Neural Machine Translation

CSC401/2511 A2 Tutorial 2
Winter 2023

Thanks to Julia Watson

Outline

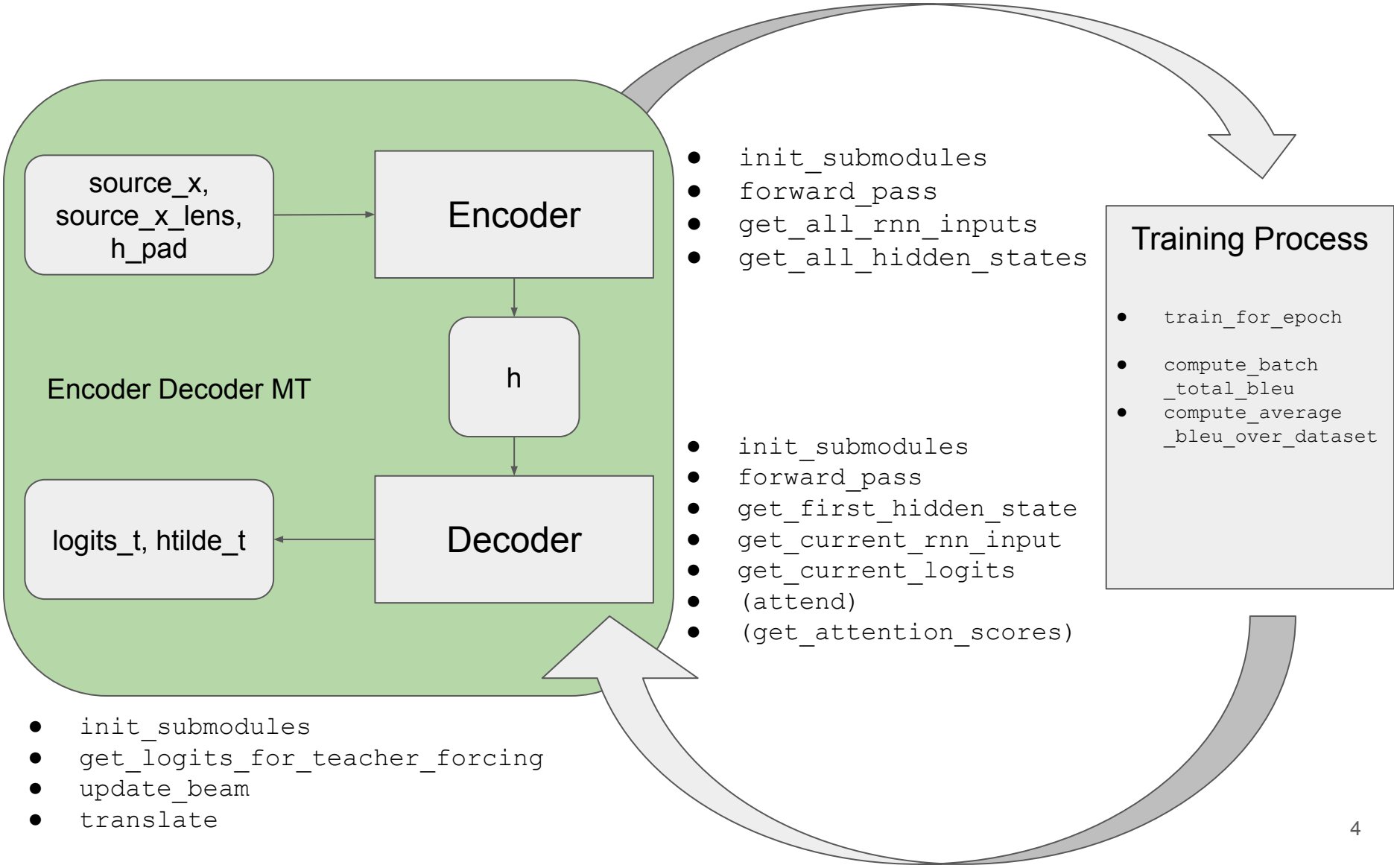
1. Walkthrough of the assignment
   a. Overview
   b. Calculating BLEU scores
   c. Encoder
   d. DecoderWithoutAttention
   e. DecoderWithAttention
   f. DecoderWithMultiHeadAttention
   g. Putting it together: EncoderDecoder
   h. Training and testing loop
2. Demo: interacting with the model
3. Q&A

Most of the material is covered in the SMT lecture

Page number to the slides: [pXX]

Overview: Canadian Hansards

Overview: The Assignment



Calculating BLEU scores

```
reference = ['friendship', 'is', 'magic']

candidate = ['friendship', 'is', ‘witchcraft’]
```

Calculating BLEU scores

```
reference = ['friendship', 'is', 'magic']

candidate = ['friendship', 'is', ‘witchcraft’]
```

grouper	>>> grouper(reference, 2) [['friendship', 'is'], ['is', 'magic']]

Calculating BLEU scores

```
reference = ['friendship', 'is', 'magic']

candidate = ['friendship', 'is', ‘witchcraft’]
```

grouper	`>>> grouper(reference, 2)` `[['friendship', 'is'], ['is', 'magic']]`
n_gram_precision [p80] $\frac{C}{N}$	`>>> n_gram_precision(reference, candidate, 2)` `0.5`

Calculating BLEU scores

```
reference = ['friendship', 'is', 'magic']

candidate = ['friendship', 'is', 'witchcraft']
```

grouper	`>>> grouper(reference, 2)` `[['friendship', 'is'], ['is', 'magic']]`
n_gram_precision [p80] $\frac{C}{N}$	`>>> n_gram_precision(reference, candidate, 2)` `0.5`
brevity_penalty [p84] $\text{Brevity}_i = \frac{r_i}{c_i}$ $\text{BP} = \begin{cases} 1 & \text{brevity}_i < 1 \\ e^{1-\text{brevity}_i} & \text{brevity}_i \geq 1 \end{cases}$	`>>> brevity_penalty(reference, candidate)` `1`

Calculating BLEU scores

```
reference = ['friendship', 'is', 'magic']

candidate = ['friendship', 'is', 'witchcraft']
```

grouper	`>>> grouper(reference, 2)` `[['friendship', 'is'], ['is', 'magic']]`
n_gram_precision [p80] $\frac{C}{N}$	`>>> n_gram_precision(reference, candidate, 2)` `0.5`
brevity_penalty [p84] $\text{Brevity}_i = \frac{r_i}{c_i}$ $\text{BP} = \begin{cases} 1 & \text{brevity}_i < 1 \\ e^{1-\text{brevity}_i} & \text{brevity}_i \geq 1 \end{cases}$	`>>> brevity_penalty(reference, candidate)` `1`

BLEU_score [p85]
$BLEU_c = BP_c \times (p_1 p_2 ... p_n)^{\frac{1}{n}}$
	`>>> BLEU_score(reference, candidate, 2)`
	`=1 x (0.5 x 2/3)^(1/2)`
	`≈0.5774`

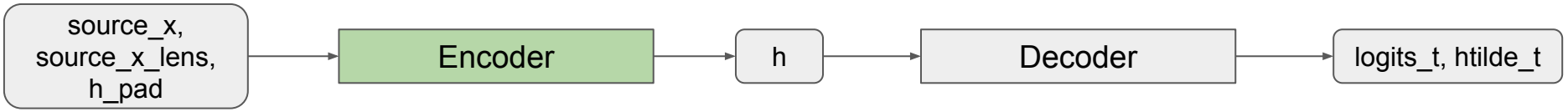
Calculating BLEU scores

- No capping
- Only 1 reference and 1 candidate
- SOS and EOS should not be included in input to BLEU_score

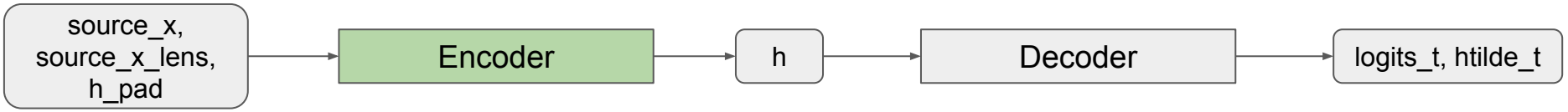
```
reference = ['friendship', 'is', 'magic']

candidate = ['friendship', 'is', 'witchcraft']
```

grouper	`>>> grouper(reference, 2)` `[['friendship', 'is'], ['is', 'magic']]`
n_gram_precision [p80] $\frac{C}{N}$	`>>> n_gram_precision(reference, candidate, 2)` `0.5`
brevity_penalty [p84] $\text{Brevity}_i = \frac{r_i}{c_i}$ $\text{BP} = \begin{cases} 1 & \text{brevity}_i < 1 \\ e^{1-\text{brevity}_i} & \text{brevity}_i \geq 1 \end{cases}$	`>>> brevity_penalty(reference, candidate)` `1`

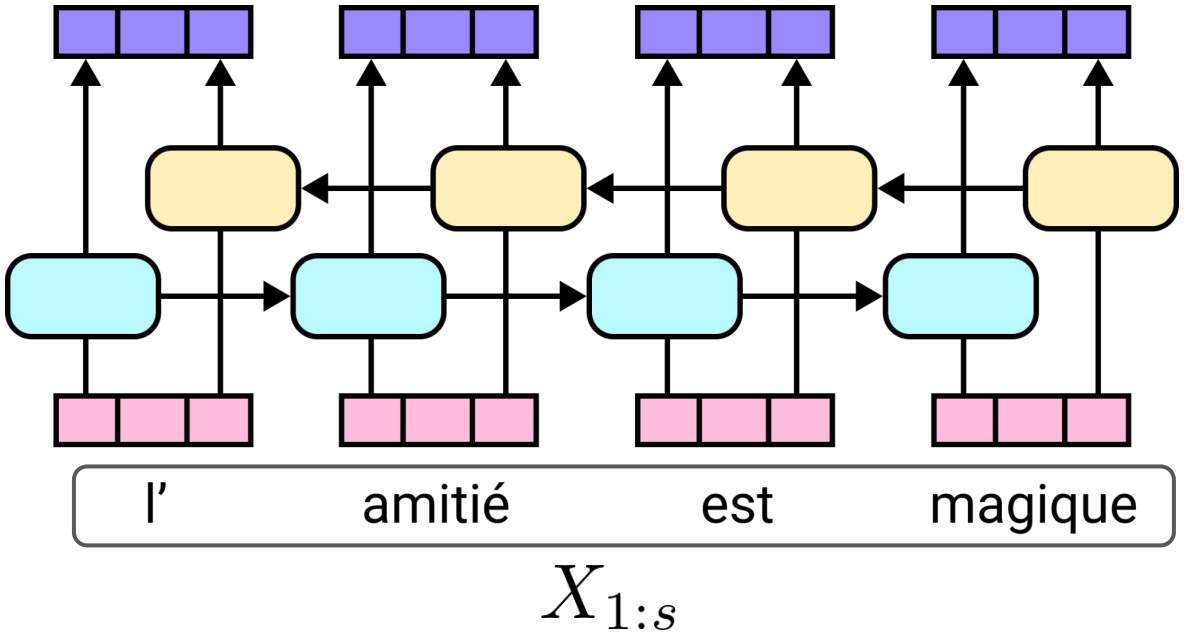
BLEU_score [p85]
$BLEU_c = BP_c \times (p_1 p_2 ... p_n)^{\frac{1}{n}}$
	`>>> BLEU_score(reference, candidate, 2)`
	`=1 x (0.5 x 1/3)^(1/2)`
	`≈0.5774`

