IN5550: Neural Methods in Natural Language Processing

Encoder-Decoder Models, Attention and Transformers

Sondre Wold

University of Oslo

March 2023



Previous weeks:

- Recurrent Neural Networks (RNNs)
  - Acceptors: sentence-level sentiment classification
  - Transducers: part-of-speech tagging

This lecture

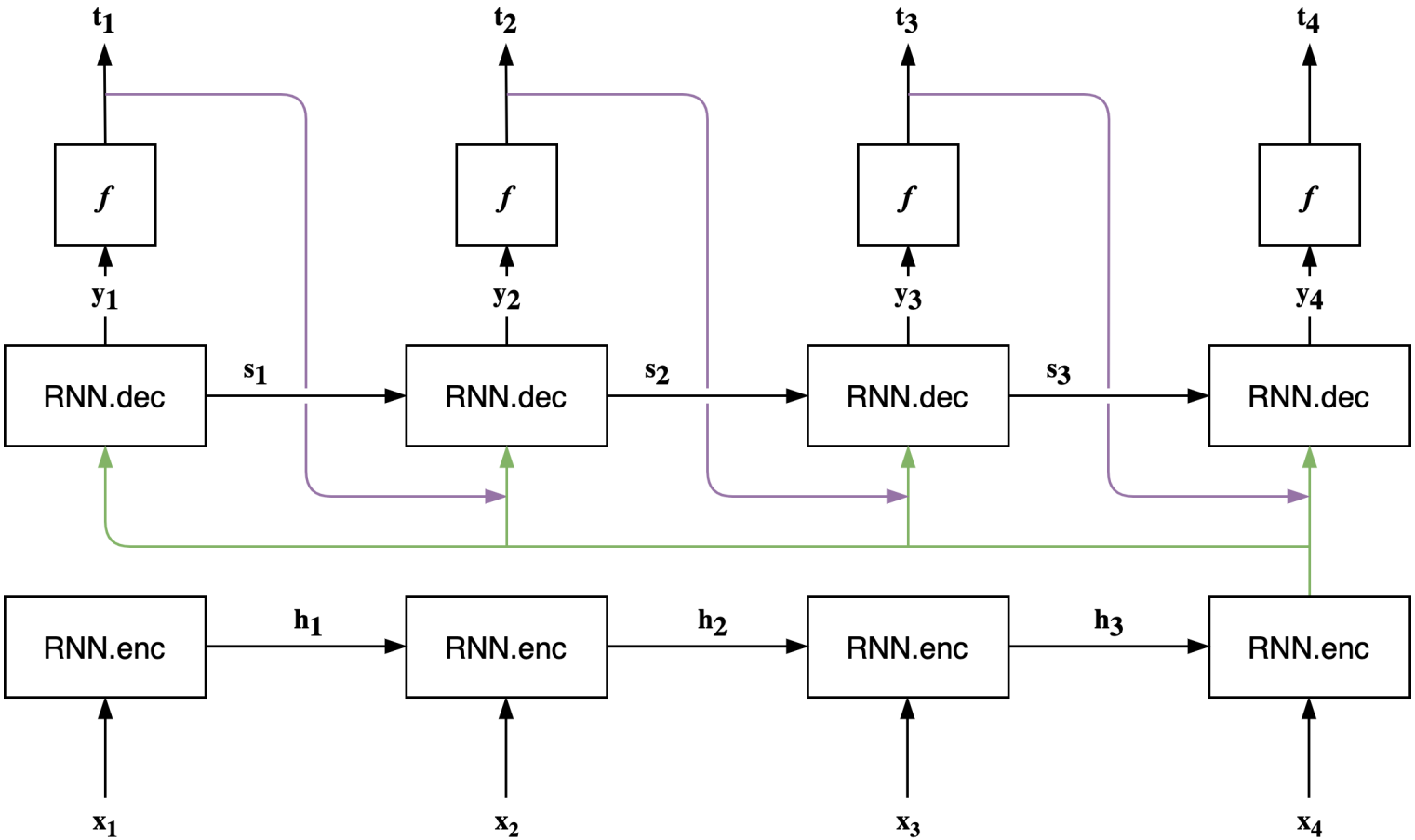
- Part 1: Encoder-Decoder model for Sequence2Sequence problems, like machine translation.
- Part 1.5: The attention mechanism
- Part 2: The Transformer architecture
  - Self-attention

Problem definition



- We want to learn a function that maps an input sequence $X_{1:n}$ to an output sequence $y_{1:n}$.
- We want this to work for arbitrarily long sequences for both the source and target.
- Examples:
  - Machine translation, text to code, summarization etc..
- Looks like we need something that can both encode $X_{1:n}$ and decode $y_{1:n}$
- What have learned about that can both encode arbitrarily long sequences into fixed size vectors and can be used as a generator?

