

Chapter 6

# **Sequence to Sequence Models and Attention**

# Content of this Chapter

1. Machine Translation
2. Sequence-to-Sequence Models
  1. From RNN to Seq2Seq
  2. Seq2Seq model
  3. Enhancements
3. Evaluating Machine Translation

# Machine Translation

# Machine Translation

- Task in Machine Translation (MT):
  - Given a piece of text in a **source language**...
  - ... translate the text to a different **target language**

Sindarin:



English:

Pedo mellon a minno!



Say friend and enter!

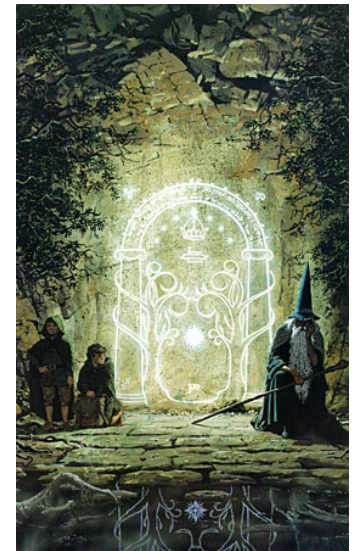



Image: Ted Nasmith

# Machine Translation: History

- Early Machine Translation research in the 1950s
  - Mostly Russian → English (Cold War!)
  - Mostly **rule-based systems**:
    - Use bilingual dictionaries to translate words
    - Use rules to fix word order
  - Yehoshua Bar-Hillel: „MT for natural languages will never work!“  
Example:  
Little John was looking for his toy box. Finally he found it. The box was in the pen.

„Pen“ can mean either a  
writing device or a container!



# Machine Translation: History

- 1990s – 2010s: Statistical Machine Translation
  - Core idea: Learn a probabilistic model from training data
  - Given a sentence  $x$  in the source language, find the „best“ sentence  $\hat{y}$  in the target language:

$$\hat{y} = \operatorname{argmax}_y P(y|x)$$

- How to model this?
  - Break down into two components using Bayes:

$$\hat{y} = \operatorname{argmax}_y P(x|y)P(y)$$

**Translation model:**

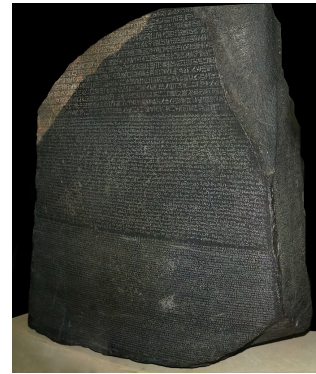
How likely does  $x$  result  
from translating  $y$ ?

**Language model:**

How likely does  $y$  occur  
in the target language?

# Machine Translation: History

- 1990s – 2010s: Statistical Machine Translation
  - How to learn the translation model?
  - Needed:
    - Large datasets of parallel text



Rosetta Stone,  
[https://commons.wikimedia.org/wiki/File:Rosetta\\_Stone.JPG](https://commons.wikimedia.org/wiki/File:Rosetta_Stone.JPG)

- Alignment between languages

Sindarin:



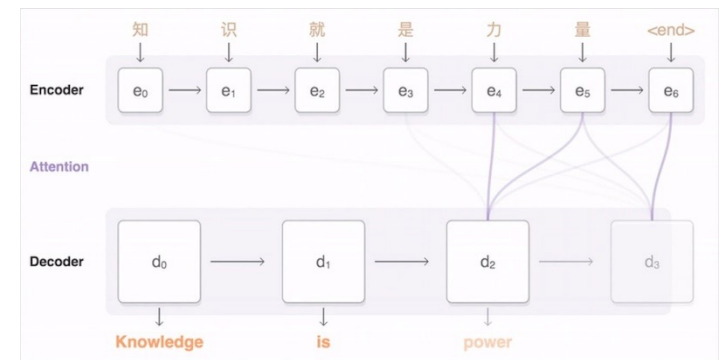
English:

Mae govannen, mellon nin!

↓ ↓ ↓ ↓  
Hello, my friend!

# Machine Translation: History

- Now: Neural Machine Translation (NMT)!
- Since everything is neural now, why not machine translation, too?
- 2014: Bahdanau et al: Details later!
  - Introduce **Attention** mechanism
  - Beat Statistical Machine Translation using Neural Networks!
- Advantages of NMT:
  - End-to-end model
  - No need for manual feature engineering
  - No language specific properties

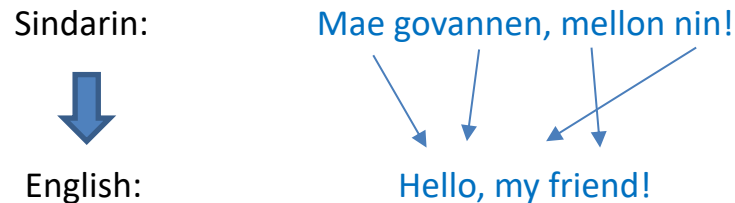


<https://github.com/google/seq2seq>



# Neural Machine Translation

- How to build a neural network for Machine Translation?
  - RNNs seem like a natural choice
    - Sequences as input and sequences as output
    - But: some problems that we need to address to get better translations!
      - Implicit model for alignment



- Performance deteriorates for long sentences

Aragorn Arathornion Edhelharn, aran Gondor ar Hir i Mbair Annui, anglennatha I  
Varanduiniant erin dolothen Ethuil, egor ben genediad Drannail erin Gwirth edwen

↓

Aragorn, Arathorn's son, Elfstone, King of Gondor and Lord of the Westlands  
.....?!?

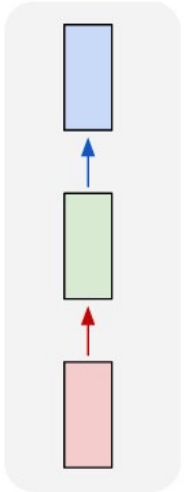
– ...

→ Use **Sequence to Sequence** models with some further tweaks!

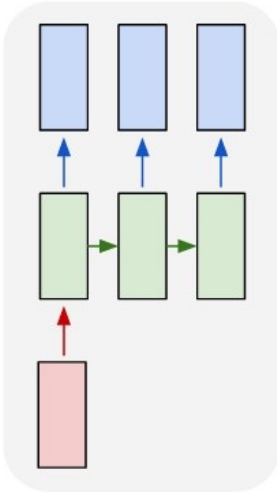
# Sequence to Sequence models (Seq2Seq)