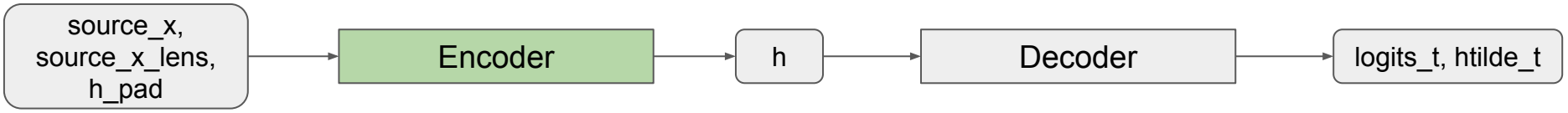

source_x,
source_x_lens,
h_pad
Encoder
h
Decoder
logits_t, htilde_t

Encoder

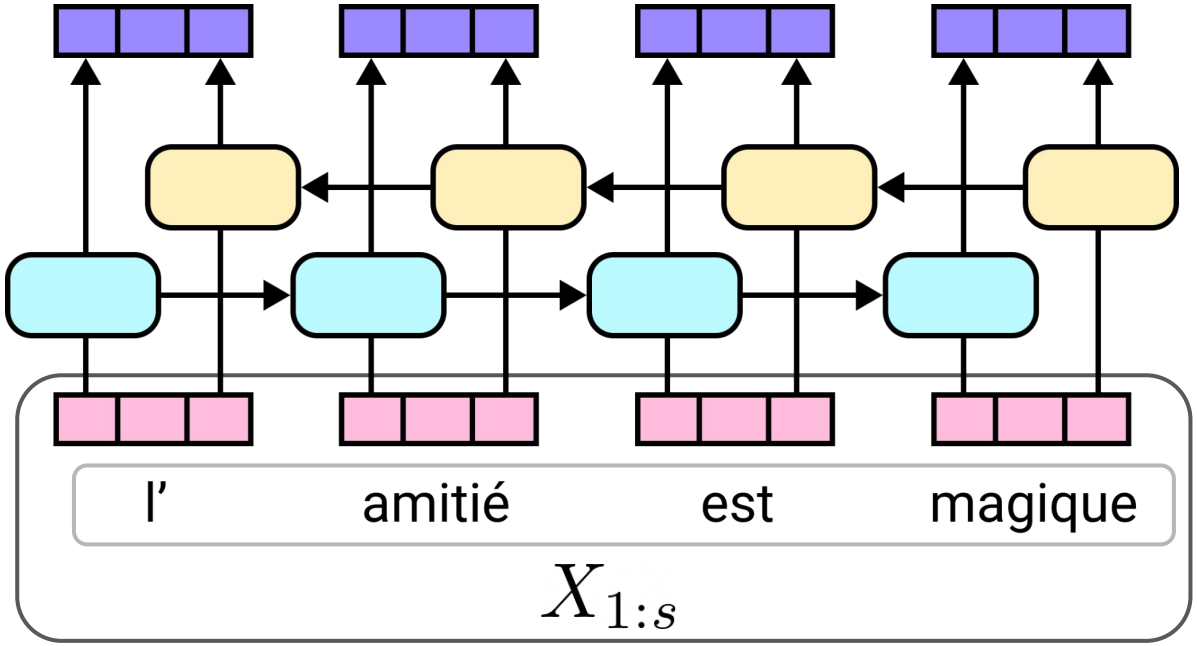

source_x,
source_x_lens,
h_pad
Encoder
h
Decoder
logits_t, htilde_t

Encoder


l'
amitié
est
magique
$X_{1:s}$

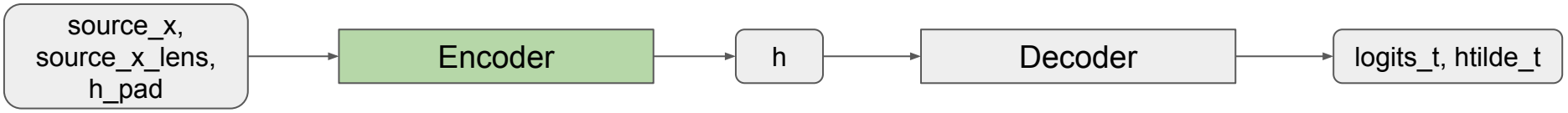


Encoder



$$x_{1:s} = T_{\text{encoder}}(X_{1:s})$$

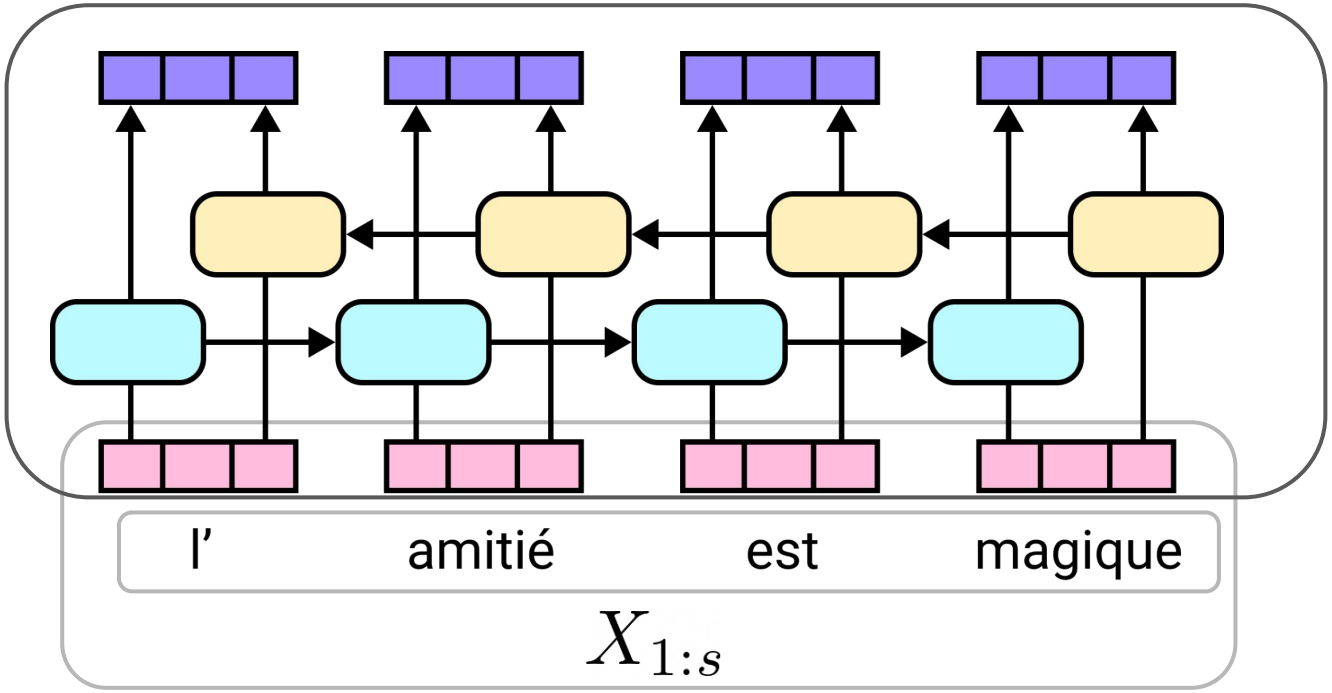
`get_all_rnn_inputs(source_x)`



Encoder

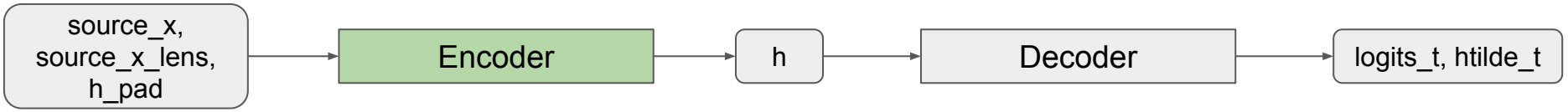
`get_all_hidden_states(x, source_x_lens, h_pad)`

$$h = f(x)$$



$$x_{1:s} = T_{\text{encoder}}(X_{1:s})$$

`get_all_rnn_inputs(source_x)`

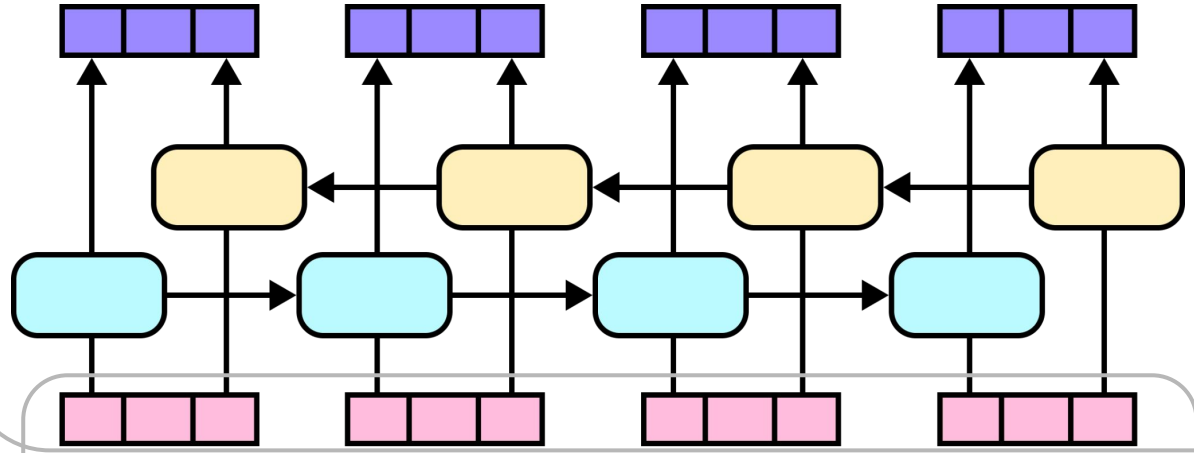


Encoder

```
forward_pass(
F, F_lens, h_pad)
```

```
get_all_hidden_states(x, source_x_lens, h_pad)
```