Encoder-Decoder Models with RNNs

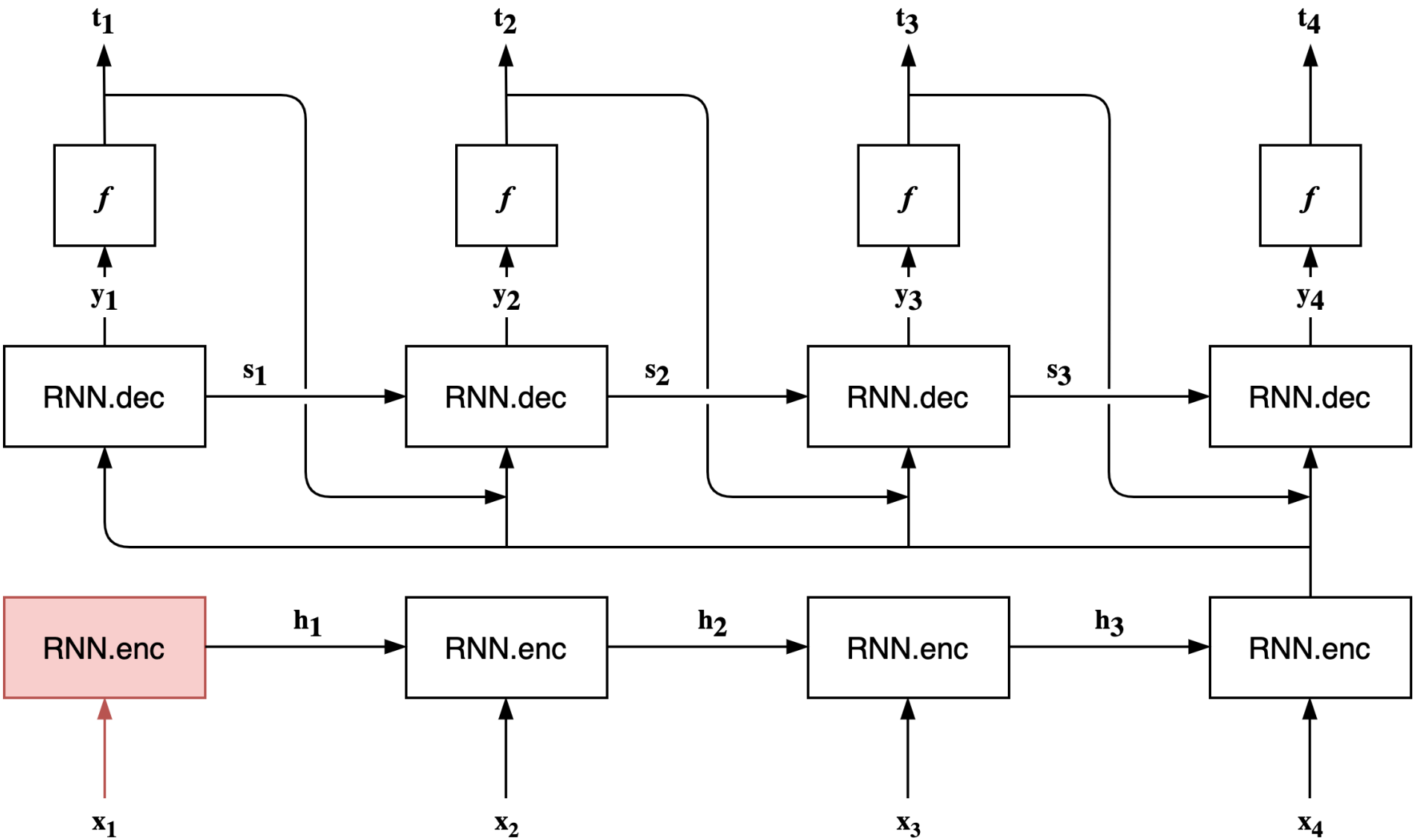


- Use an RNN as an encoder, mapping $X_{1:n}$ to a context vector $\mathbf{c}$
- Use another RNN as a decoder to generate $y_{1:n}$ — conditioned on $\mathbf{c}$.
- This is sometimes also referred to as the "sequence to sequence conditioned generation framework"

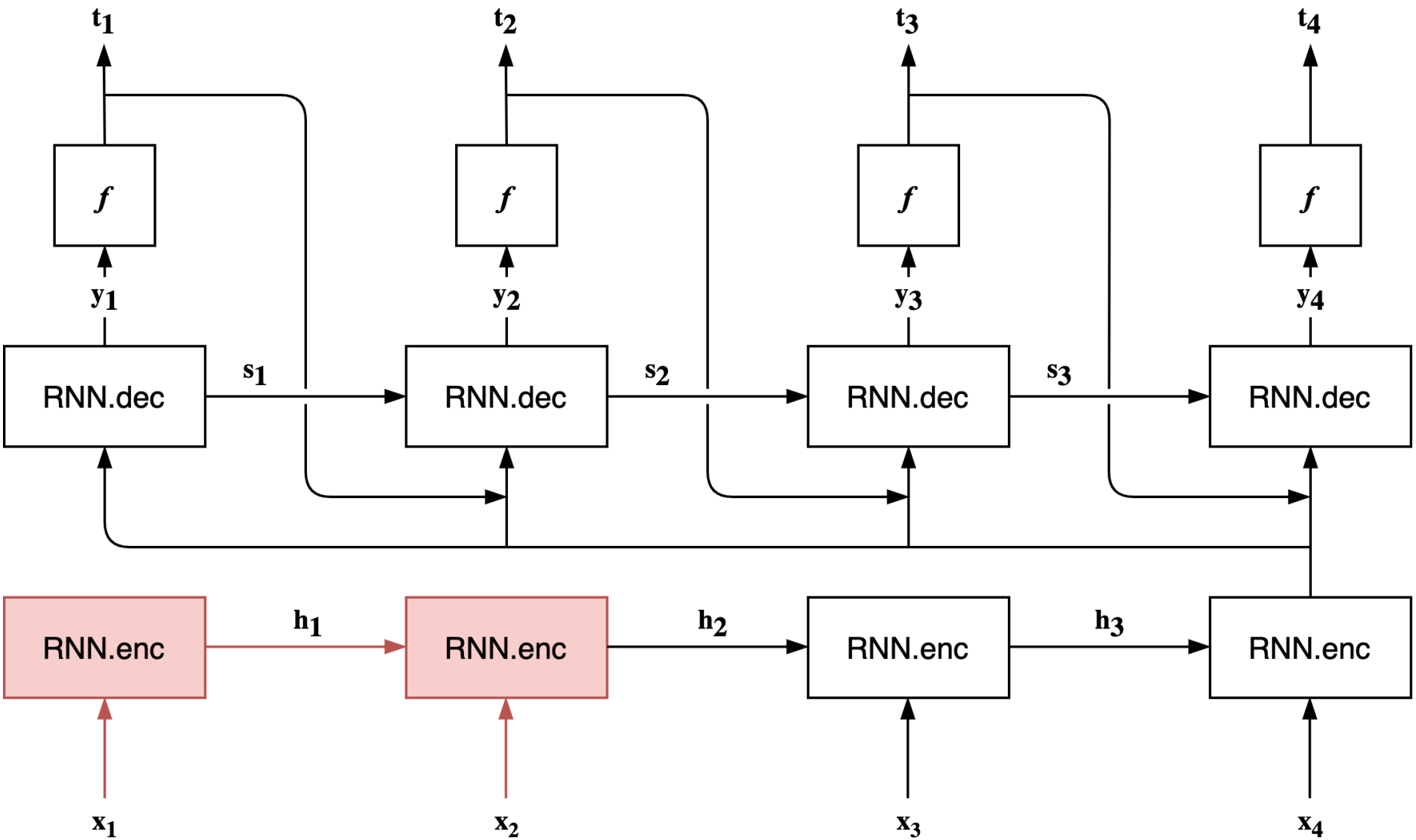
Example

We want to translate the sentence "Jeg liker fint vær" to "I like nice weather". What steps do we have to take?

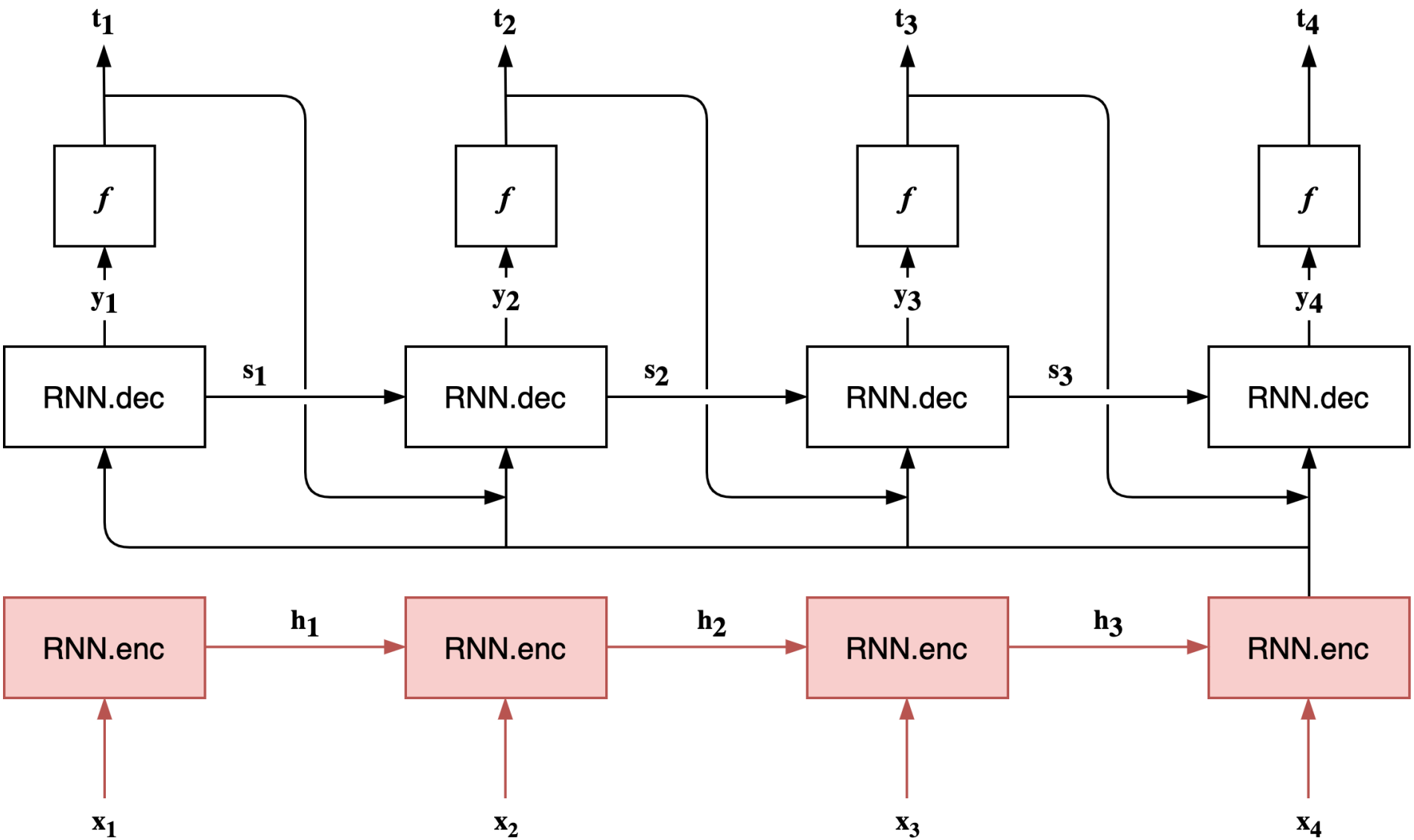
Encode "jeg"