# Recap types of RNN

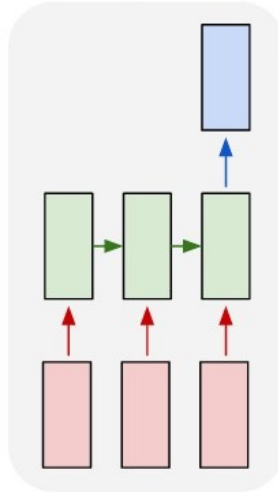
one to one



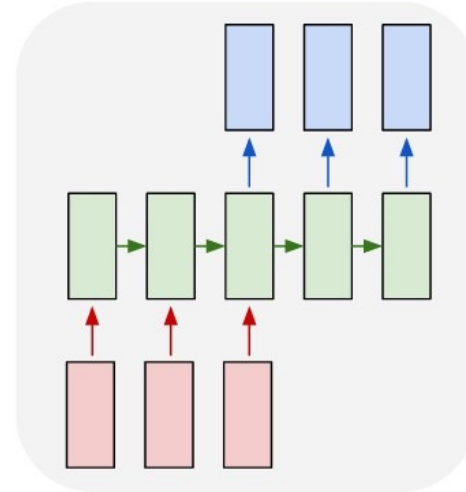
one to many



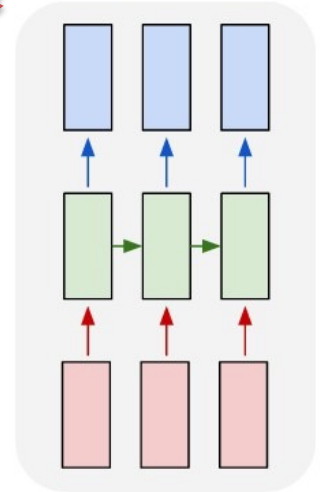
many to one



many to many



many to many



Seq2Seq

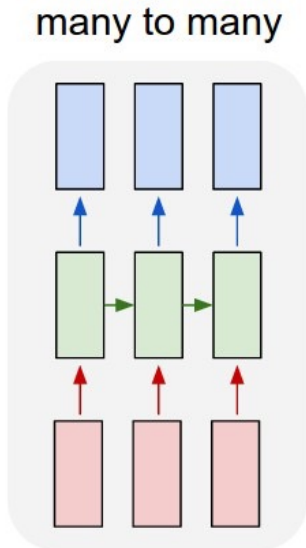
Fully Connected  
Network

Network mapping a  
sequence to one output

Network returning an  
output after each timestep  
of a sequence

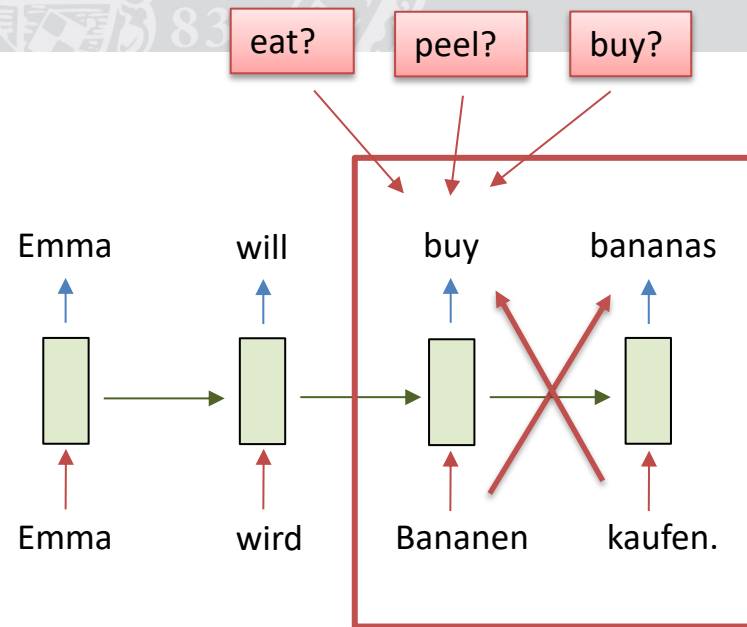
Recurrent Neural Networks

# Single RNN



## Idea:

Use the RNN output at every timestep to predict the translated word.

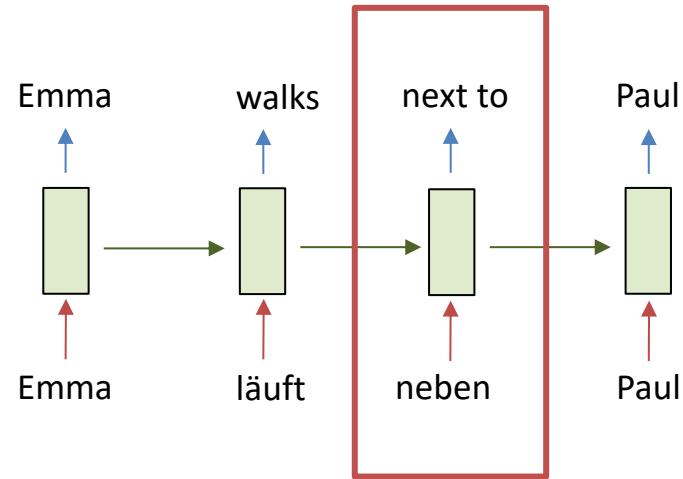
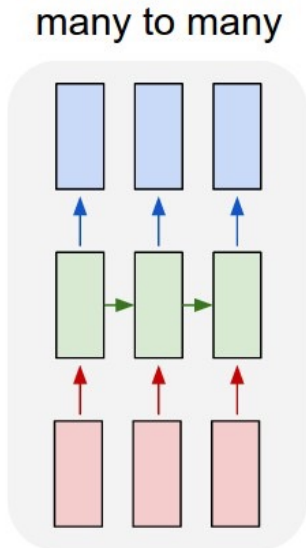


## Problem:

The word order differs between translated sentences.

In the above case, the model can't know whether Emma is going to eat, peel or buy the banana!

# Single RNN



## Problem:

Usually input and target sentence do not have the same number of words.

## Idea:

Use the RNN output at every timestep to predict the translated word.

We can not accommodate source and target sentences of different lengths with a single RNN!

# From RNN to Seq2Seq

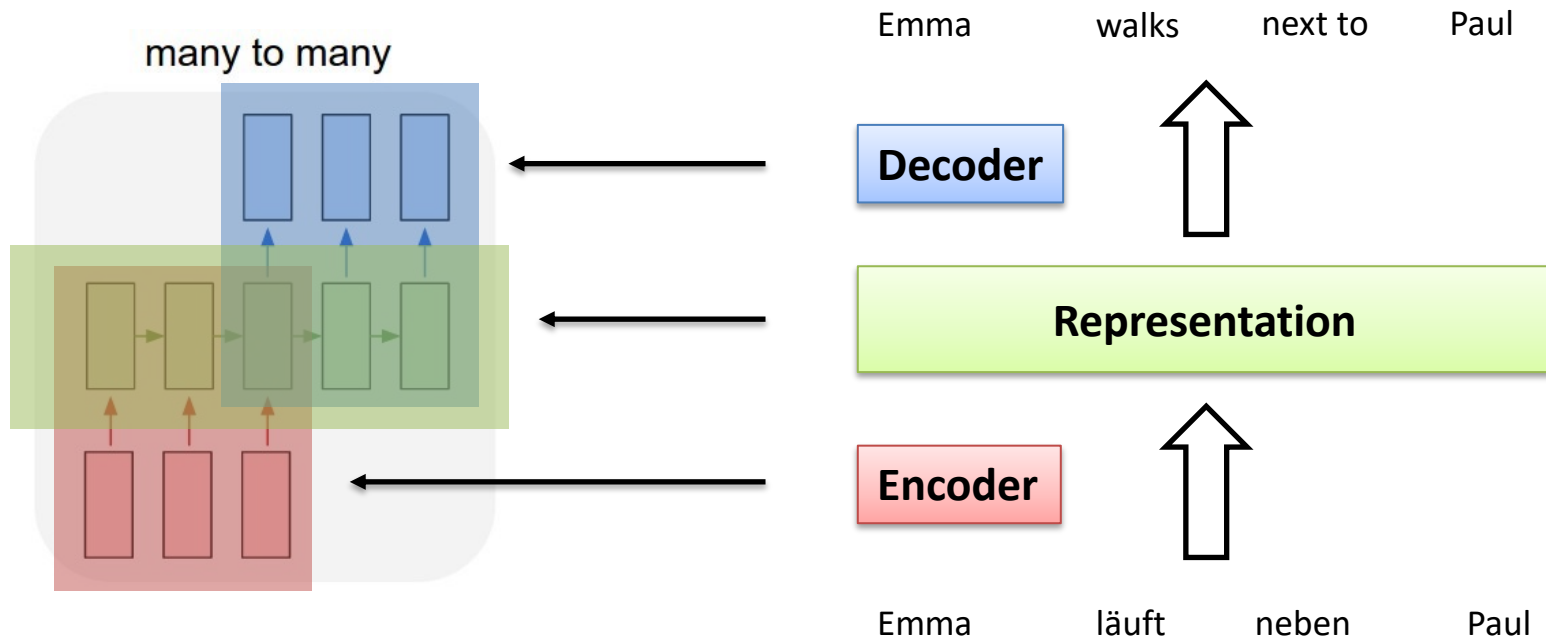
- Clearly, a simple RNN is not sufficient!
- How can we tweak the model to overcome these problems?