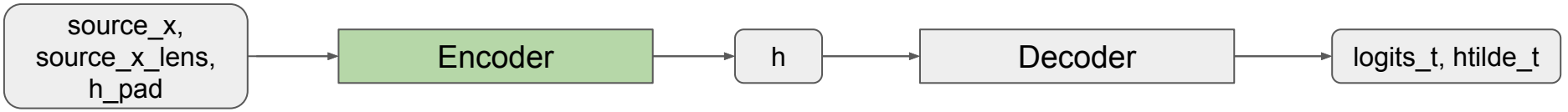
$$h = f(x)$$



$$X_{1:s}$$

$$x_{1:s} = T_{\text{encoder}}(X_{1:s})$$

```
get_all_rnn_inputs(source_x)
```



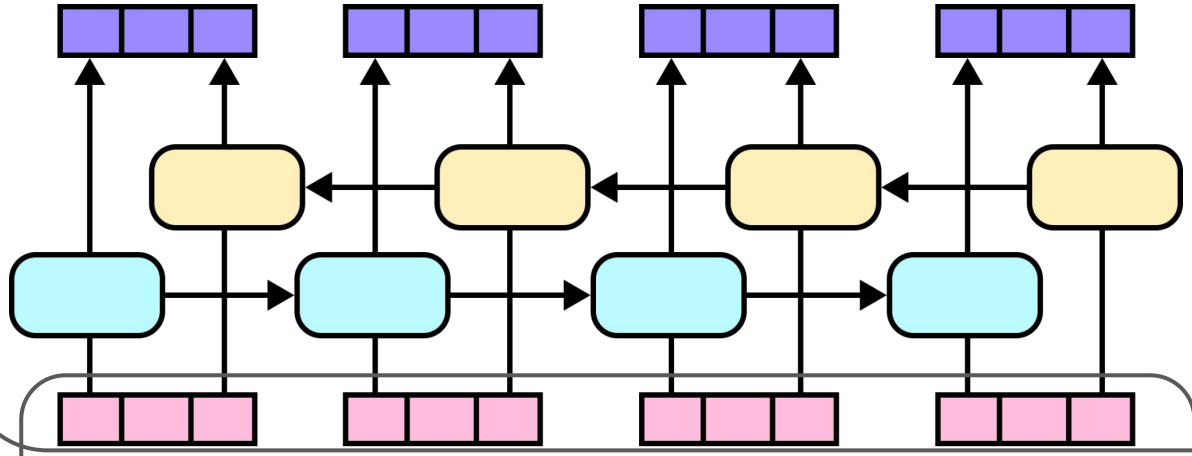
Encoder

`forward_pass(F, F_lens, h_pad)`

note: embeddings and the rnn layer should be initialized in `init_submodules`

`get_all_hidden_states(x, source_x_lens, h_pad)`

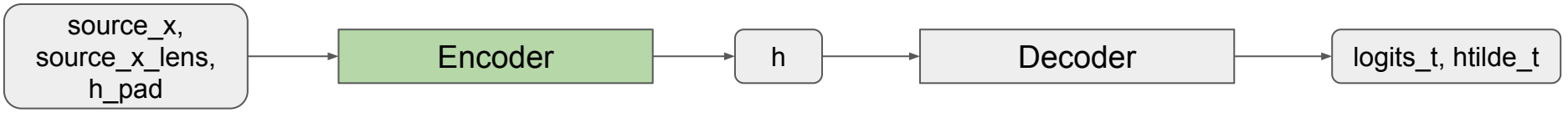
$$h = f(x)$$



$$X_{1:s}$$

$$x_{1:s} = T_{\text{encoder}}(X_{1:s})$$

`get_all_rnn_inputs(source_x)`



Encoder

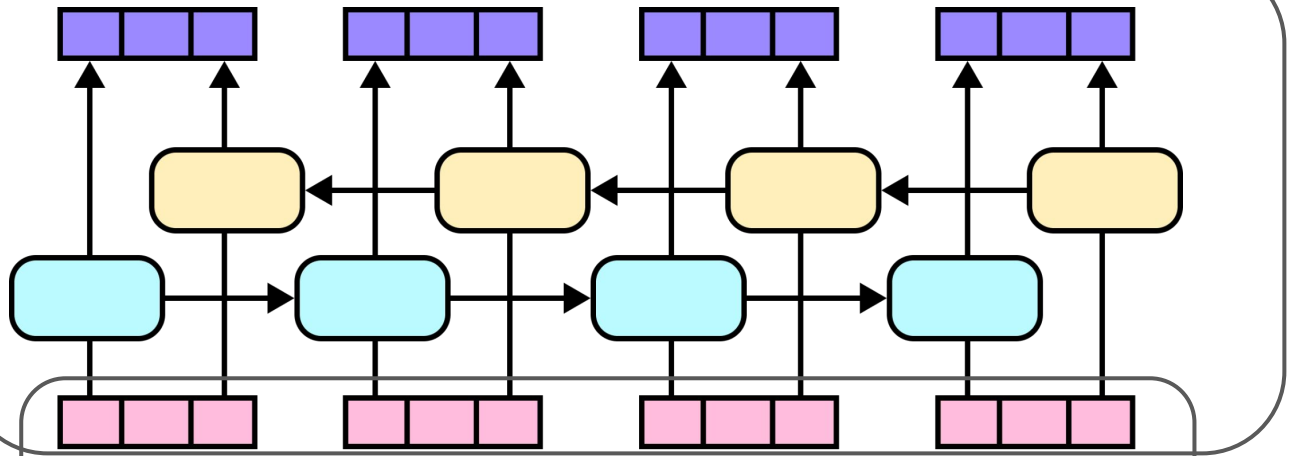
`forward_pass(F, F_lens, h_pad)`

note: embeddings and the rnn layer should be initialized in `init_submodules`

Q: what is *h*? why do we pass all of *h* to the decoder?

`get_all_hidden_states(x, source_x_lens, h_pad)`

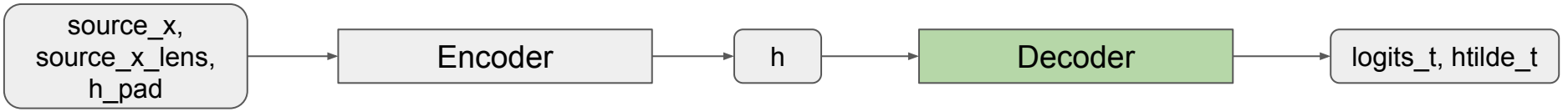
$$h = f(x)$$



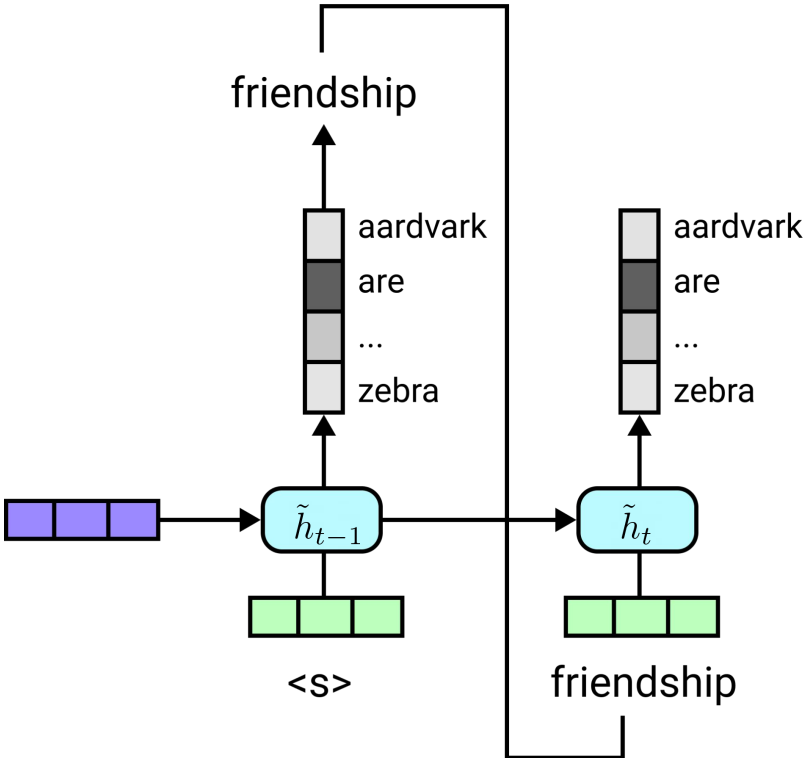
$$X_{1:s}$$

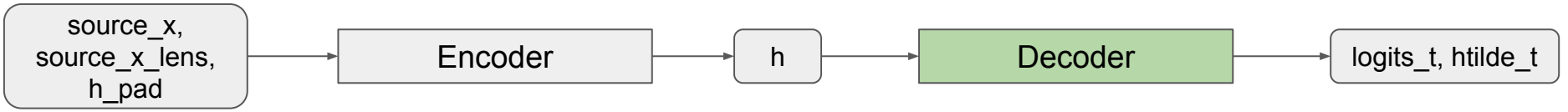
$$x_{1:s} = T_{\text{encoder}}(X_{1:s})$$