t1
t2
t3
t4
f
f
f
f
y1
y2
y3
y4
RNN.dec
RNN.dec
RNN.dec
RNN.dec
s1
s2
s3
RNN.enc
RNN.enc
RNN.enc
RNN.enc
h1
h2
h3
x1
x2
x3
x4

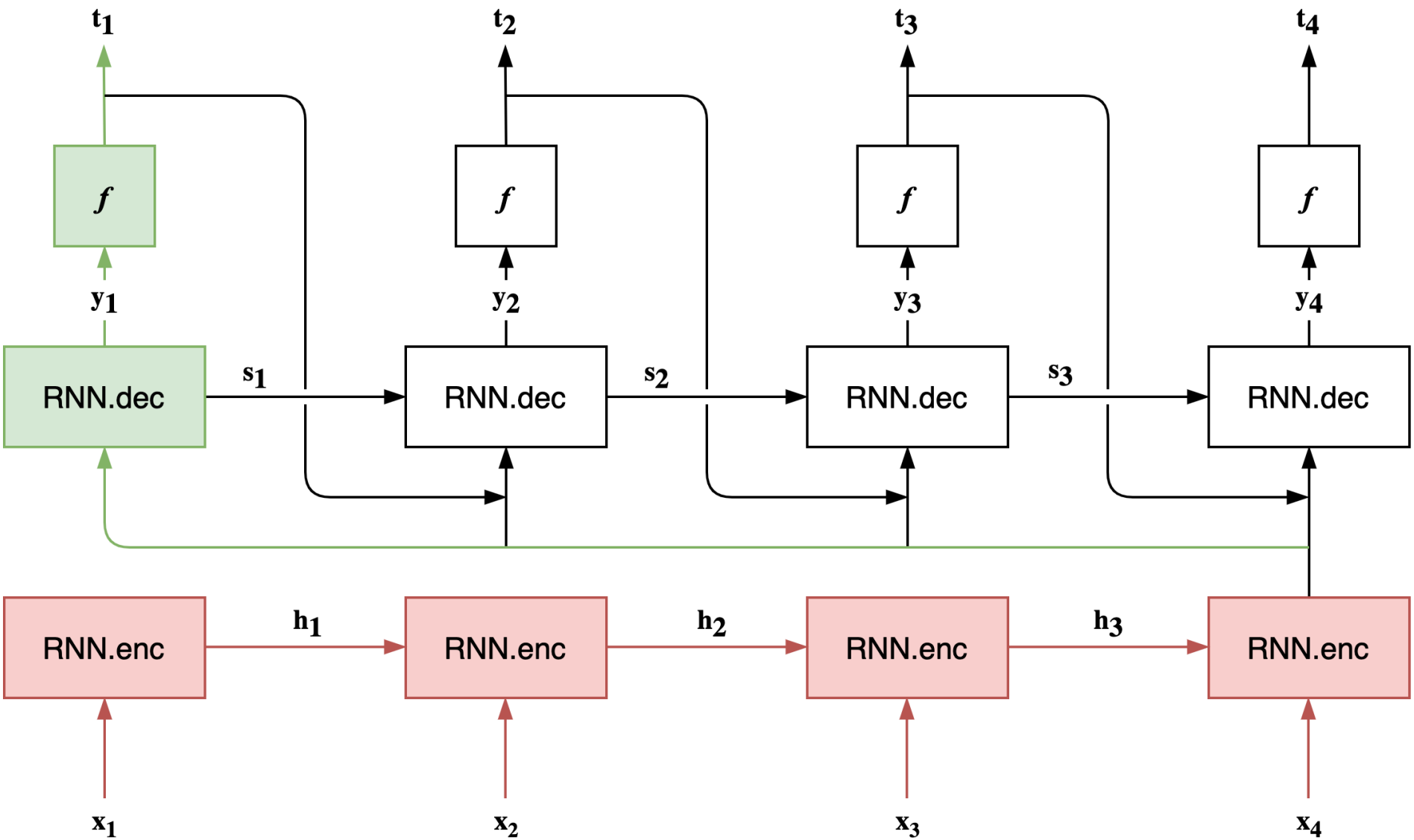
Encode "liker"



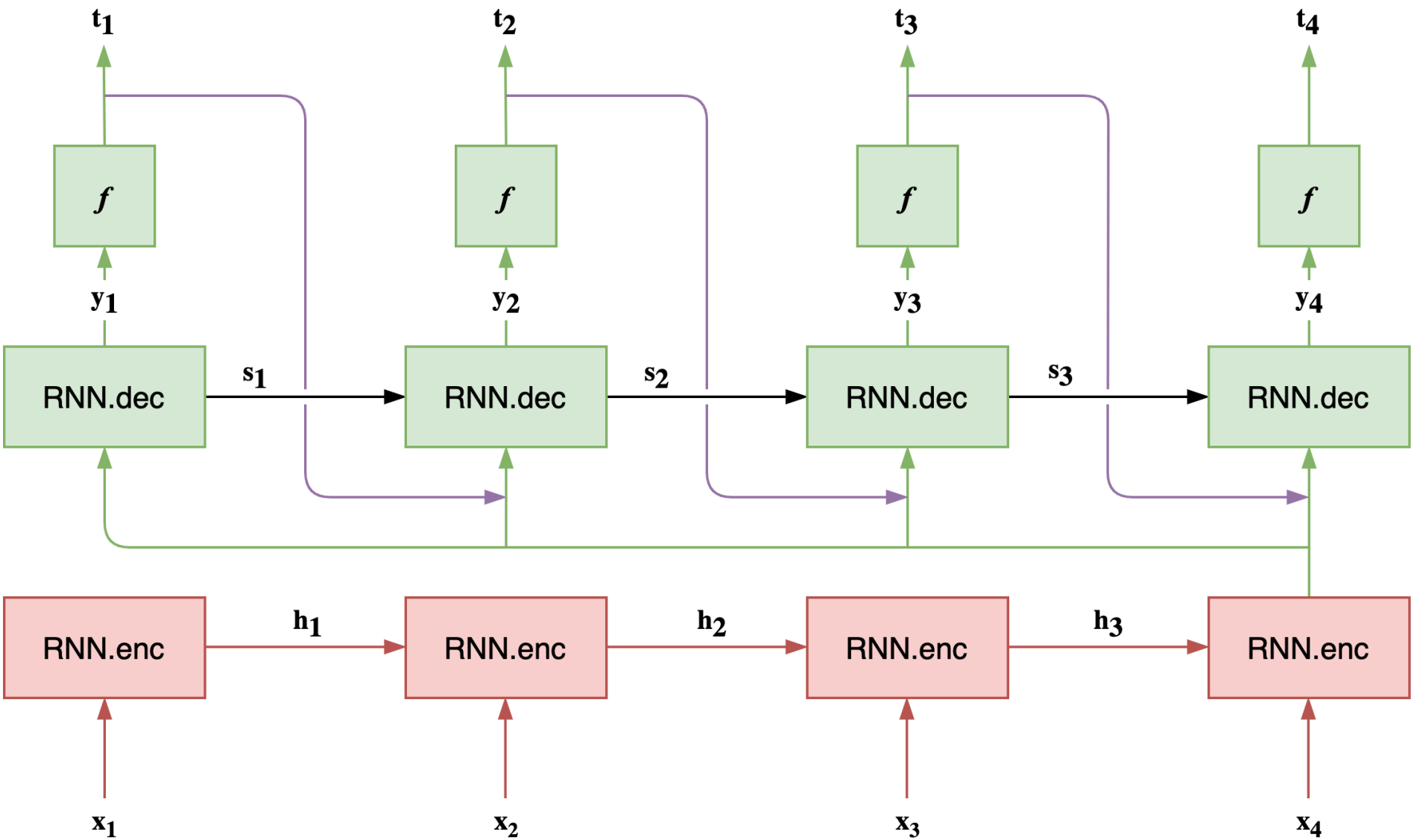
Encode..