## Challenges:

1. Handle input and target sequences of different lengths
  - Use two distinct RNNs:
    - An **encoder** network maps the input to an internal representation  $h$
    - A **decoder** network generates a target sequence from  $h$
2. Generate target sequence based on the full input sequence
  - Encoder reads the **entire sentence** before passing it to the decoder

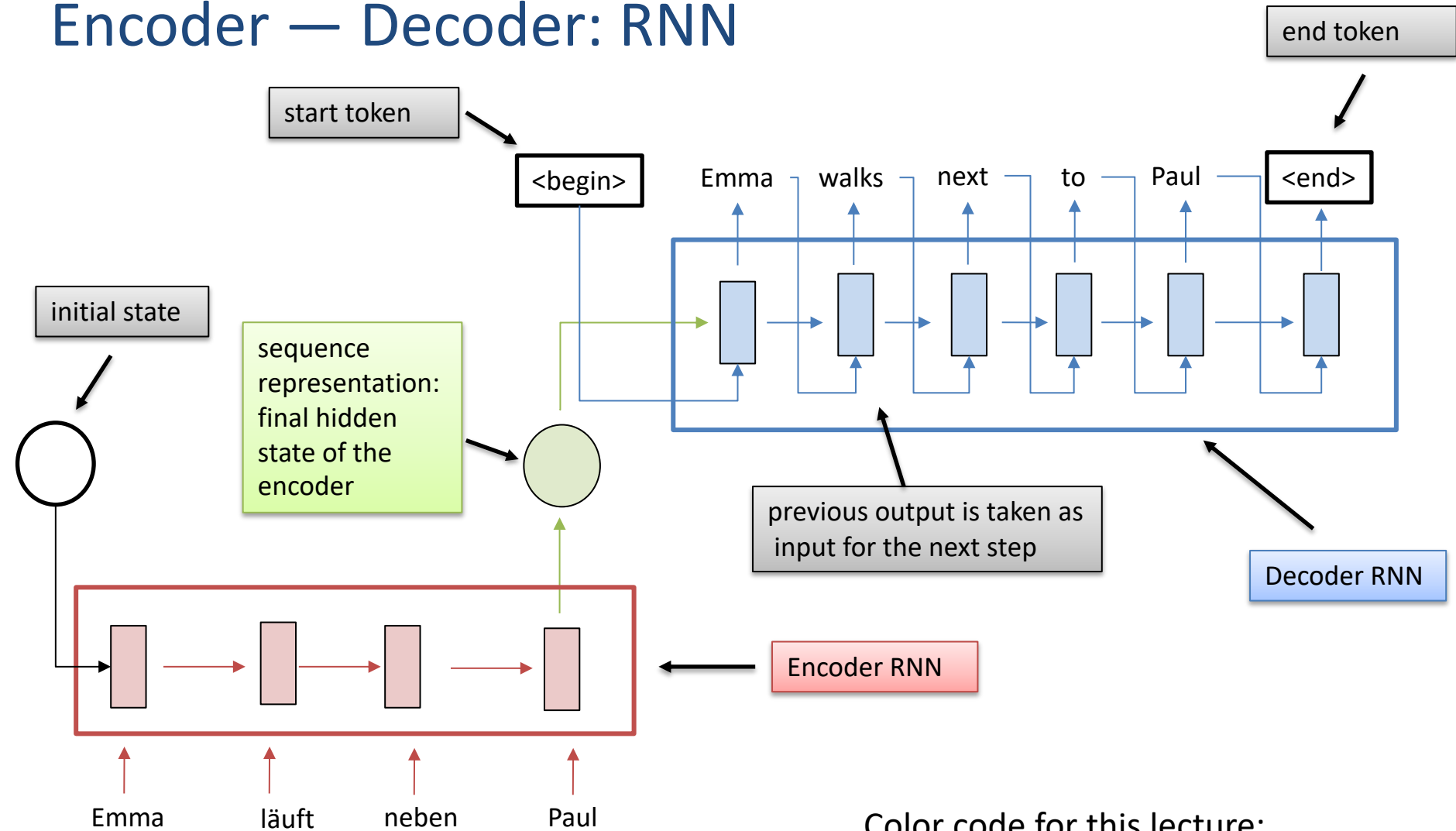
# Encoder — Decoder Model

- High level visualisation of an Encoder-Decoder architecture:



- Now let's look at the details!

# Encoder — Decoder: RNN



Color code for this lecture:  
Encoder — Representation — Decoder



# Encoder — Decoder Loss

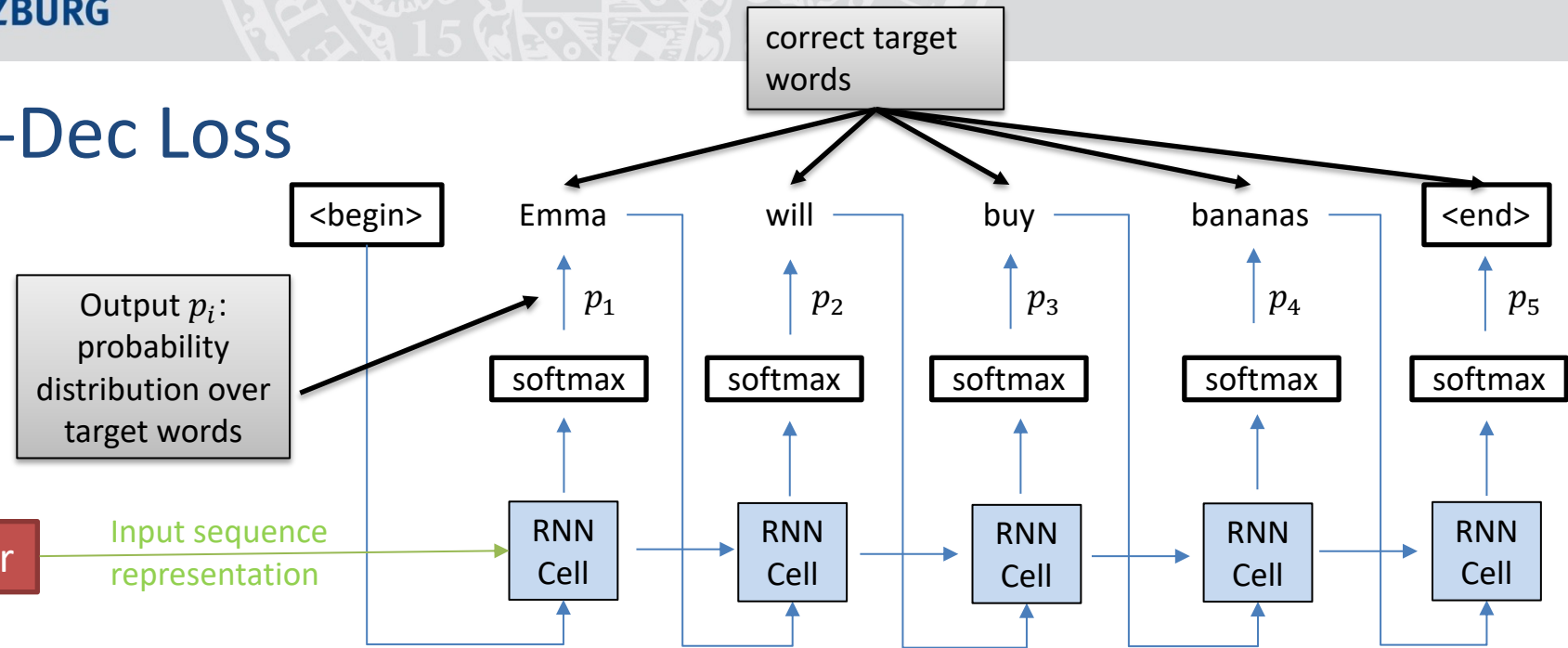
- To train the model, we need to define a loss function
- Since we have multiple outputs, we need to aggregate the loss somehow
- **Idea:** Use the probability the model assigns to the entire target sentence!

→ Probability of the target sentence  $(y_1, \dots, y_n)$ :

$$P(y_1, \dots, y_n) = \prod_{i=1}^n p_i[y_i]$$

Probability of target word  $y_i$  at  
timestep  $i$  according to the decoder

# Enc-Dec Loss



Probability of the target sentence according to the decoder:

$$P(y_1, \dots, y_n) = \prod_{i=1}^n p_i[y_i]$$

A good model should assign a high probability to the target translation!