`get_all_rnn_inputs(source_x)`

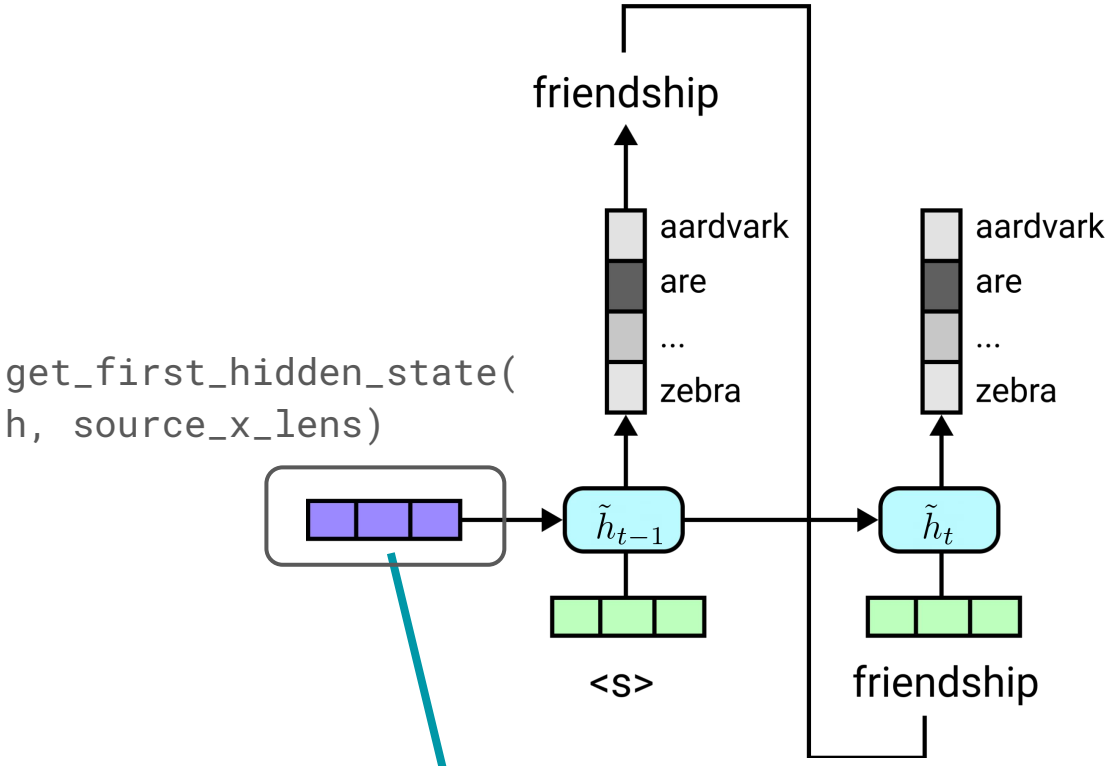


DecoderWithoutAttention

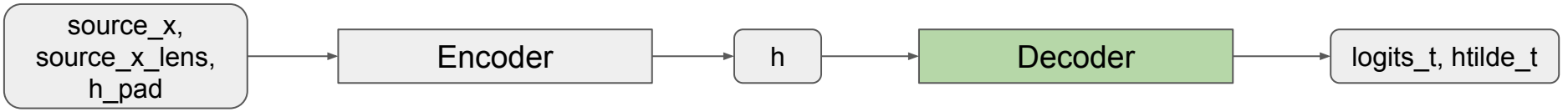




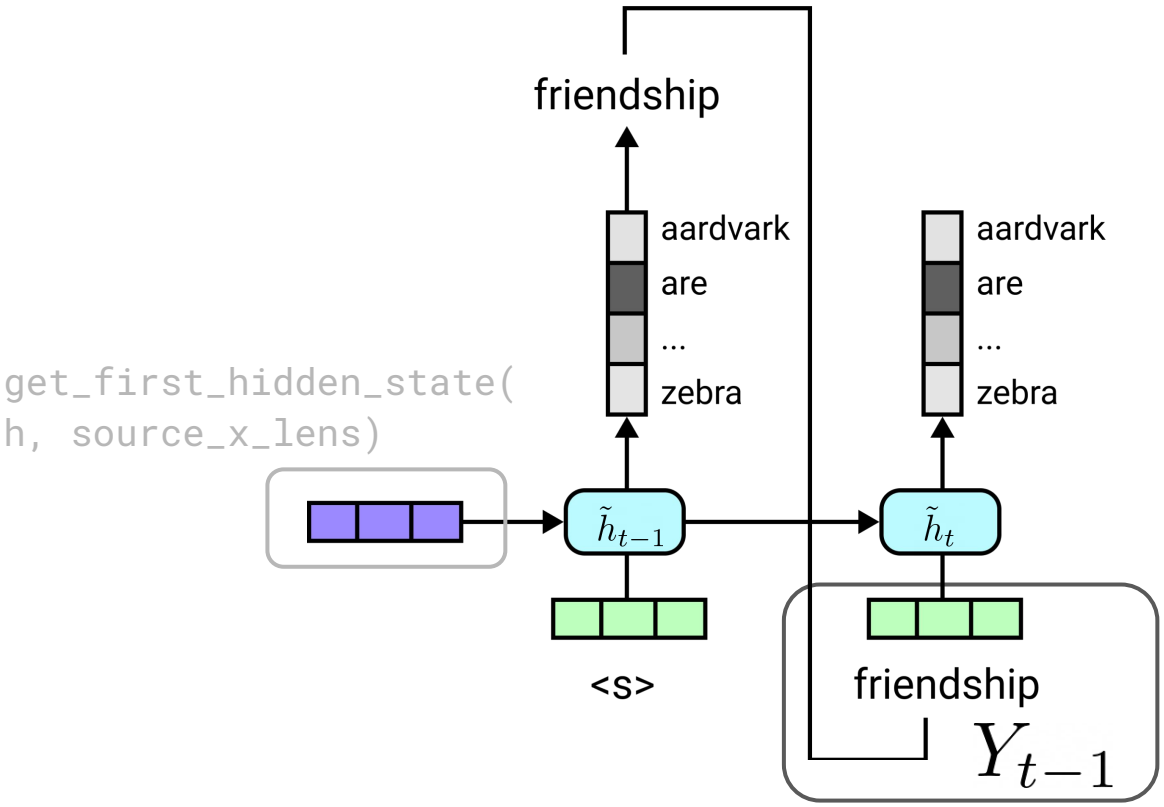
DecoderWithoutAttention



Q: how do we initialize htilde_1?



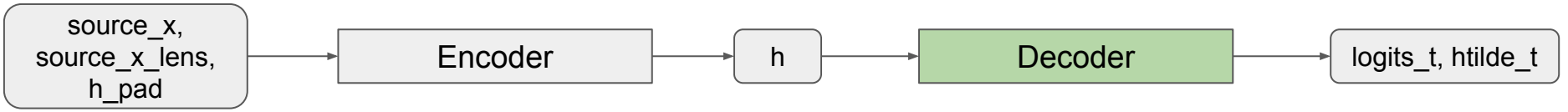
DecoderWithoutAttention



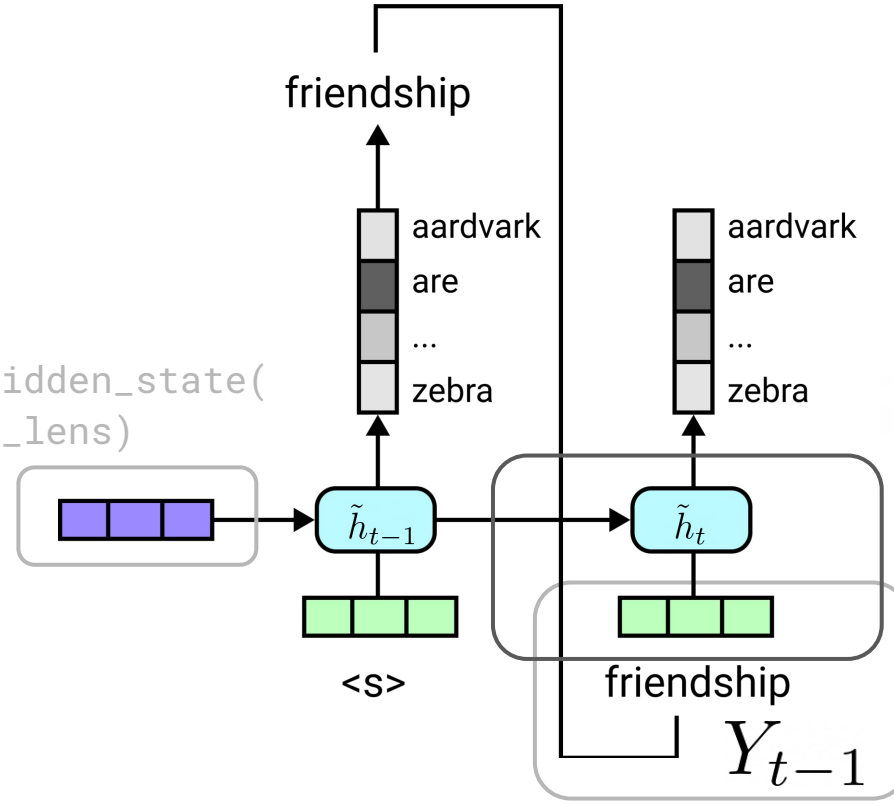

```
get_first_hidden_state(
h, source_x_lens)
```

$$\tilde{x}_t = T_{\text{decoder}}(Y_{t-1})$$

```
get_current_rnn_input(
E_tm1, htilde_tm1, h, F_lens)
```



DecoderWithoutAttention



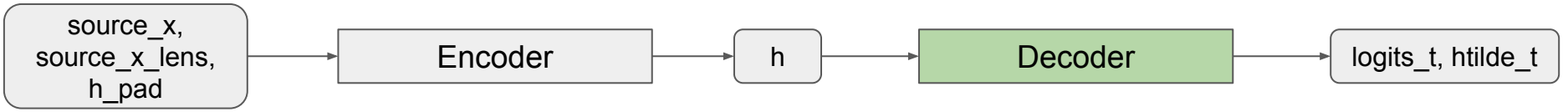

```
get_first_hidden_state(
h, source_x_lens)
```

$$\tilde{h}_t = g(\tilde{x}_t, \tilde{h}_{t-1})$$

```
get_current_hidden_state(
xtilde_t, htilde_tm1)
```

$$\tilde{x}_t = T_{\text{decoder}}(Y_{t-1})$$

```
get_current_rnn_input(
E_tm1, htilde_tm1, h, F_lens)
```

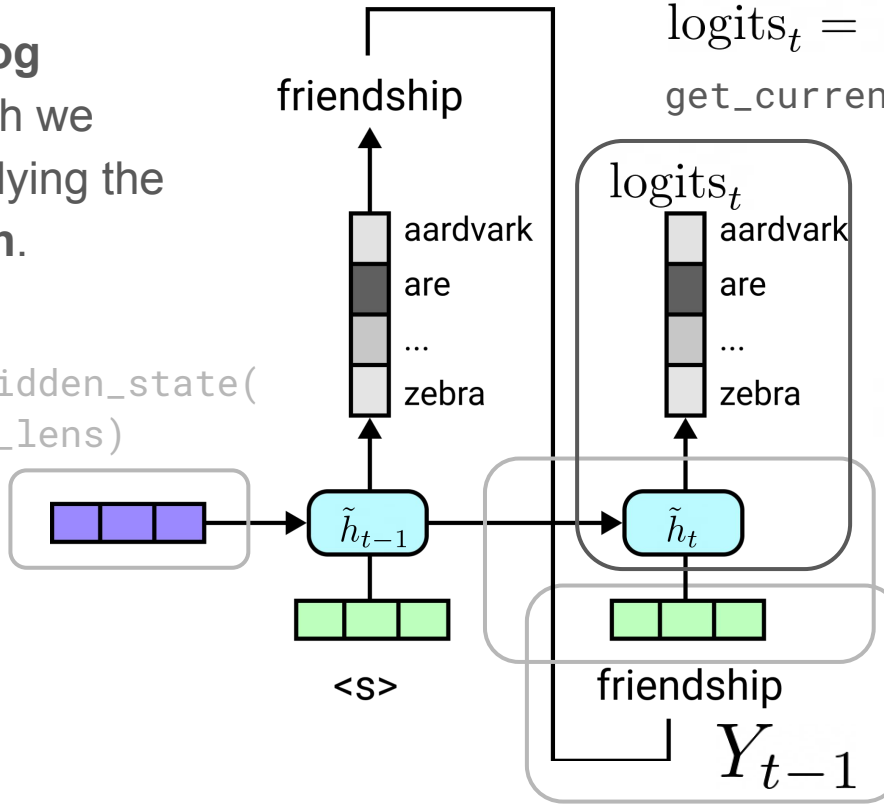


DecoderWithoutAttention

logits_t is the **un-normalized log probability**, which we normalize by applying the **softmax function**.

$\text{logits}_t = f(\tilde{h}_t)$

```
get_current_logits(htilde_t)
```



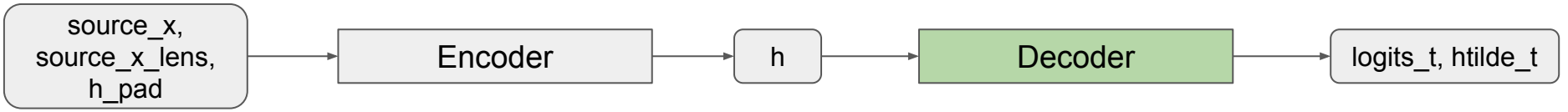
$\tilde{h}_t = g(\tilde{x}_t, \tilde{h}_{t-1})$

```
get_first_hidden_state(
h, source_x_lens)
```

```
get_current_hidden_state(
xtilde_t, htilde_tm1)
```

$\tilde{x}_t = T_{\text{decoder}}(Y_{t-1})$

```
get_current_rnn_input(
E_tm1, htilde_tm1, h, F_lens)
```