t1
t2
t3
t4
f
f
f
f
y1
y2
y3
y4
RNN.dec
s1
RNN.dec
s2
RNN.dec
s3
RNN.dec
RNN.enc
h1
RNN.enc
h2
RNN.enc
h3
RNN.enc
x1
x2
x3
x4

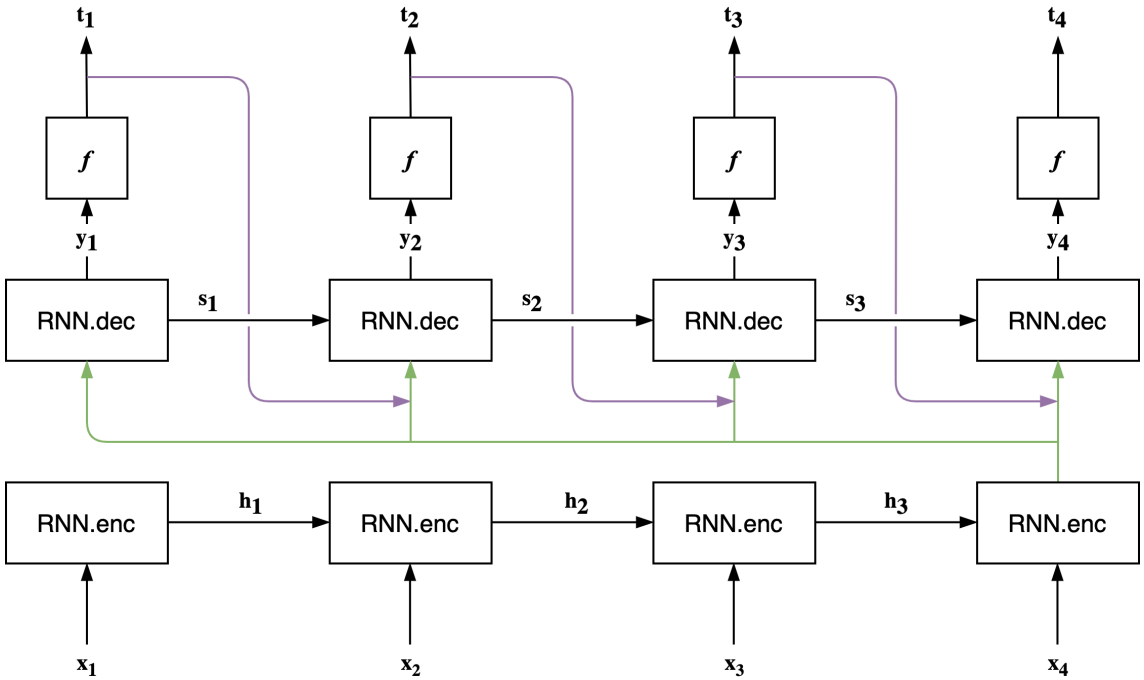
Decode "I"


t1
t2
t3
t4
f
f
f
f
y1
y2
y3
y4
RNN.dec
s1
RNN.dec
s2
RNN.dec
s3
RNN.dec
RNN.enc
h1
RNN.enc
h2
RNN.enc
h3
RNN.enc
x1
x2
x3
x4

Decode ...

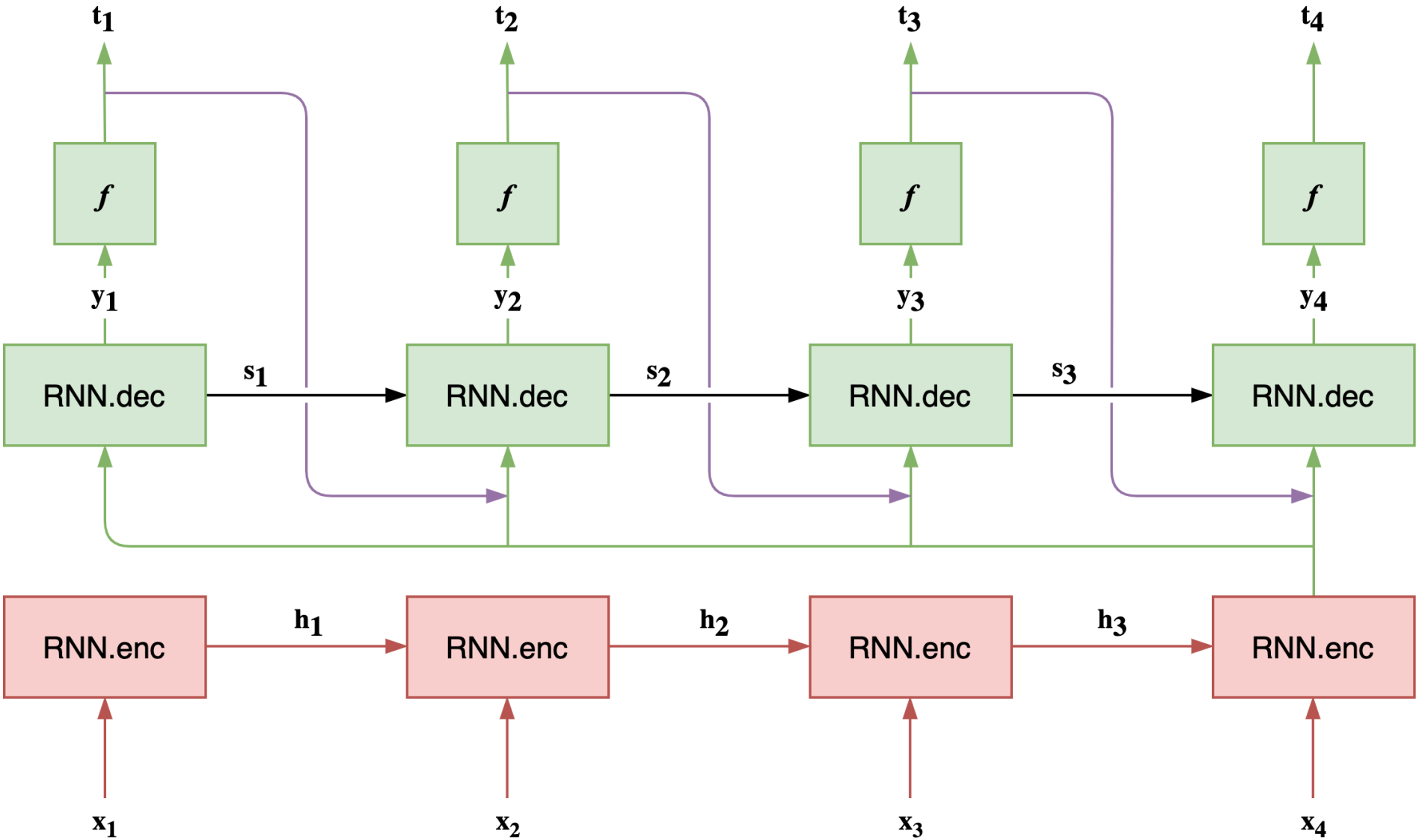

t1
t2
t3
t4
f
f
f
f
y1
y2
y3
y4
RNN.dec
s1
RNN.dec
s2
RNN.dec
s3
RNN.dec
RNN.enc
h1
RNN.enc
h2
RNN.enc
h3
RNN.enc
x1
x2
x3
x4