This gives us the loss function:

$$L = -\log P(y_1, \dots, y_n) = -\log \prod_{i=1}^n p_i[y_i] = -\sum_{i=1}^n \log p_i[y_i]$$

# Improving Seq2Seq Models

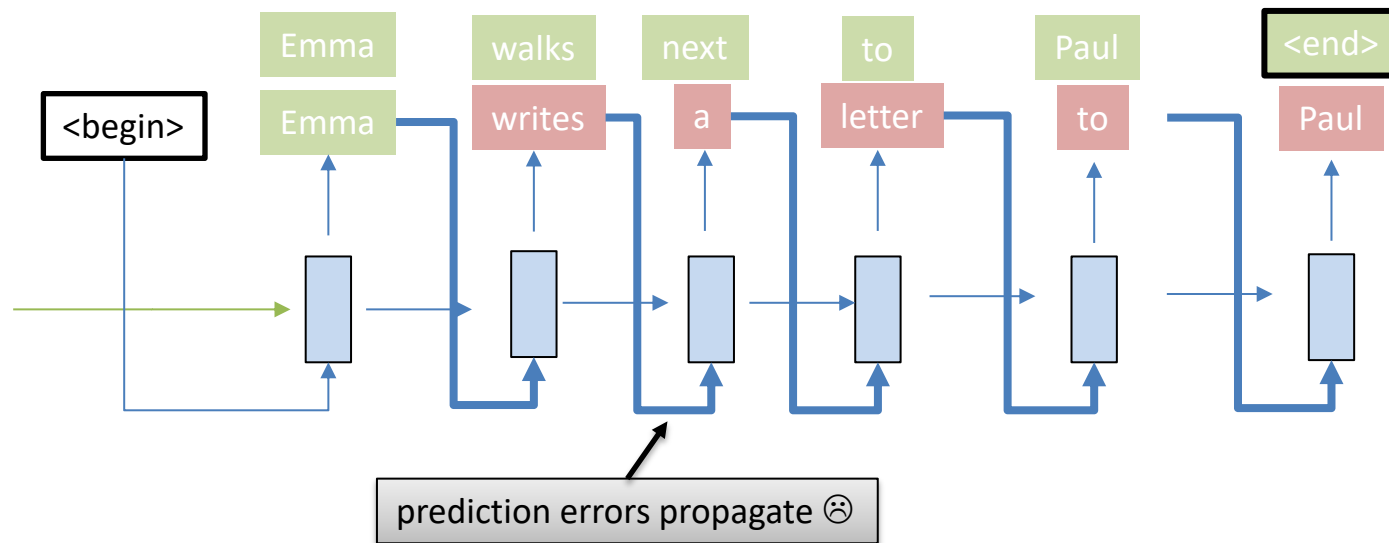
- That's basically all we need!
- We can now train Sequence to Sequence models with an Encoder-Decoder architecture
- Let's look at some optimisations to improve our results

No backprop today 😞

But you're welcome to do it yourself!

# Seq2Seq: Teacher Forcing

Remember: We use the output from the previous step as input to the next step.



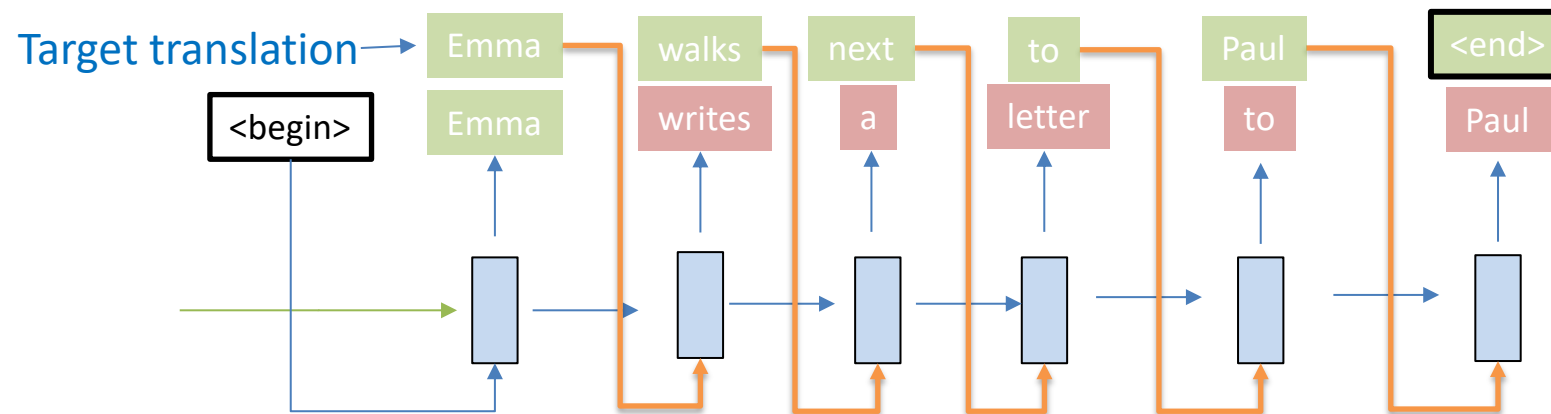
Prediction errors propagate as input for the next decoding step!  
This can slow down training or could even lead to a diverging model.

# Seq2Seq: Teacher Forcing

## Teacher Forcing:

During training, we already know the correct target words!

→ Feed those to the decoder instead of the previous prediction

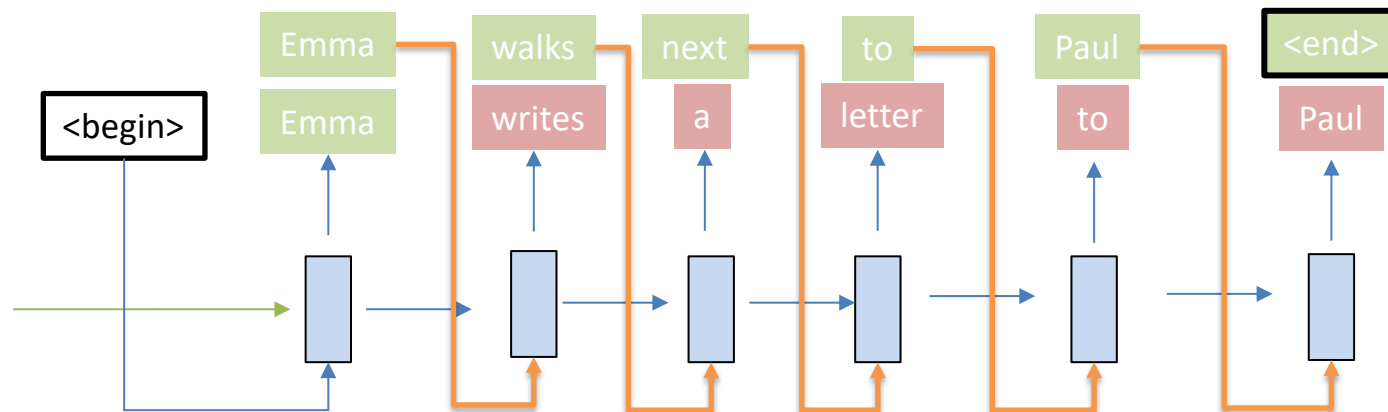


# Seq2Seq: Teacher Forcing

- Problem of Teacher Forcing:
  - During training, previous “output” is always correct  
→ Model relies too much on it!
  - During prediction, the network can not handle wrong inputs



Start training with teacher forcing, then switch to supplying the real outputs.