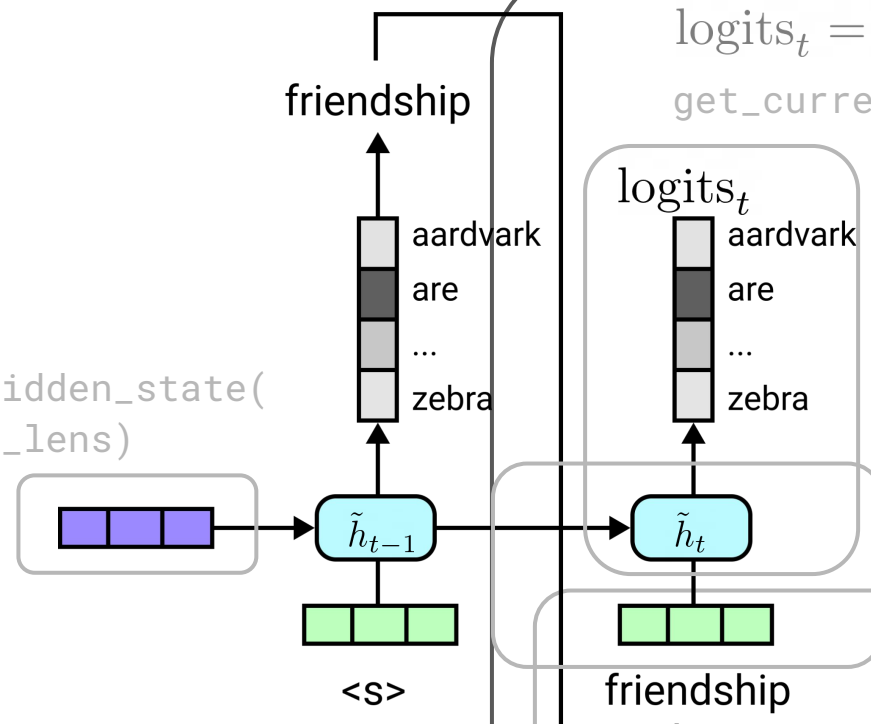


DecoderWithoutAttention

```
forward_pass(target_y_tm1, htilde_tm1, h,
source_x_lens)
```

$\text{logits}_t = f(\tilde{h}_t)$

```
get_current_logits(htilde_t)
```



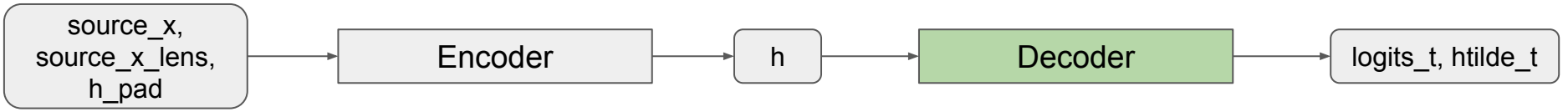

```
get_first_hidden_state(
h, source_x_lens)
```

$\tilde{h}_t = g(\tilde{x}_t, \tilde{h}_{t-1})$

```
get_current_hidden_state(
xtilde_t, htilde_tm1)
```

$\tilde{x}_t = T_{\text{decoder}}(Y_{t-1})$

```
get_current_rnn_input(
E_tm1, htilde_tm1, h, F_lens)
```



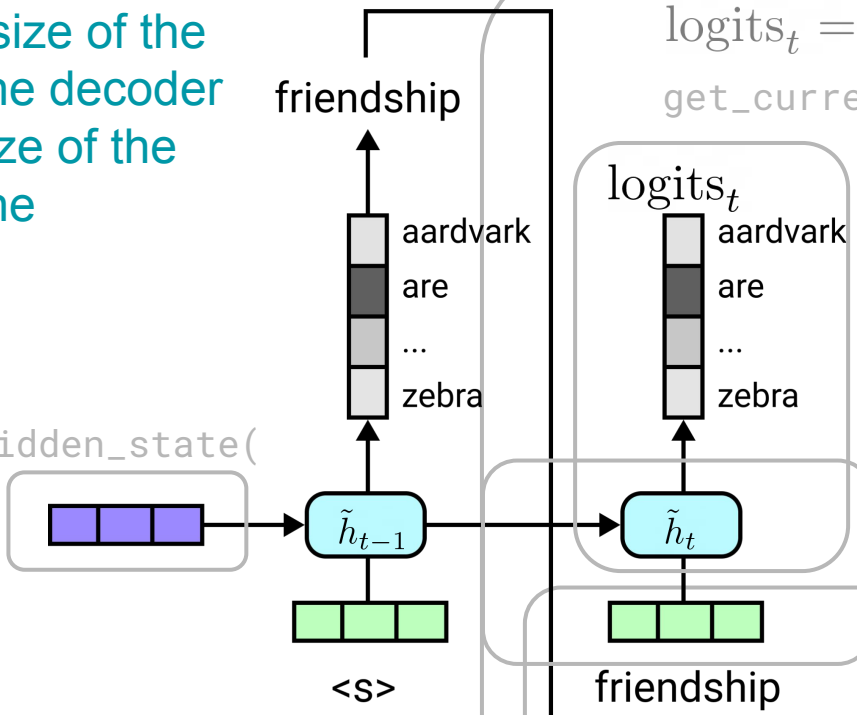
DecoderWithoutAttention

Q: how does the size of the hidden state for the decoder compare to the size of the hidden state for the encoder?

```
forward_pass(target_y_tm1, htilde_tm1, h,
source_x_lens)
```

$\text{logits}_t = f(\tilde{h}_t)$

```
get_current_logits(htilde_t)
```



$\tilde{h}_t = g(\tilde{x}_t, \tilde{h}_{t-1})$

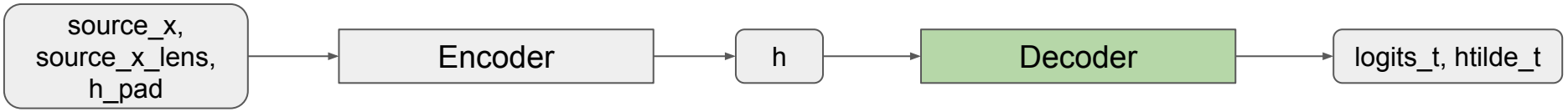
```
get_first_hidden_state(
h, F_lens)
```

```
get_current_hidden_state(
xtilde_t, htilde_tm1)
```

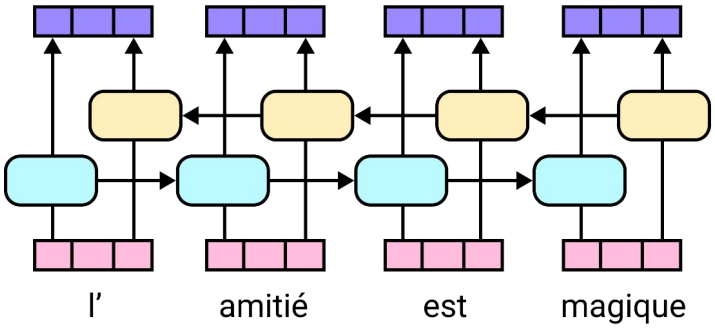
Y_{t-1}

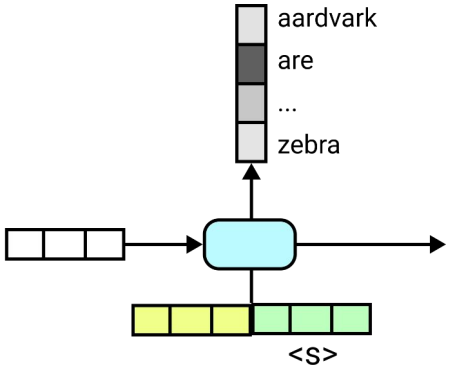
$\tilde{x}_t = T_{\text{decoder}}(Y_{t-1})$

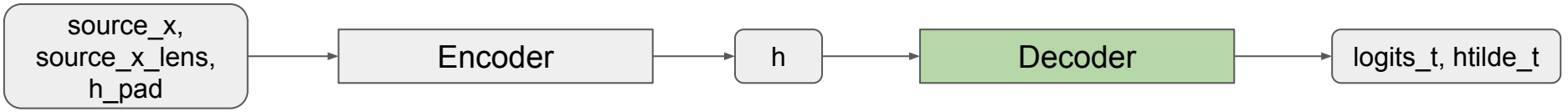
```
get_current_rnn_input(
E_tm1, htilde_tm1, h, F_lens)
```



DecoderWithAttention

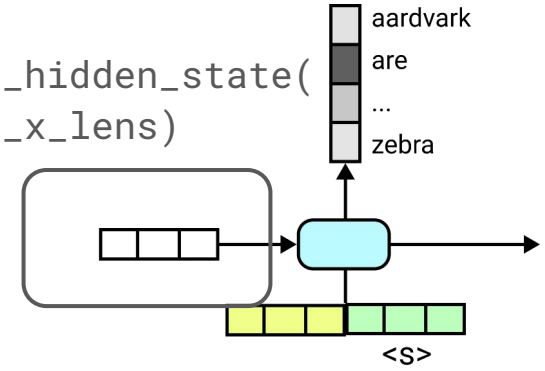


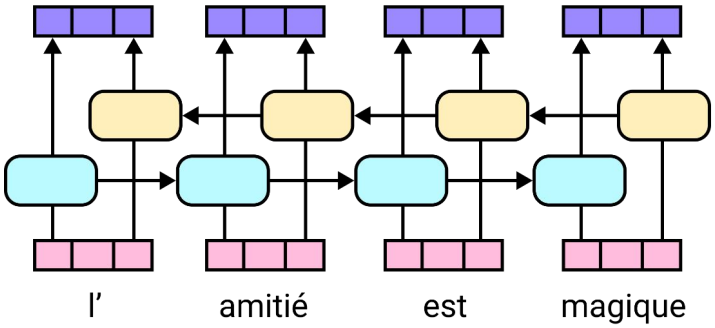


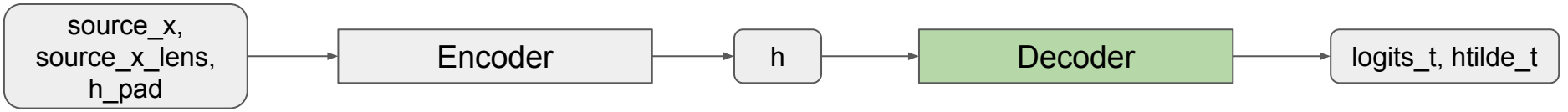


DecoderWithAttention

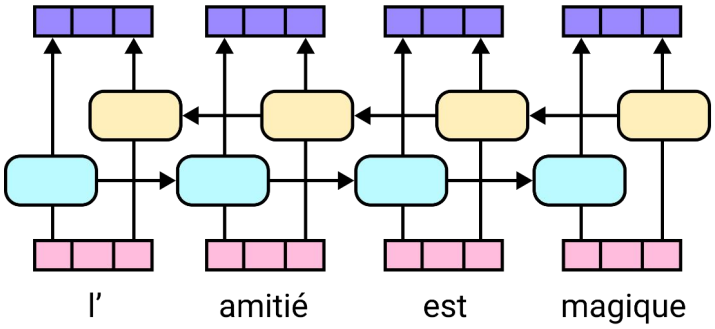
```
get_first_hidden_state(
h, source_x_lens)
```