Encoder-Decoder Models with RNNs



- The generated output tokens are given by $p(t_{j+1}) = f(RNN([\hat{t}_j, c], s_j))$
- Softmax over the vocabulary to get the actual words
- The supervision signal is only given to the decoder, but the model is trained jointly
- For the first decoder step, we give a special <sos> token as the input, for the other steps the input is a concatenation of the previously generated token and $\mathbf{c}$.

Problem



There is a problem with this approach....Can you see where the weak spot is?

The weak spot

"You can't cram the meaning of a whole —ing sentence into a single —ing vector!"

– Raymond Mooney

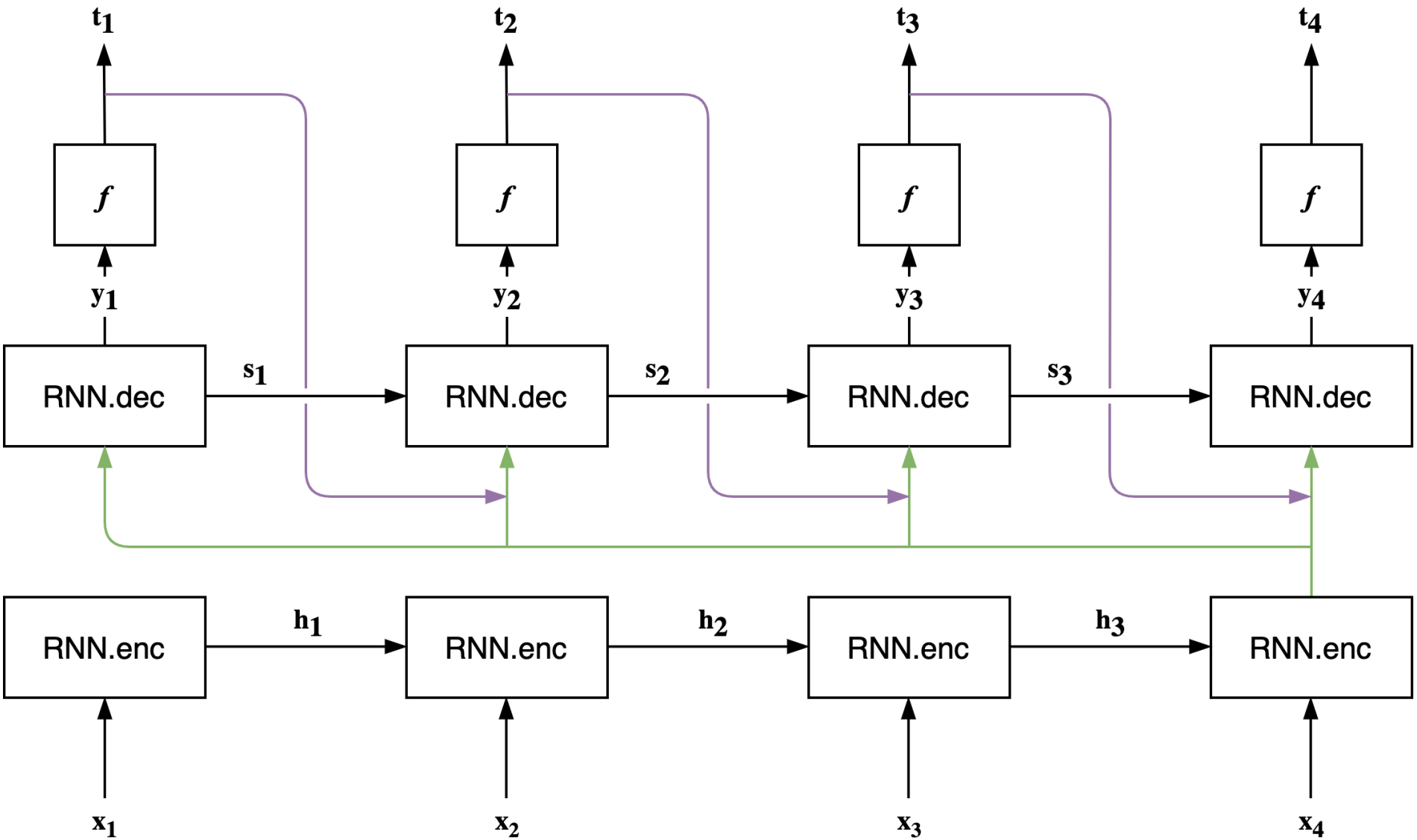
- We are encoding the entire input sequence into a single vector!
- Not only is this conceptually problematic, it also computationally intensive!

Enters Attention

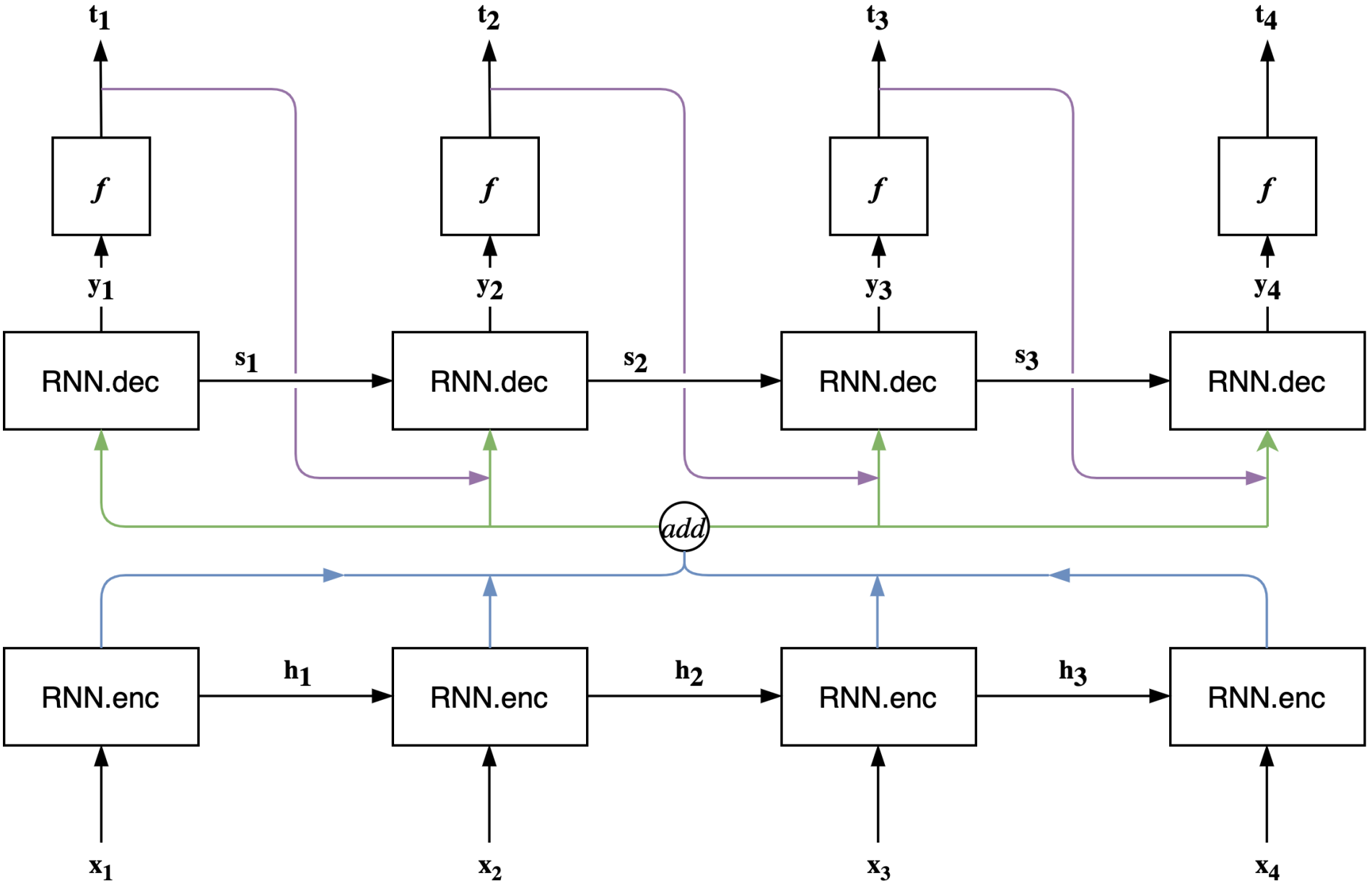
"The models proposed recently for neural machine translation often belong to a family of encoder-decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. . . . we conjecture that the use of a fixed-length vector is a bottleneck ... and propose to to extend this by allowing a model to automatically search for **parts of a source sentence that are relevant to predicting a target word**..."