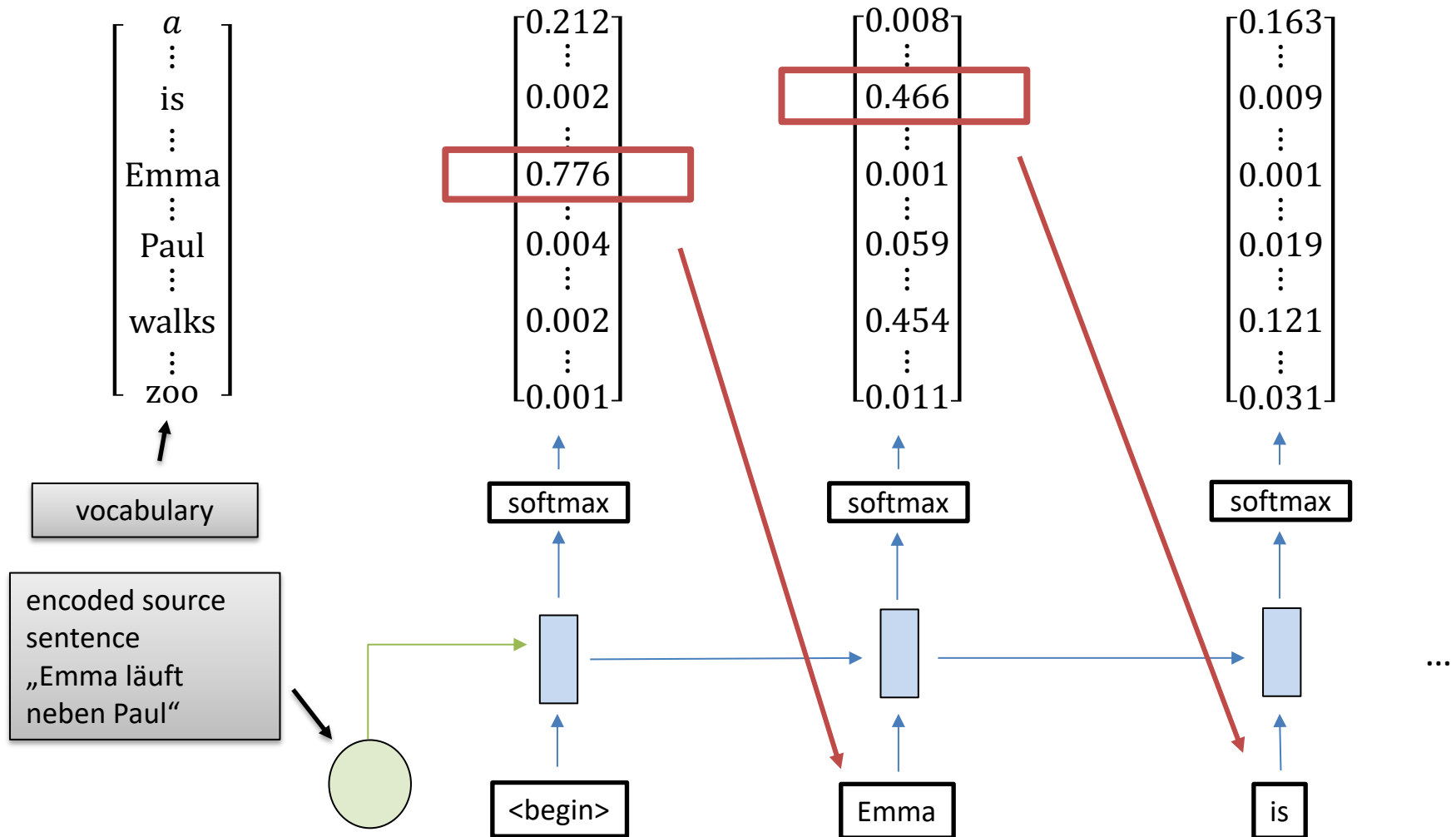


# Seq2Seq: Improved Sampling

- Remember:
  - Decoder returns a (local) probability distribution over target words at each timestep
  - We want to find the most likely sequence **globally**
- So far, we just selected the most likely word  
→ **Greedy Search**
- Alternative: Consider multiple options and decide later  
→ **Beam Search**

# Inference — Greedy Search

- At each step: use the token with the highest probability





# Inference — Beam Search

- At each step: Parameter needs to be selected/tuned!  
use the  $B$  sequences with the highest probabilities
- $B = 1 \rightarrow$  Greedy Search
- $B > 1$ : Give the model the chance to find a globally more likely translation by selecting a locally less likely word
- Still no guarantee for exact global maximum, unlike exact search algorithms like
  - Breadth First Search (BFS)
  - Depth First Search (DFS)

# Inference — Beam Search Example

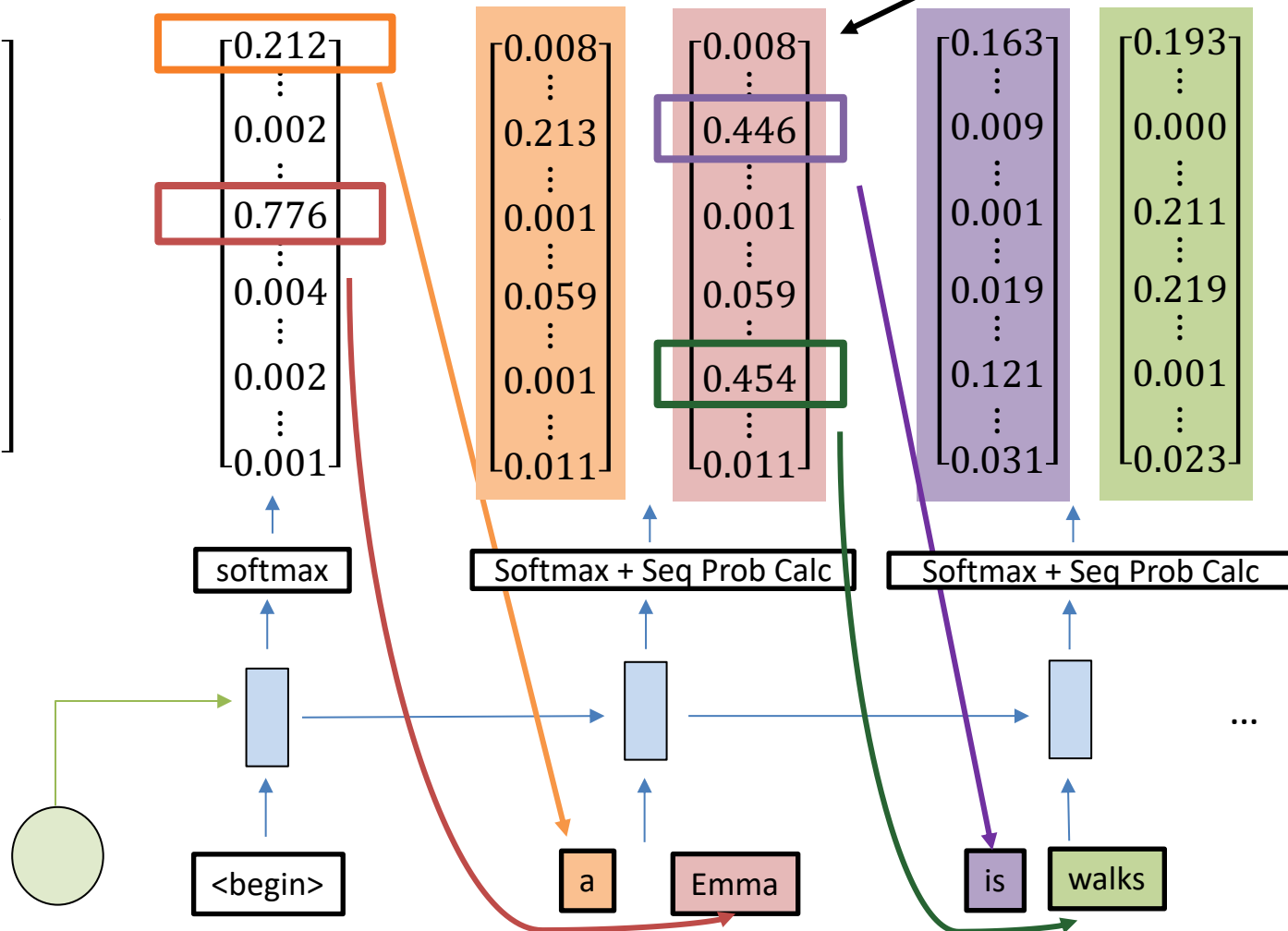
- $B = 2$

[<b> a], 0.212  
[<b>, Emma], 0.776

[<b>, Emma, is], 0.446  
[<b>, Emma, walks], 0.454

Probabilities of  
the sequences!

$\begin{bmatrix} a \\ \vdots \\ is \\ \vdots \\ Emma \\ \vdots \\ Paul \\ \vdots \\ walks \\ \vdots \\ zoo \end{bmatrix}$



# Inference

- Use Beam-Search whenever possible:
  - Leads to better results
  - Drawback: Slower than Greedy Search
- Tuning parameter  $B$  (number of sequences to consider):
  - $B$  large: better result, slower
  - $B$  small: worse result, faster
  - In practice:  $B \approx 10$
  - Adapt for your problem!

# Enhancing the Model

- Encoder – Decoder is a very general and flexible model
  - We can change individual parts separately to tune the model for a task.
- Let's take a look at some improvements!

