputs = layers.Dense(_VOCAB_SIZE, activation='softmax')(decoder_outputs)
    return models.Model([encoder_outputs, teacher_inputs, context_input],
                                        [outputs, decoder_context, attention_scores], name="decoder")
```

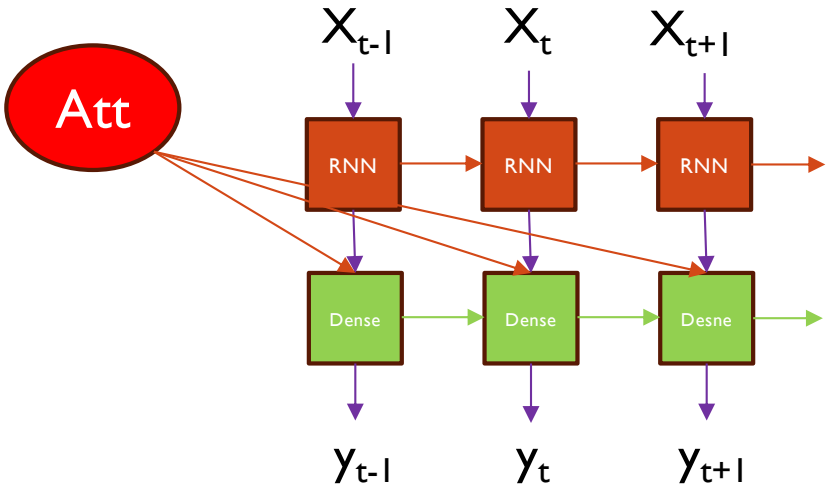
- Seq2Seq → inchangé

Il nous reste à construire le réseau

- Encodeur → inchangé
- Decodeur
  - → il faut calculer l'attention
  - → il faut intégrer l'attention
    - Solution 1



Solution 2 : présentée dans les précédents transparents
On l'utilisera lors de TD (plus simple à impléter)



- Seq2Seq → inchangé

48

Calcul de l'attention

- Dépend évidemment de la fonction score
- def attention(query, value):
  - Query → vient du decoder
  - Value → vient de l'encoder

- Par exemple pour 'dot-product'

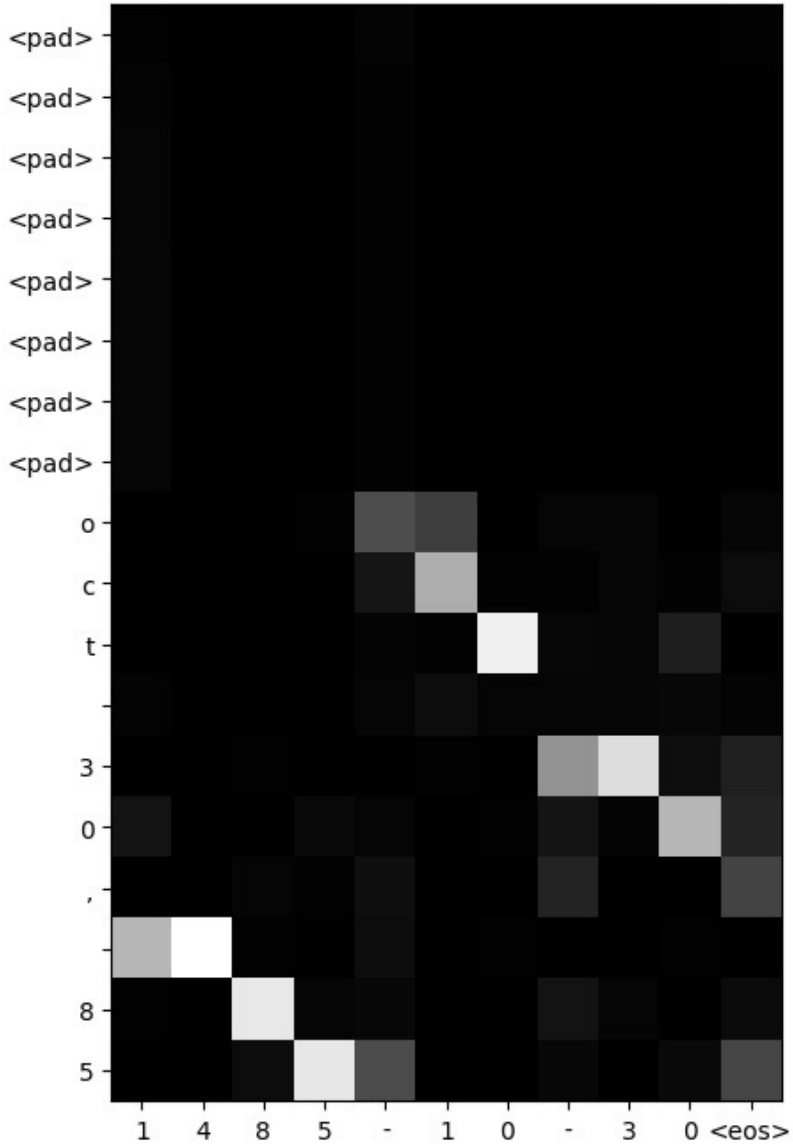
```
attention = layers.dot([query, value], axes=[2,2], name="Att_score")
attention_scores = layers.Activation('softmax', name="Att_weight")(attention)
attention_context = layers.dot([attention_scores, value], axes=[2,1])
return attention_context, attention_scores
```

- Ou utilisation du layer Attention de Keras

```
attention_context, attention_scores = layers.Attention()(inputs=[query, value],
                                                         return_attention_scores=True)
```

Attention : ne fonctionne pas sur toutes les plate-formes, vous pouvez essayer un autre modèle d'attention disponible dans Keras

Afficher de la matrice d'attention



Possible uniquement si on a les scores d'attention

Lab of the day

- Tache : traduction
  - Des dates dans différents formats → date dans un seul format
  - On travaille au niveau des 'caractères'

- Partie 1 : on construit le jeu de données
  - X, y et teacher
- Partie 2 : on construit, entraine et prédit avec un modèle Seq2Seq
  - Encodeur
  - Decodeur
  - On assemble Encodeur et decodeur pour construire le modèle complet
  - On entraine le réseau avec ealy stopping / On babysit
  - On prédit
- Partie 3 : Seq2Seq avec attention
  - On ajoute un calcul d'attention
  - On refait les mêmes étapes que précédemment
  - In fine, on peut afficher la matrice d'attention pour un exemple donné

- Les éléments pour construire le réseau de neurones, l'entrainer et prédire sont dans les transparents