[Bahdanau et al., 2015]

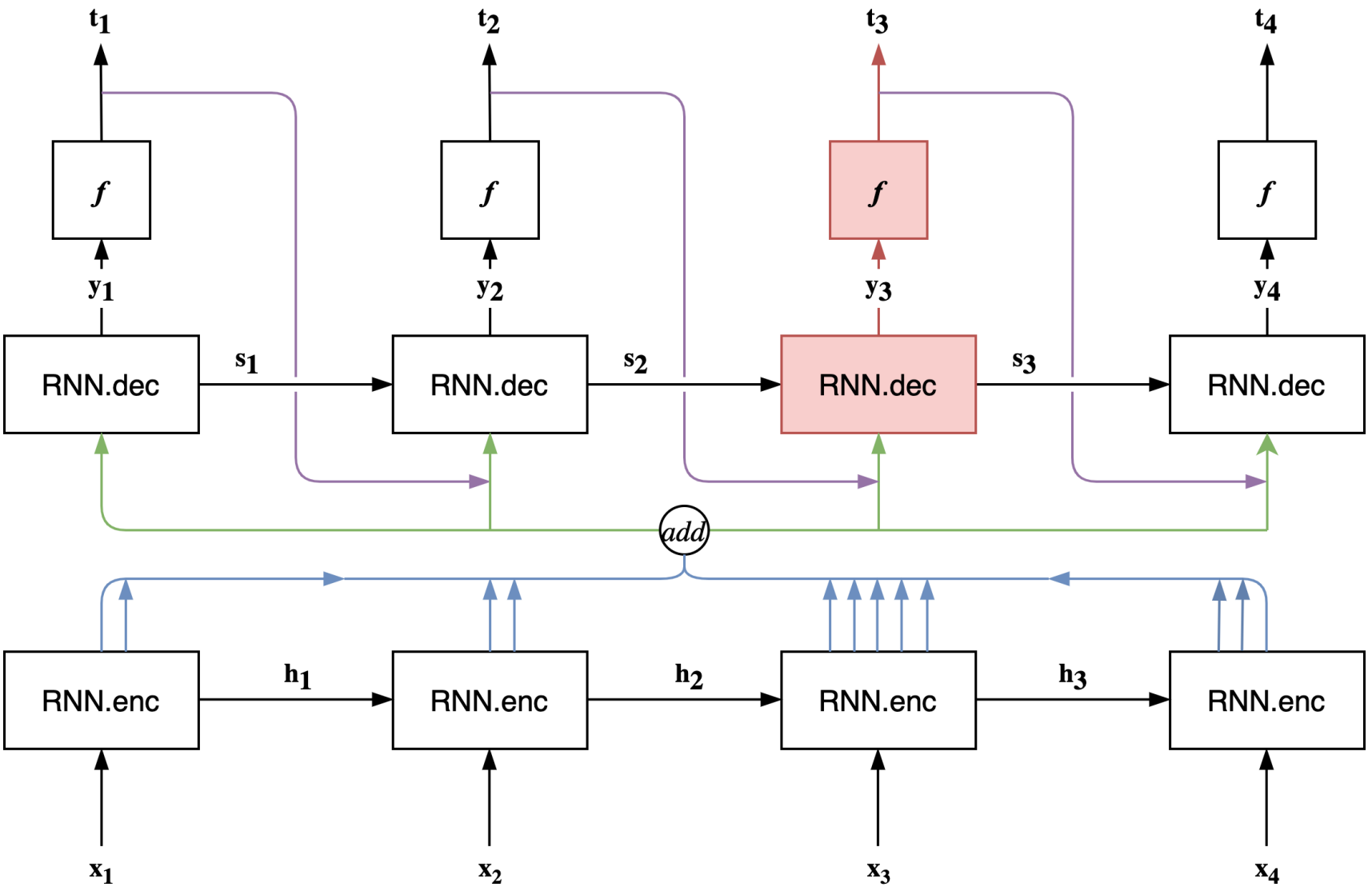
We now go from this:



To this:


t1
t2
t3
t4
f
f
f
f
y1
y2
y3
y4
RNN.dec
RNN.dec
RNN.dec
RNN.dec
s1
s2
s3
add
RNN.enc
RNN.enc
RNN.enc
RNN.enc
h1
h2
h3
x1
x2
x3
x4

Or to be more specific:



- We use a weighted sum of encoder hidden states for every decoder output state

Attention: explanation pass 1

- We want a way to let the generated outputs "attend", with different weight, to all the input tokens in the encoder.

Attention: explanation pass 1

- We want a way to let the generated outputs "attend", with different weight, to all the input tokens in the encoder.
- Instead of only passing on the last hidden state as the context vector, we pass *all* the hidden states from the encoder

Attention: explanation pass 1

- We want a way to let the generated outputs "attend", with different weight, to all the input tokens in the encoder.
- Instead of only passing on the last hidden state as the context vector, we pass *all* the hidden states from the encoder
- Then, in the decoder, for each time step:
  - We look at all of these hidden states from the encoder

Attention: explanation pass 1

- We want a way to let the generated outputs "attend", with different weight, to all the input tokens in the encoder.
- Instead of only passing on the last hidden state as the context vector, we pass *all* the hidden states from the encoder
- Then, in the decoder, for each time step:
  - We look at all of these hidden states from the encoder
  - Assign each of them a score based on how relevant they are for the token we are about to generate

Attention: explanation pass 1

- We want a way to let the generated outputs "attend", with different weight, to all the input tokens in the encoder.
- Instead of only passing on the last hidden state as the context vector, we pass *all* the hidden states from the encoder
- Then, in the decoder, for each time step:
  - We look at all of these hidden states from the encoder
  - Assign each of them a score based on how relevant they are for the token we are about to generate
  - We softmax the scores