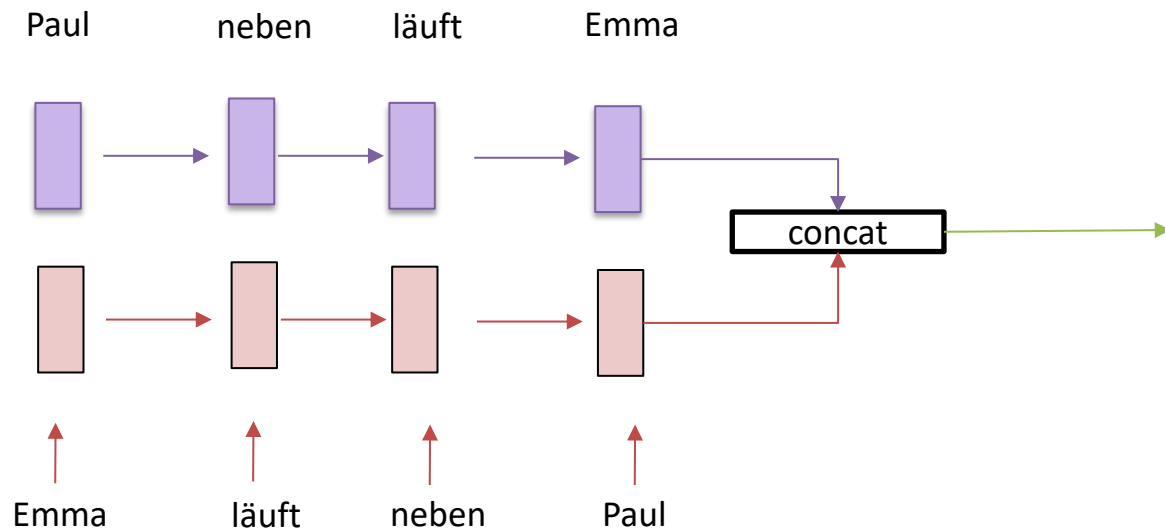
# Enhancement: BiRNN Encoder

## Problem:

Long sequences can cause the sequence representation to lose information about the first steps.

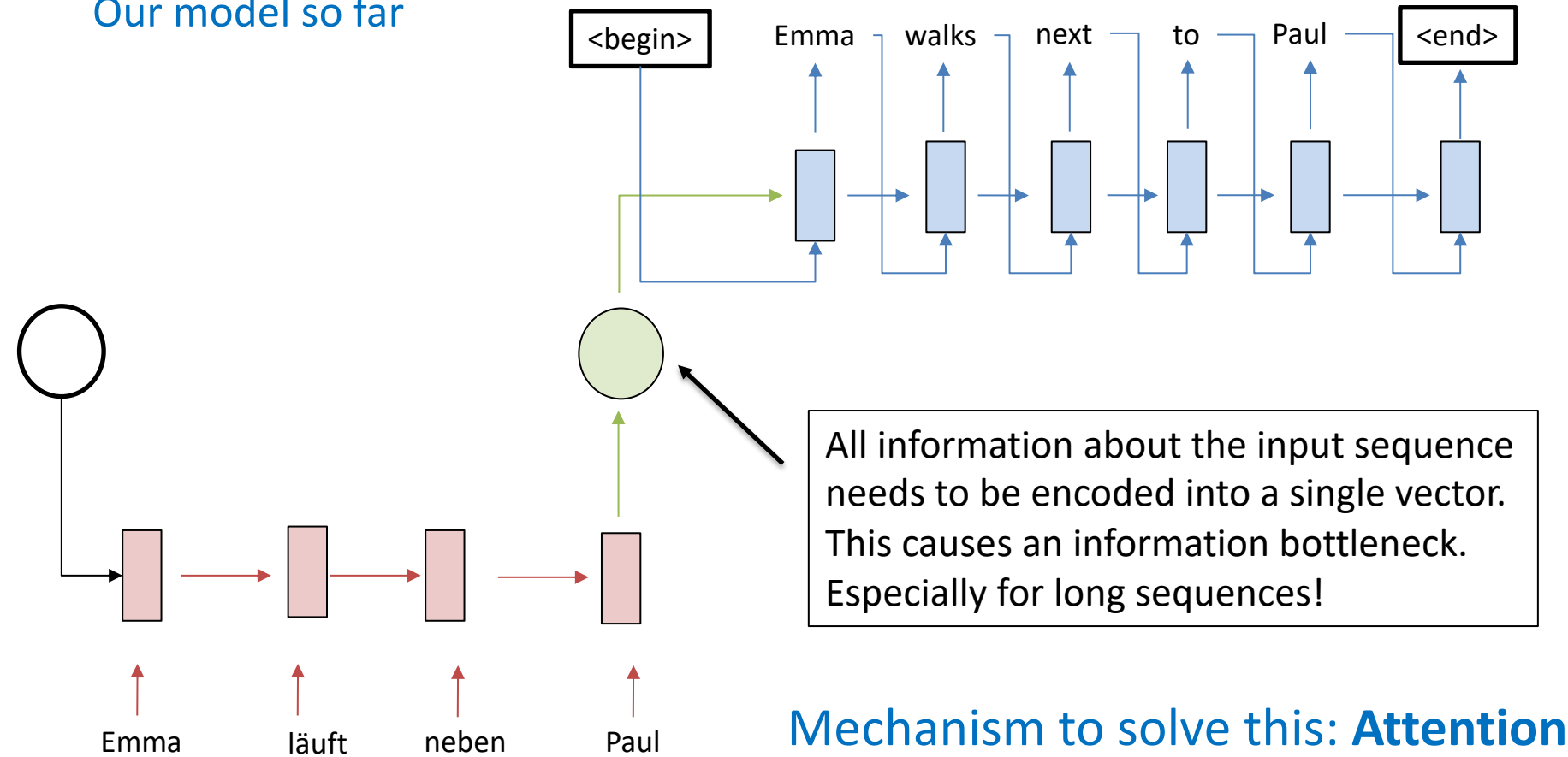
## Solution:

Additionally feed the reverse input sequence into a separate RNN and concatenate the output.