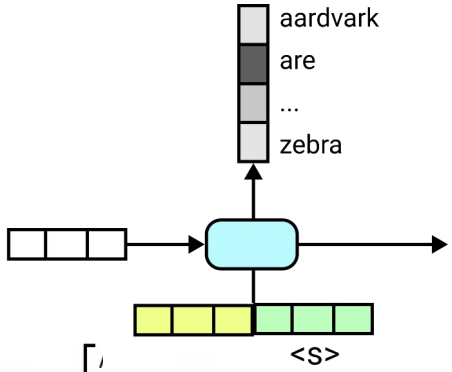






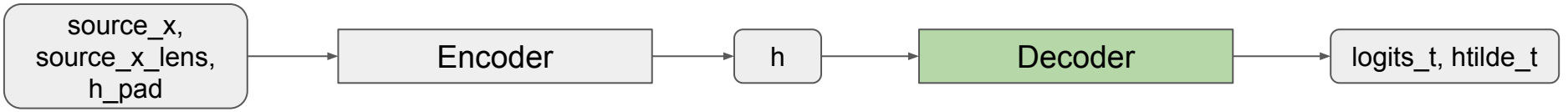
DecoderWithAttention





$$\tilde{x}_t = [c_{t-1}, T_{\text{dec}}(Y_{t-1})]$$

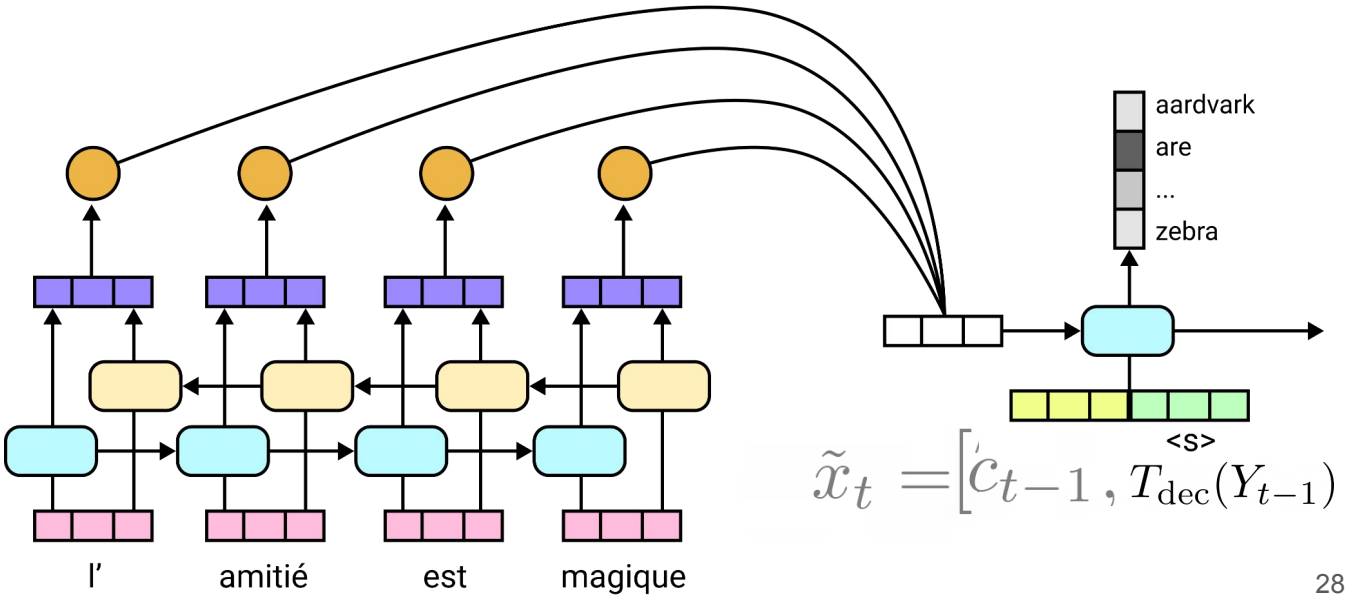
Q: How do we compute c_tm1?



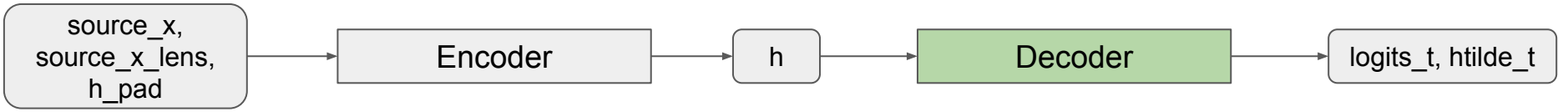
DecoderWithAttention

`get_attention_scores(htilde_t, h)`

$$a_{t-1,s} = \text{cosine\_sim}(\tilde{h}_{t-1}, h_s)$$



$$\tilde{x}_t = [c_{t-1}, T_{\text{dec}}(Y_{t-1})]$$

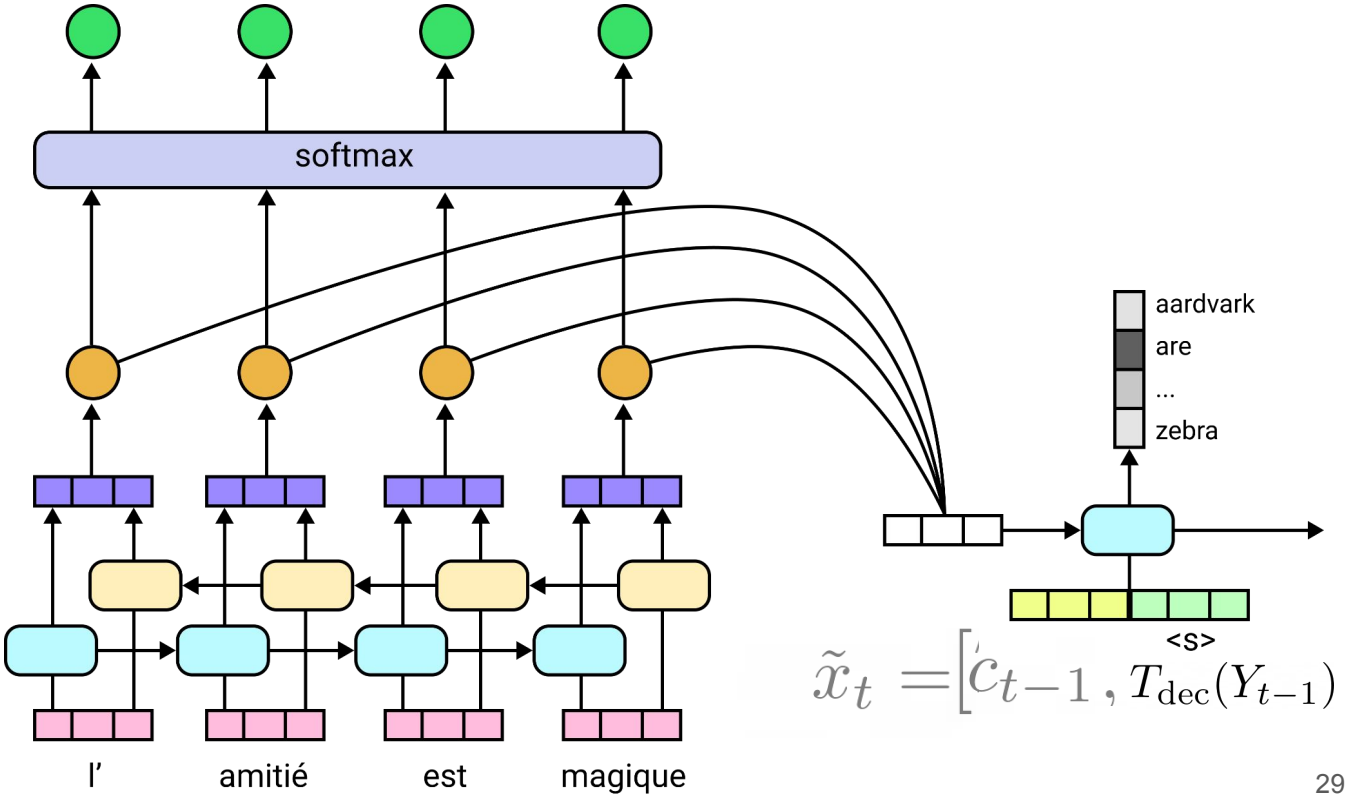


DecoderWithAttention

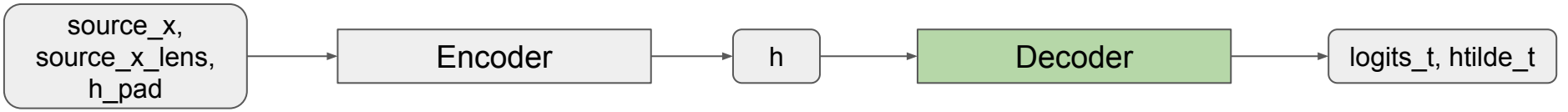
`get_attention_weights(htilde_t, h, source_x_lens)`

(implemented for you; calls get_attention_scores)