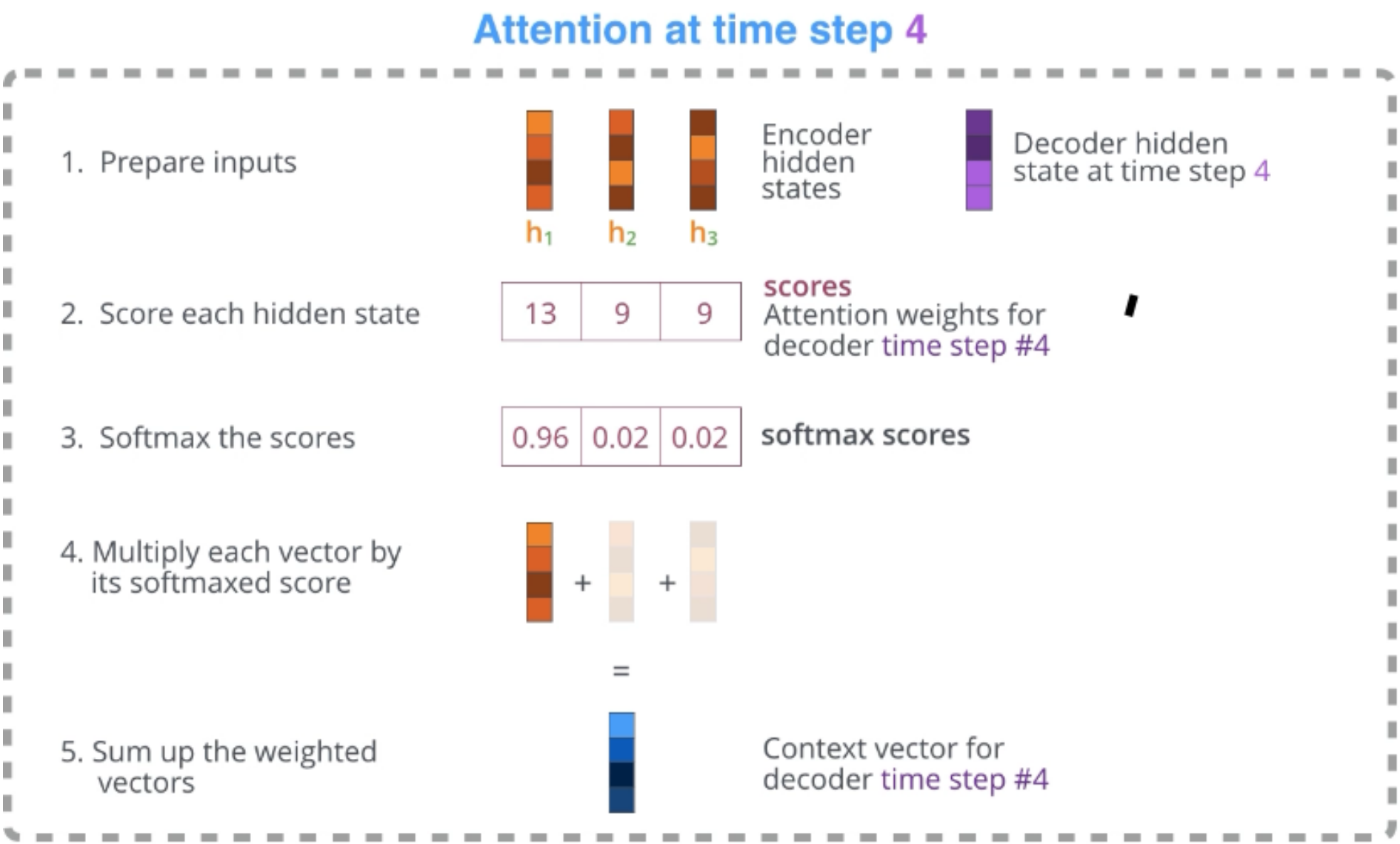
Attention: explanation pass 1

- We want a way to let the generated outputs "attend", with different weight, to all the input tokens in the encoder.
- Instead of only passing on the last hidden state as the context vector, we pass *all* the hidden states from the encoder
- Then, in the decoder, for each time step:
  - We look at all of these hidden states from the encoder
  - Assign each of them a score based on how relevant they are for the token we are about to generate
  - We softmax the scores
  - We multiply the hidden states with this softmaxed score, essentially "scaling" its importance up or down.

Attention: explanation pass 1

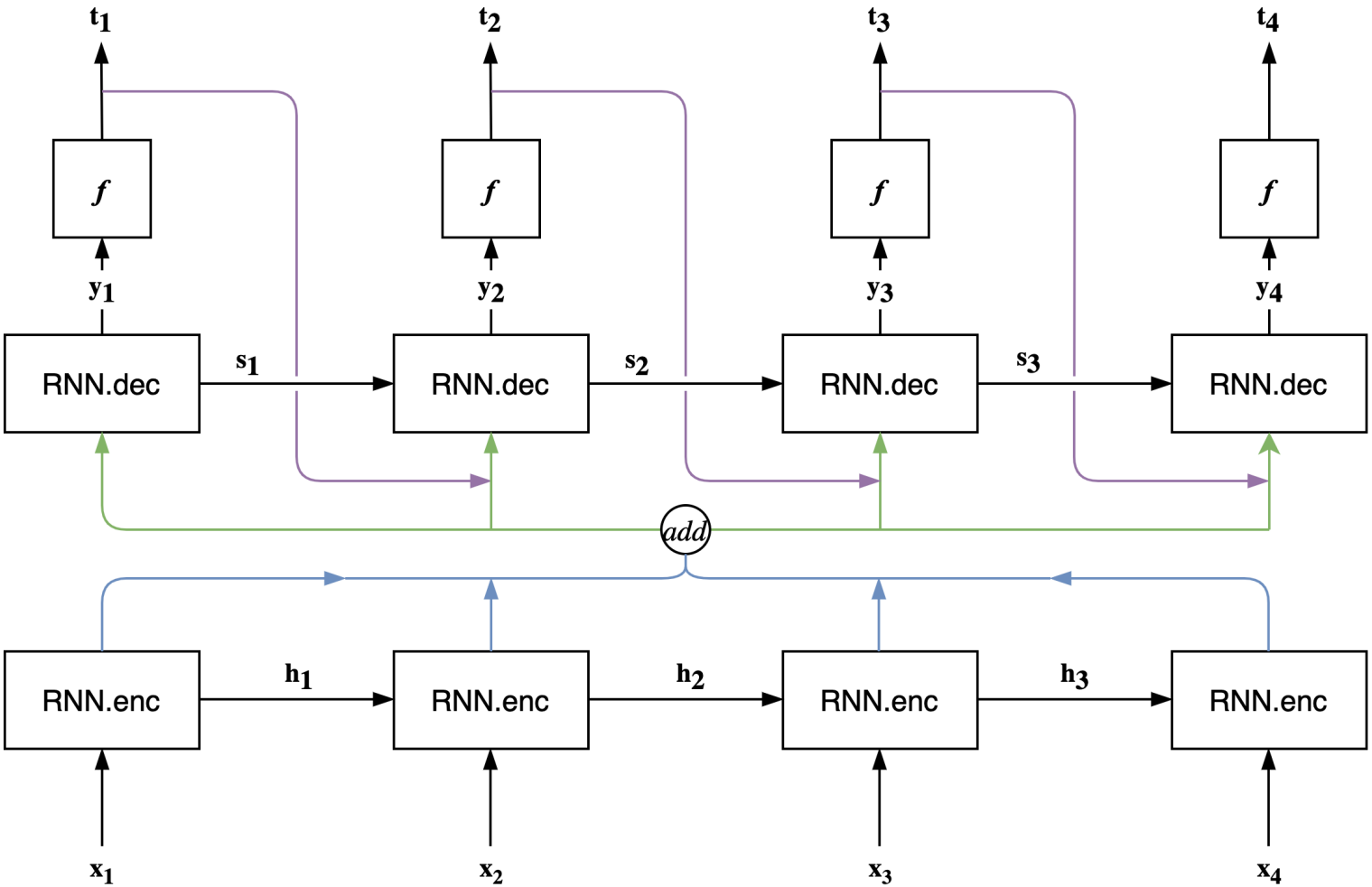
- We want a way to let the generated outputs "attend", with different weight, to all the input tokens in the encoder.
- Instead of only passing on the last hidden state as the context vector, we pass *all* the hidden states from the encoder
- Then, in the decoder, for each time step:
  - We look at all of these hidden states from the encoder
  - Assign each of them a score based on how relevant they are for the token we are about to generate
  - We softmax the scores
  - We multiply the hidden states with this softmaxed score, essentially "scaling" its importance up or down.
  - We then sum up these weighted hidden states from the encoder in order to form the new contex vector for the time step