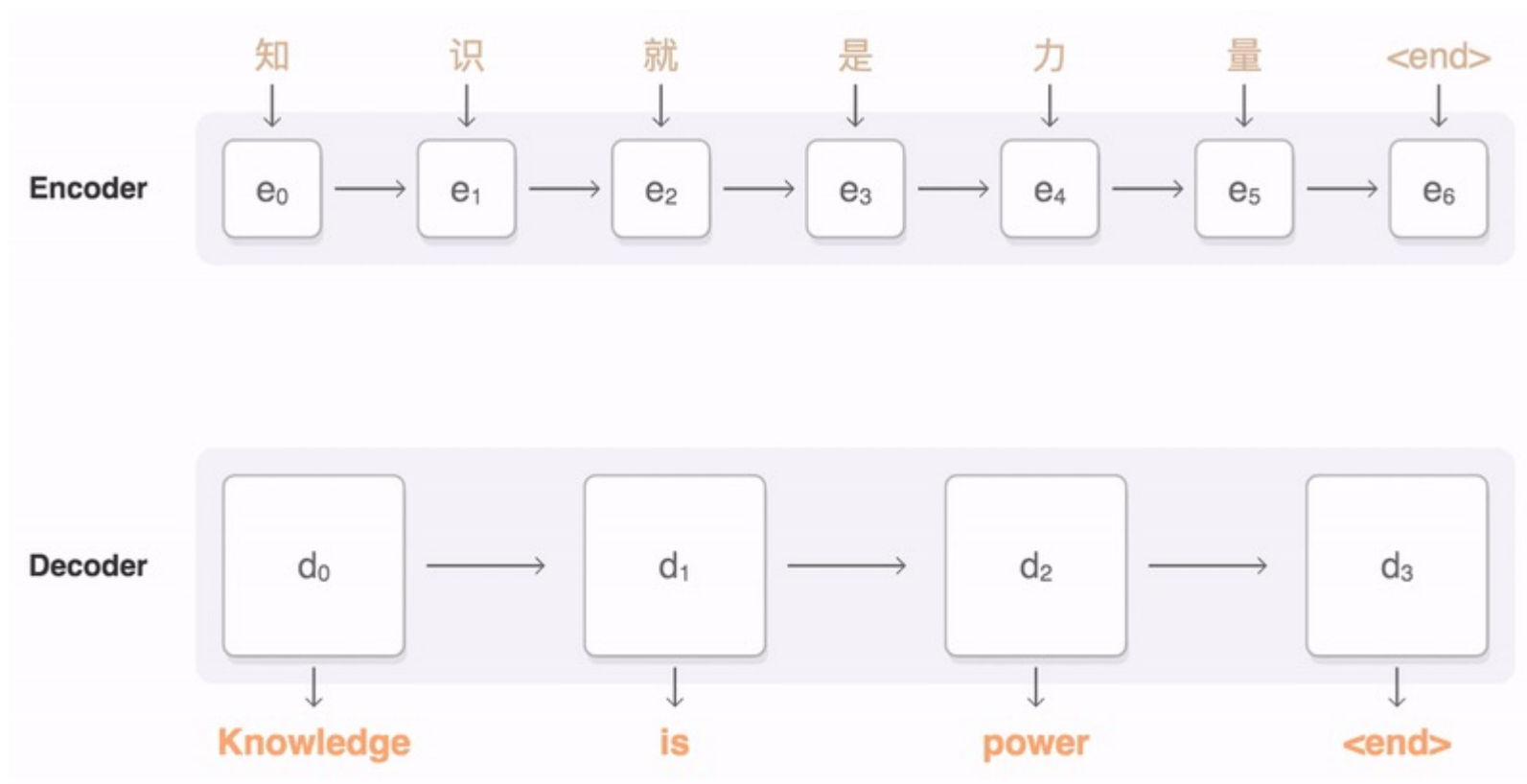


# Enhancement: Attention!

Our model so far



# Attention: Idea



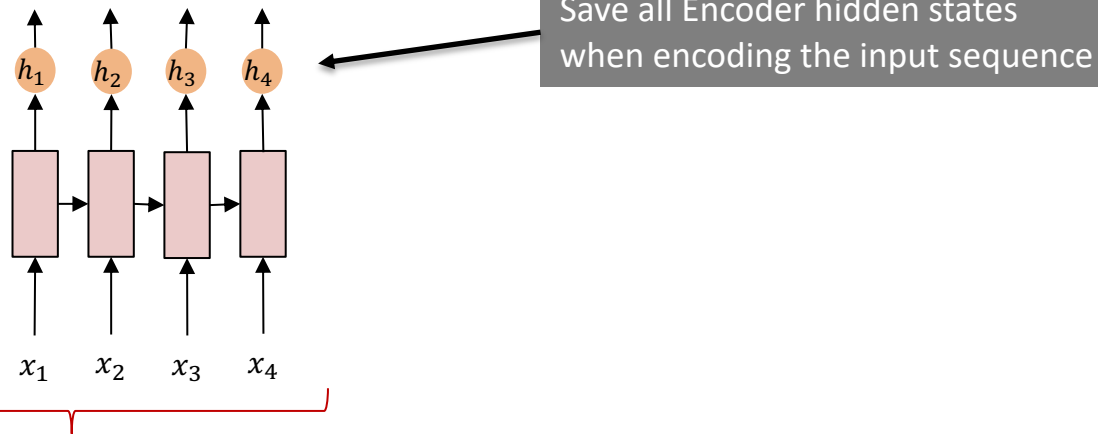
Animation from: <https://github.com/google/seq2seq>

## Attention: Idea

- Not just the last Encoder hidden state → save all Encoder hidden states
  - For every decoding step, find a weighting of the Encoder hidden states depending on the Decoder hidden state
  - Calculate a weighted sum of Encoder hidden states and use it in the Decoder
- 
- Encoder can capture per-step information; no need to squeeze everything into one hidden state
  - Decoder can pay attention to the important Encoder hidden states



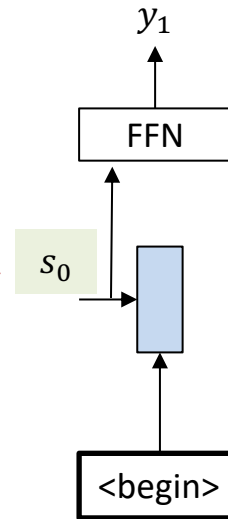
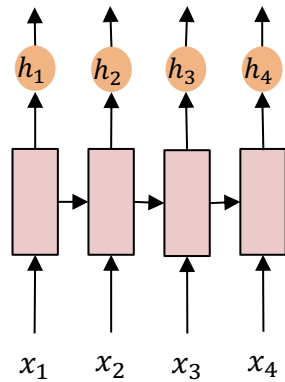
# Attention



Encoder

# Attention

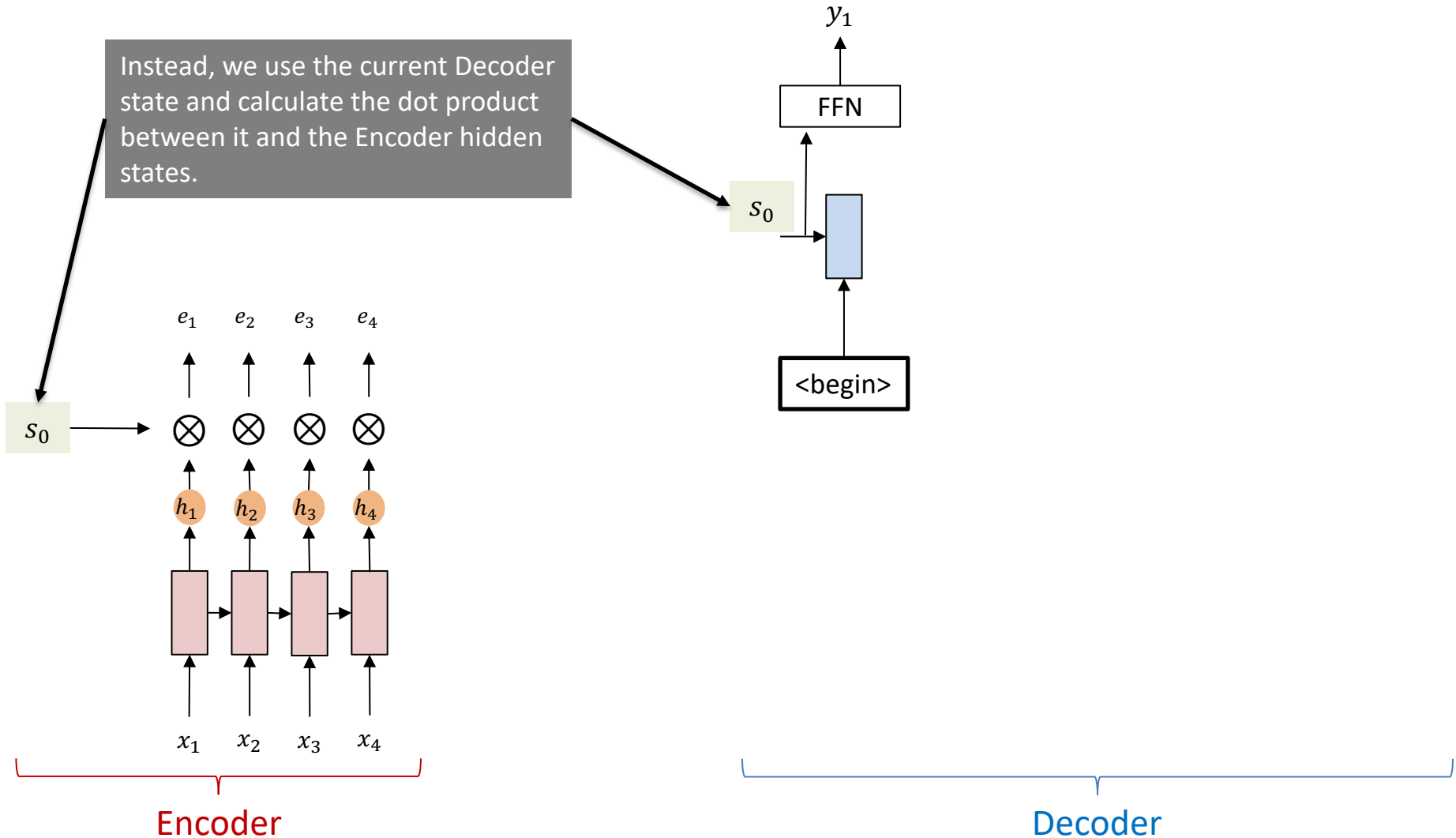
Normally,  $h_4$  would be used as  $s_0$ .  
We won't do this using attention.



Decoder

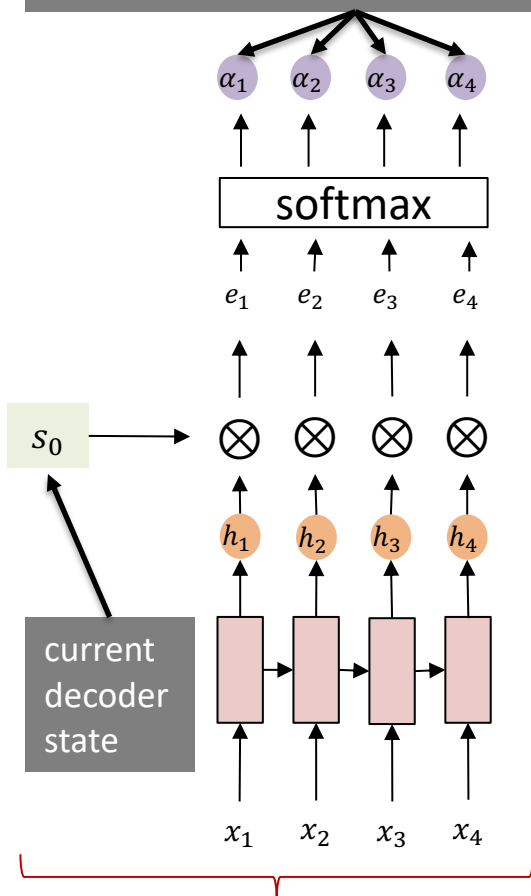
# Attention

Instead, we use the current Decoder state and calculate the dot product between it and the Encoder hidden states.