$$\alpha_{t-1} = \text{softmax}(a_{t-1,1:s}, s)$$



$$\tilde{x}_t = [c_{t-1}, T_{\text{dec}}(Y_{t-1})]$$

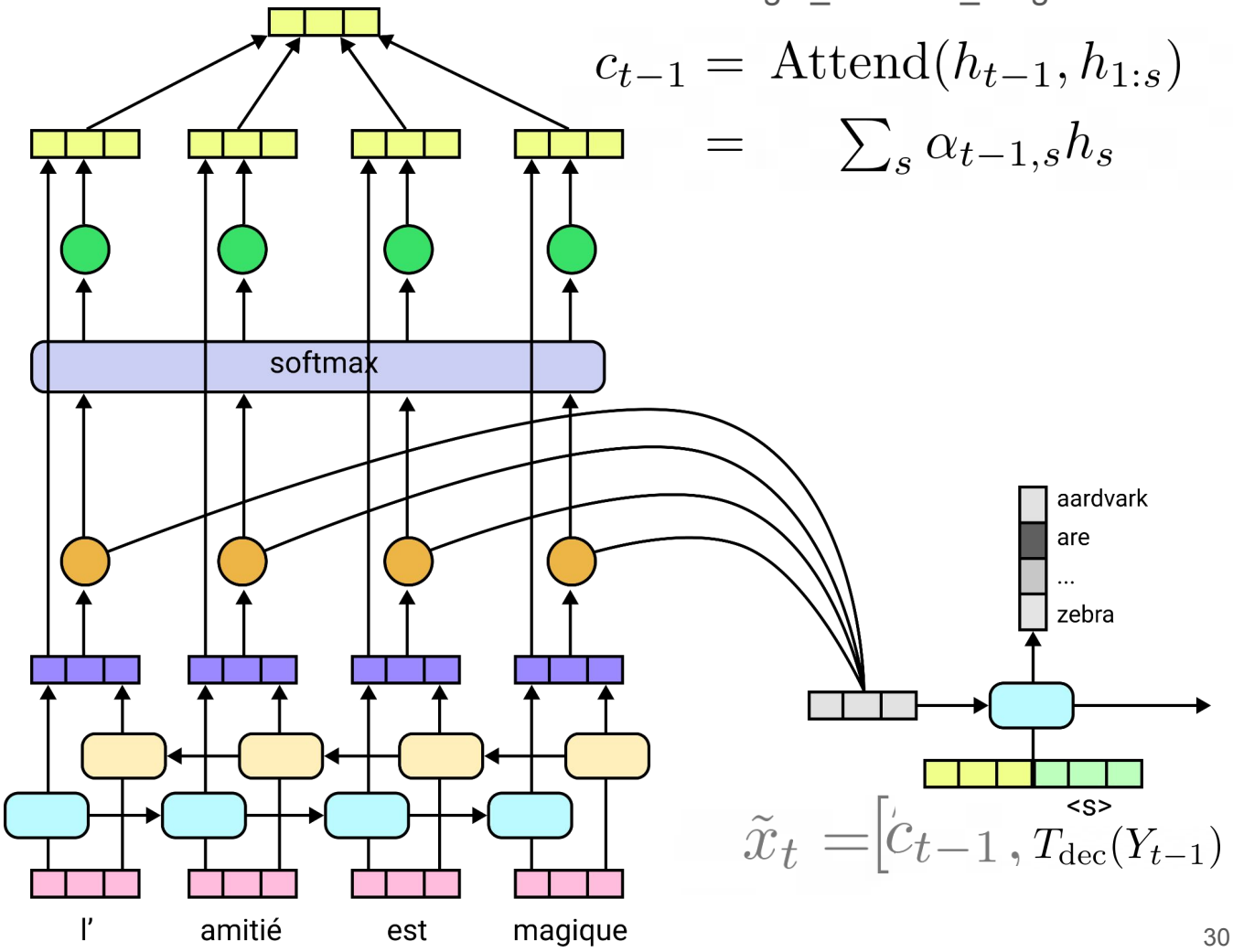


DecoderWithAttention

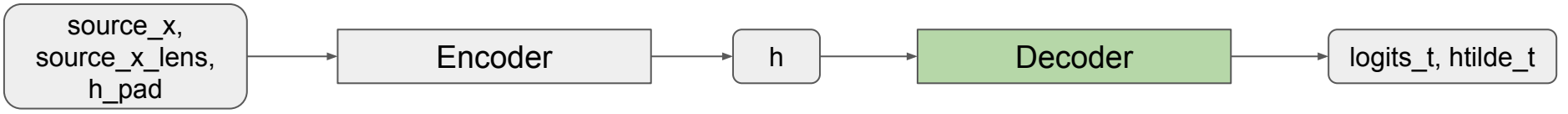
`attend(htilde_t, h, source_x_lens)`

should call get_attention_weights

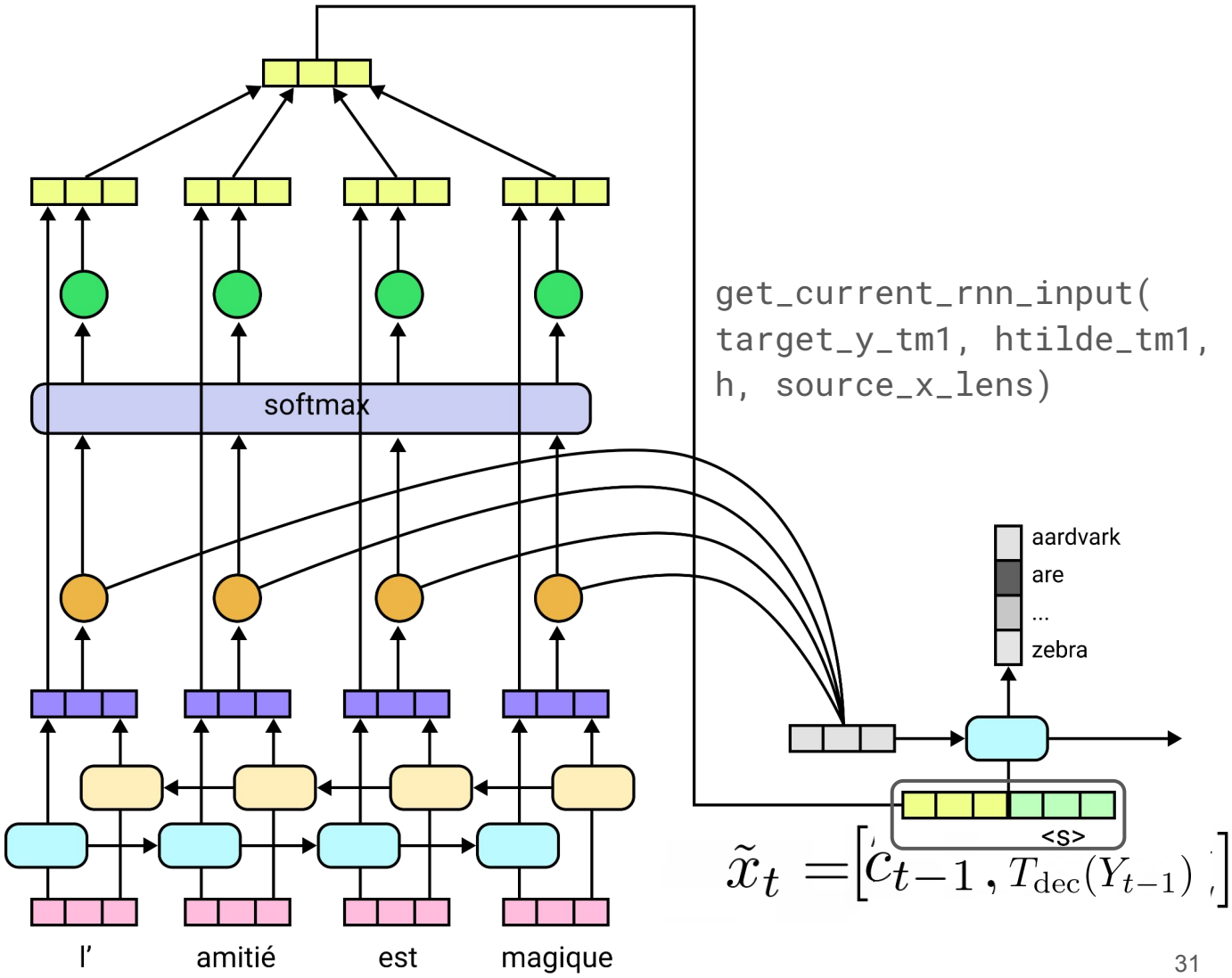
$$c_{t-1} = \text{Attend}(h_{t-1}, h_{1:s}) = \sum_s \alpha_{t-1,s} h_s$$