Attention: explanation pass 1



(Figure from Jay Alammars blog, link at the end of the slide deck)

So now...



- The context, here represented with the green line, is now this weighted sum

Attention: explanation pass 2

- Previously, $p(t_{j+1}) = f(RNN([\hat{t}_j, c], s_j))$, with c being constant across the sequence

Attention: explanation pass 2

- Previously, $p(t_{j+1}) = f(RNN([\hat{t}_j, c], s_j))$, with c being constant across the sequence
- With attention, we use a separate context c for every output element $c_{j:n}$

Attention: explanation pass 2

- Previously, $p(t_{j+1}) = f(RNN([\hat{t}_j, c], s_j))$, with c being constant across the sequence
- With attention, we use a separate context c for every output element $c_{j:n}$
- c_j is a weighted sum of of the encoder hidden states:
  - $c_j = \sum_{i=1}^{T_x} \alpha_{ij} h_i$

Attention: explanation pass 2

- Previously, $p(t_{j+1}) = f(RNN([\hat{t}_j, c], s_j))$, with c being constant across the sequence
- With attention, we use a separate context c for every output element $c_{j:n}$
- c_j is a weighted sum of of the encoder hidden states:
  - $c_j = \sum_{i=1}^{T_x} \alpha_{ij} h_i$
- α_{ij} is the softmaxed scores:
  - $\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$

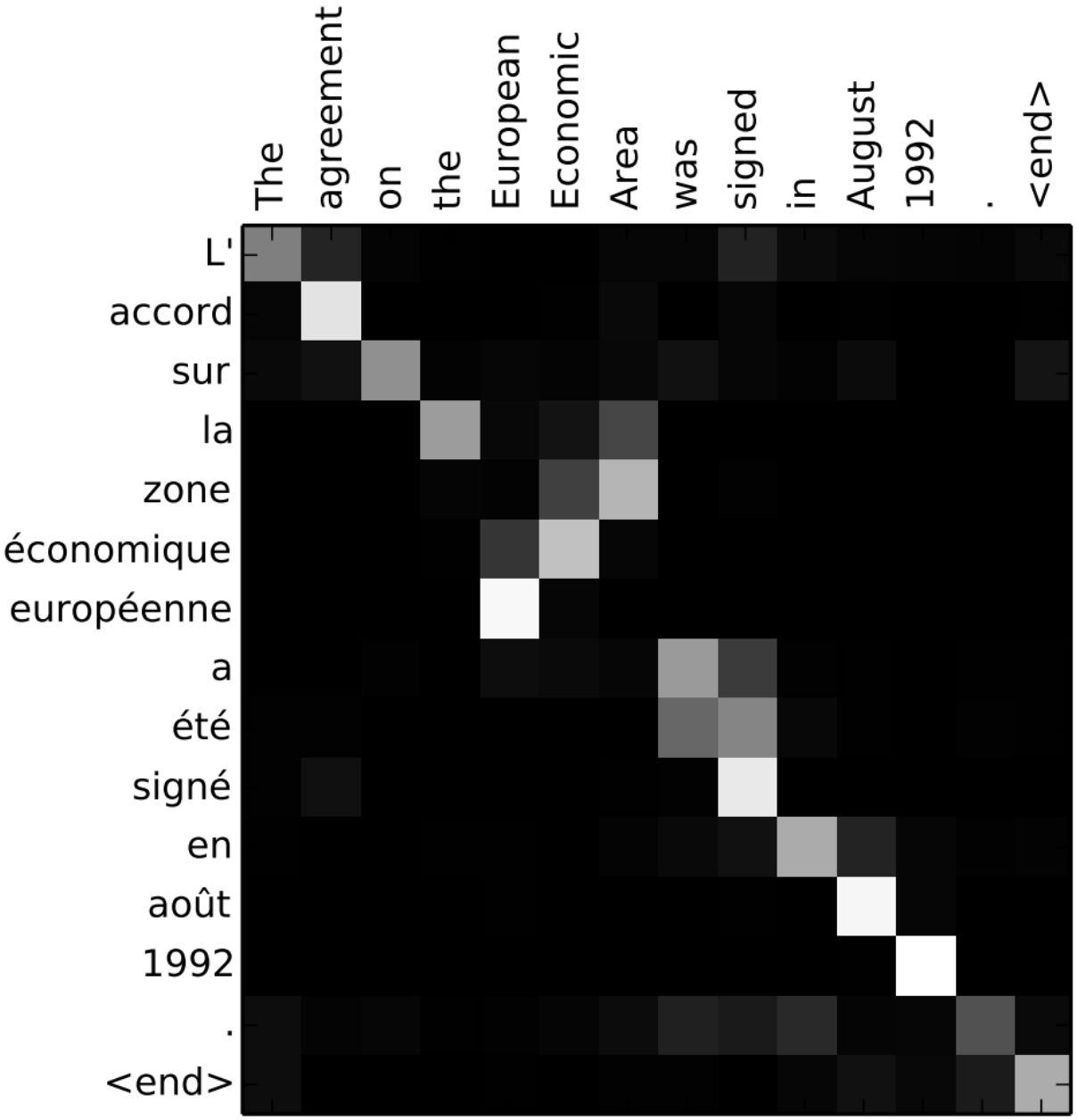
Attention: explanation pass 2

- Previously, $p(t_{j+1}) = f(RNN([\hat{t}_j, c], s_j))$, with c being constant across the sequence
- With attention, we use a separate context c for every output element $c_{j:n}$
- c_j is a weighted sum of of the encoder hidden states:
  - $c_j = \sum_{i=1}^{T_x} \alpha_{ij} h_i$
- α_{ij} is the softmaxed scores:
  - $\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$
- But the scores!! What are they?!

Attention: explanation pass 2

- The scores are learned, of course!
- That is, we have a scoring function, an *alignment model*:
  - $e_{ij} = a(s_{i-1}, h_j)$
  - a is a function that tells how relevant an input state i is to an output token j
- In [Bahdanau et al., 2015]: "we parametrize the alignment model a as a feedforward neural network which is jointly trained with all the other components of the proposed system".
- A FFNN that takes as input the concatenation of the decoder RNN hidden state at i, s_{i-1}, and the encoder hidden state at position j.
- Be aware: Different ways of calculating attention!

Added bonus: attention weights!



Useful resources

- Illustrative Blog Post, by Jay Alammar:
- `https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq`

References I


Bahdanau, D., Cho, K., and Bengio, Y. (2015).
Neural machine translation by jointly learning to align and translate.
In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.