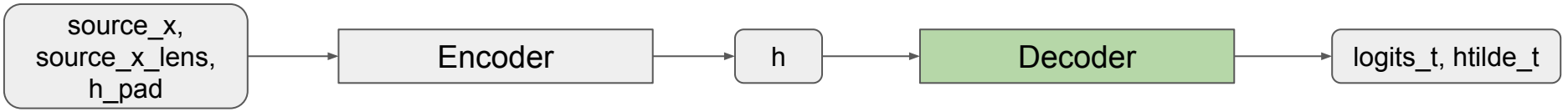


$$\tilde{x}_t = [c_{t-1}, T_{\text{dec}}(Y_{t-1})]$$

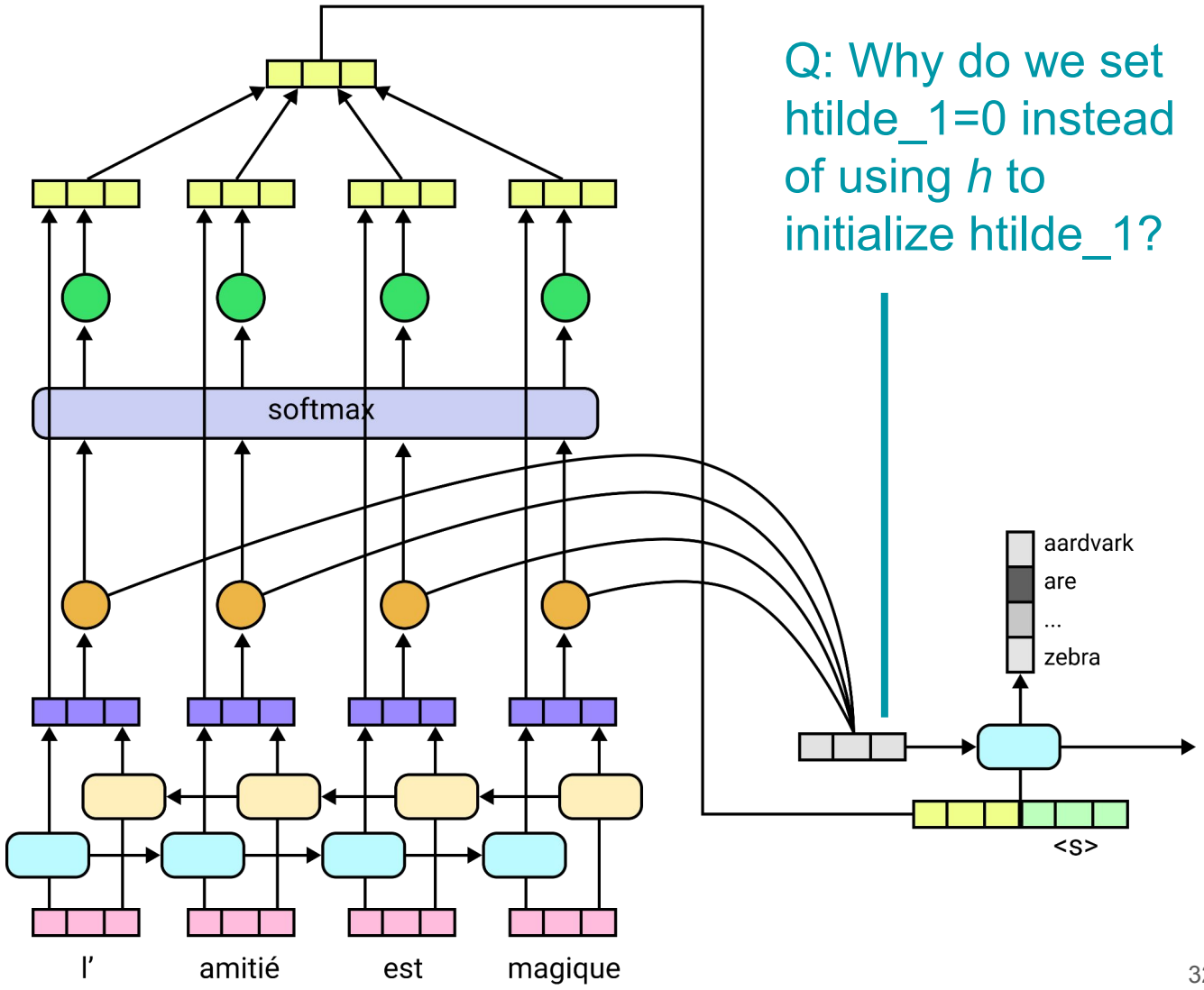


DecoderWithAttention

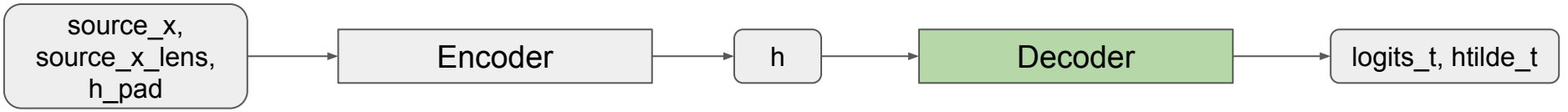




DecoderWithAttention



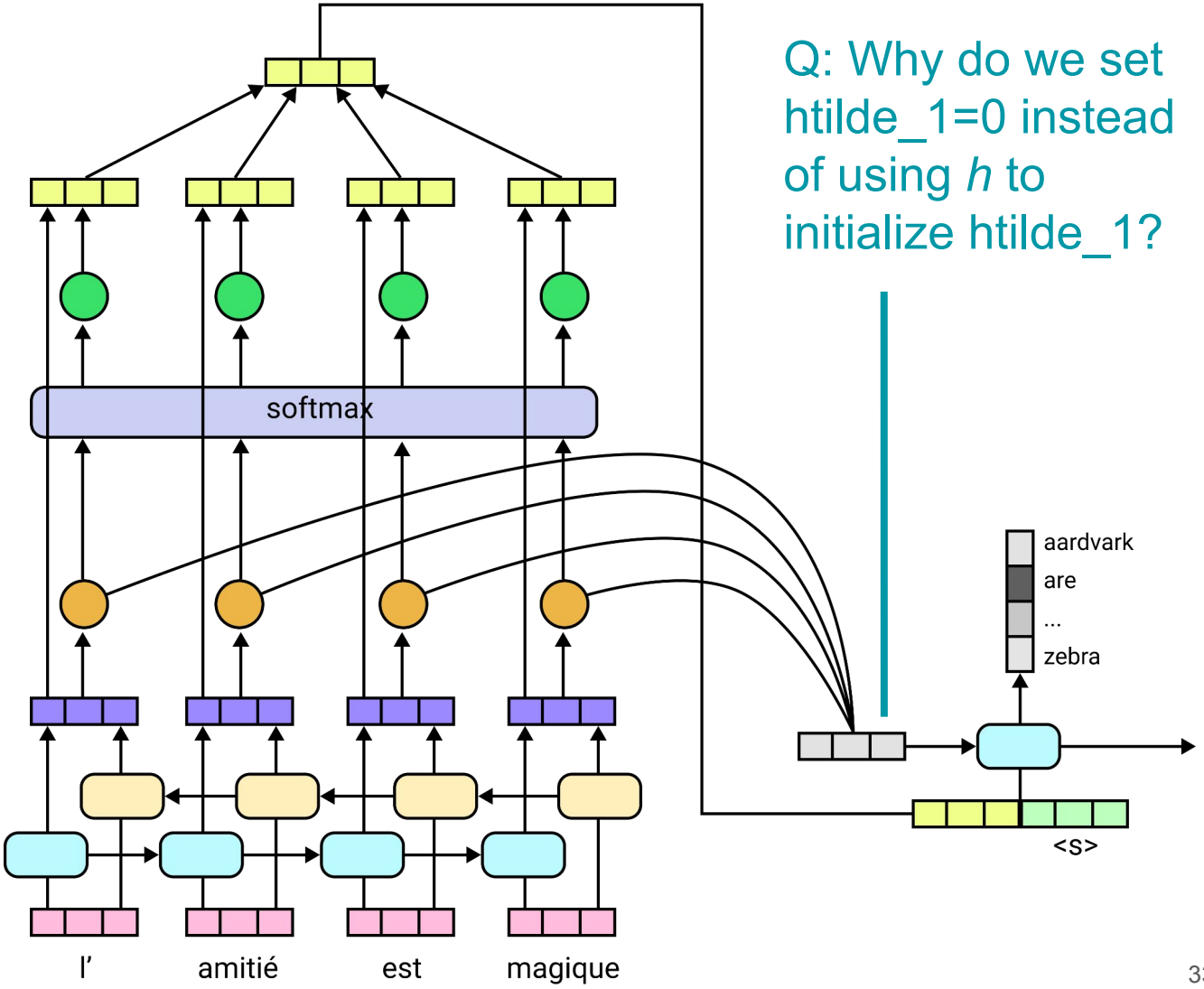
Q: Why do we set htilde_1=0 instead of using *h* to initialize htilde_1?



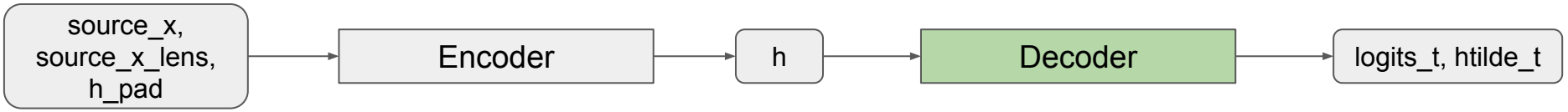
DecoderWithAttention

Tips:

- Be careful with t vs. t-1!
- Note that we use cosine similarity as the score function (1.4) -- different from in class
- [p52-53]



Q: Why do we set htilde_1=0 instead of using *h* to initialize htilde_1?



DecoderWithMultiHeadAttention

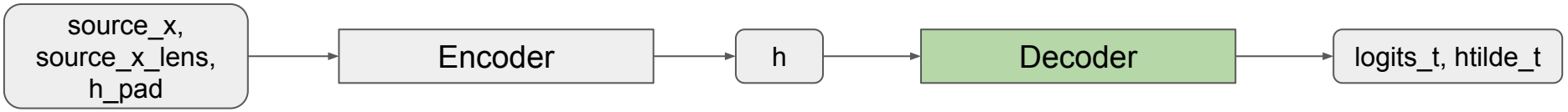
Decoder

$$\tilde{h}_{t-1}^{(n)} = \tilde{W}^{(n)} \tilde{h}_{t-1}$$

$$h_s^{(n)} = W^{(n)} h_s$$

$$c_{t-1}^{(n)} = \text{Attention}(\tilde{h}_{t-1}^{(n)}, h_{1:s}^{(n)})$$

$$\tilde{x}_t = [Q c_{t-1}^{(1:N)}, T_{\text{dec}}(Y_{t-1})]$$



DecoderWithMultiHeadAttention

Decoder

$$\tilde{h}_{t-1}^{(n)} = \tilde{W}^{(n)} \tilde{h}_{t-1}$$

$$h_s^{(n)} = W^{(n)} h_s$$

$$c_{t-1}^{(n)} = \text{Attention}(\tilde{h}_{t-1}^{(n)}, h_{1:s}^{(n)})$$

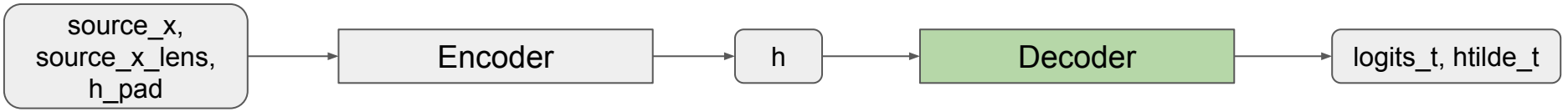
$$\tilde{x}_t = [Qc_{t-1}^{(1:N)}, T_{\text{dec}}(Y_{t-1})]$$

Main change happens in

```
attend(htilde_t,
    h, source_x_lens),
```

which performs these steps and returns

$$Qc_{t-1}^{(1:N)}$$



DecoderWithMultiHeadAttention

Decoder

$$\tilde{h}_{t-1}^{(n)} = \tilde{W}^{(n)} \tilde{h}_{t-1}$$

$$h_s^{(n)} = W^{(n)} h_s$$

$$c_{t-1}^{(n)} = \text{Attention}(\tilde{h}_{t-1}^{(n)}, h_{1:s}^{(n)})$$

$$\tilde{x}_t = [Qc_{t-1}^{(1:N)}, T_{\text{dec}}(Y_{t-1})]$$

Main change happens in

```
attend(htilde_t,
    h, source_x_lens),
```

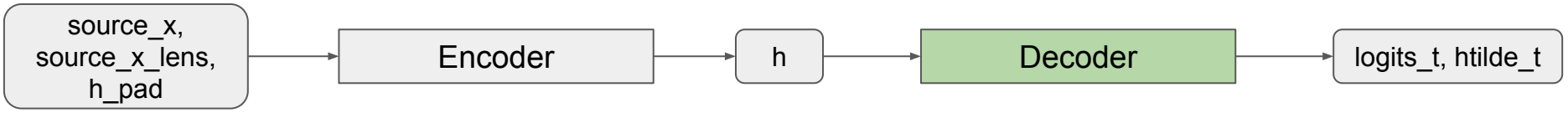
which performs these steps and returns

$$Qc_{t-1}^{(1:N)}$$

You can perform this step by calling

```
super().attend(...)
```

Q: The assignment handout talks about how the $W^{(n)}$ matrices don't have to be square. Why might non-square $W^{(n)}$ matrices be desirable?



DecoderWithMultiHeadAttention

Decoder

$$\tilde{h}_{t-1}^{(n)} = \tilde{W}^{(n)} \tilde{h}_{t-1}$$

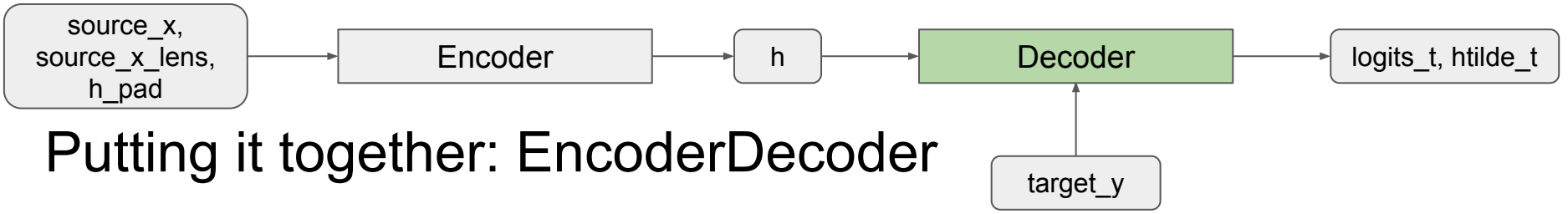
$$h_s^{(n)} = W^{(n)} h_s$$

$$c_{t-1}^{(n)} = \text{Attention}(\tilde{h}_{t-1}^{(n)}, h_{1:s}^{(n)})$$

$$\tilde{x}_t = [Q c_{t-1}^{(1:N)}, T_{\text{dec}}(Y_{t-1})]$$

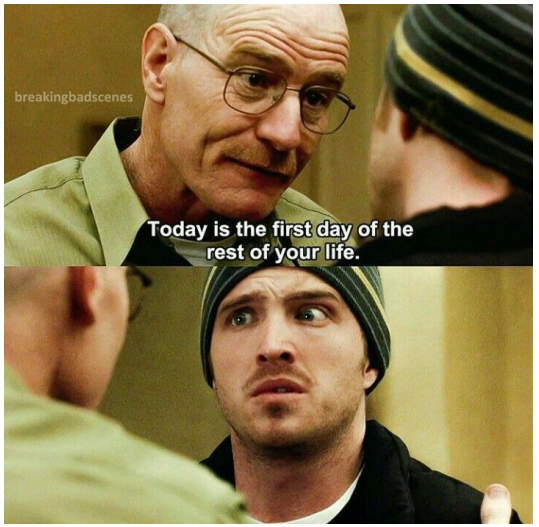
Tips:

- You don't really need to slice the hidden weights
- Try starting with heads=1
- You also shouldn't use for loops!
- But you can use for loops for testing.
- [p55]



Putting it together: EncoderDecoder

- `init_submodules`
- `get_logits_for_teacher_forcing`
  - Basic idea: replace y with target_y
  - [p49]
  - Q: How might this affect training?
- `update_beam`
  - One step of the beam search
  - A greedy update function is provided to you, you can test the rest of the assignment by using the `--greedy` option
  - [p70-79]
- `translate`
  - Inference on one input sentence
  - Clarification: you can assume the model is one cpu



Training and Testing Loop

- `train_for_epoch`
  - Follow the instructions in the docstring
  - Don’t forget to normalize loss!
  - tqdm: easy progress bar

- `compute_batch_total_bleu`
  - `a2_bleu_score.BLEU_score` for a batch of sentences
  - tip: don’t pass sos and eos tokens to `a2_bleu_score.BLEU_score`

- `compute_average_bleu_over_dataset`
  - Calculate the average BLEU score of the given dataset
  - Use `compute_batch_total_bleu`
  - Should be very similar to `translate`