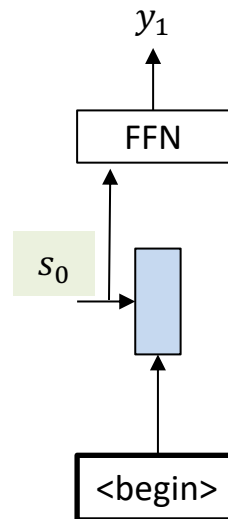


# Attention

Using a Softmax activation across all hidden states, we get **attention weights**



Encoder

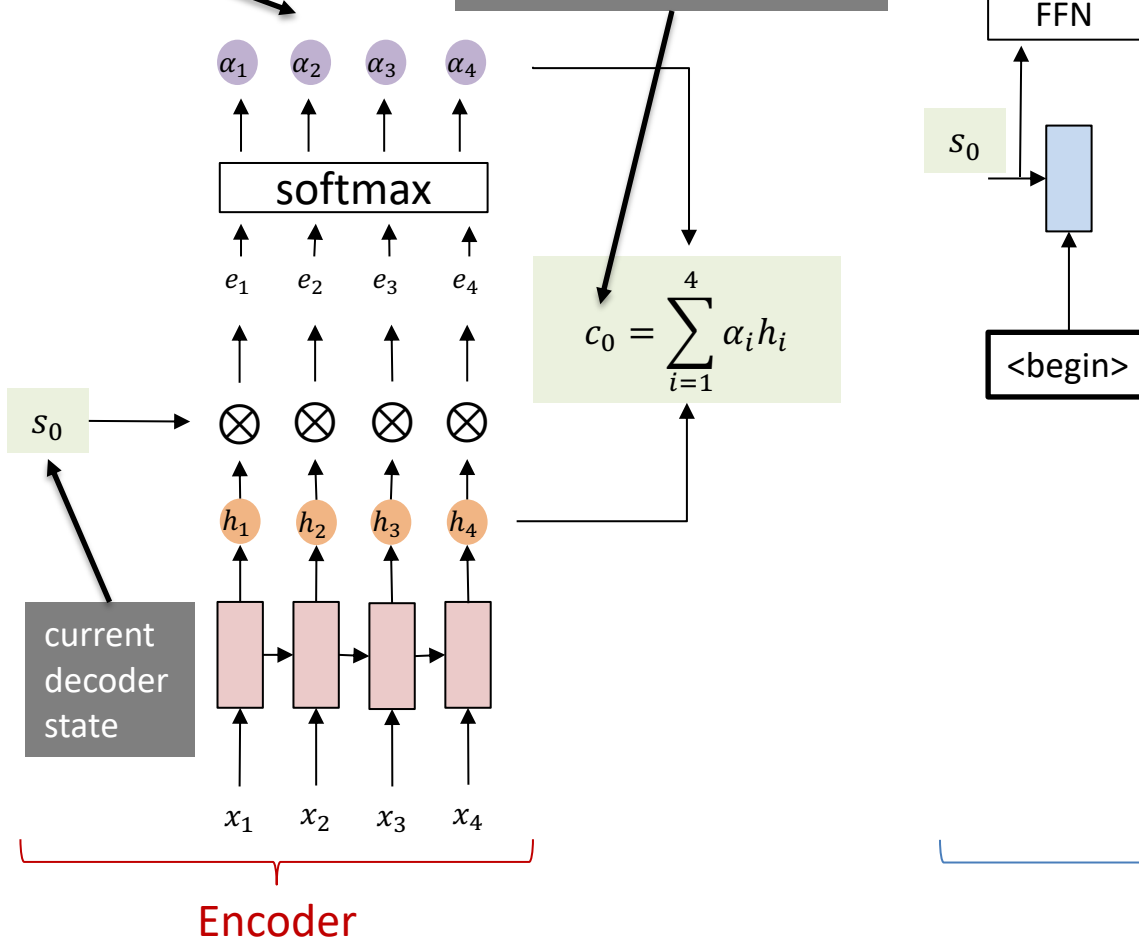


Decoder

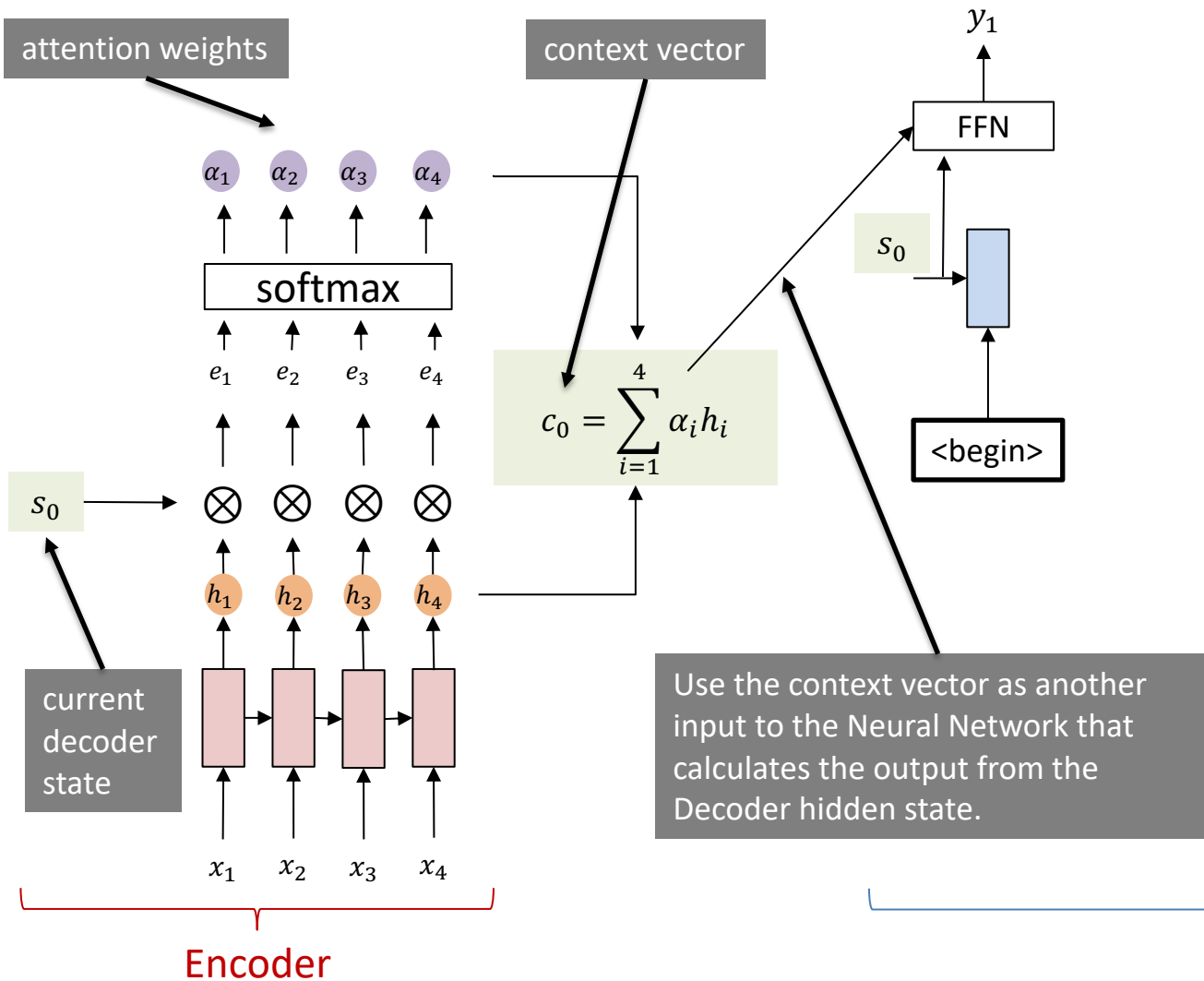
# Attention

attention weights

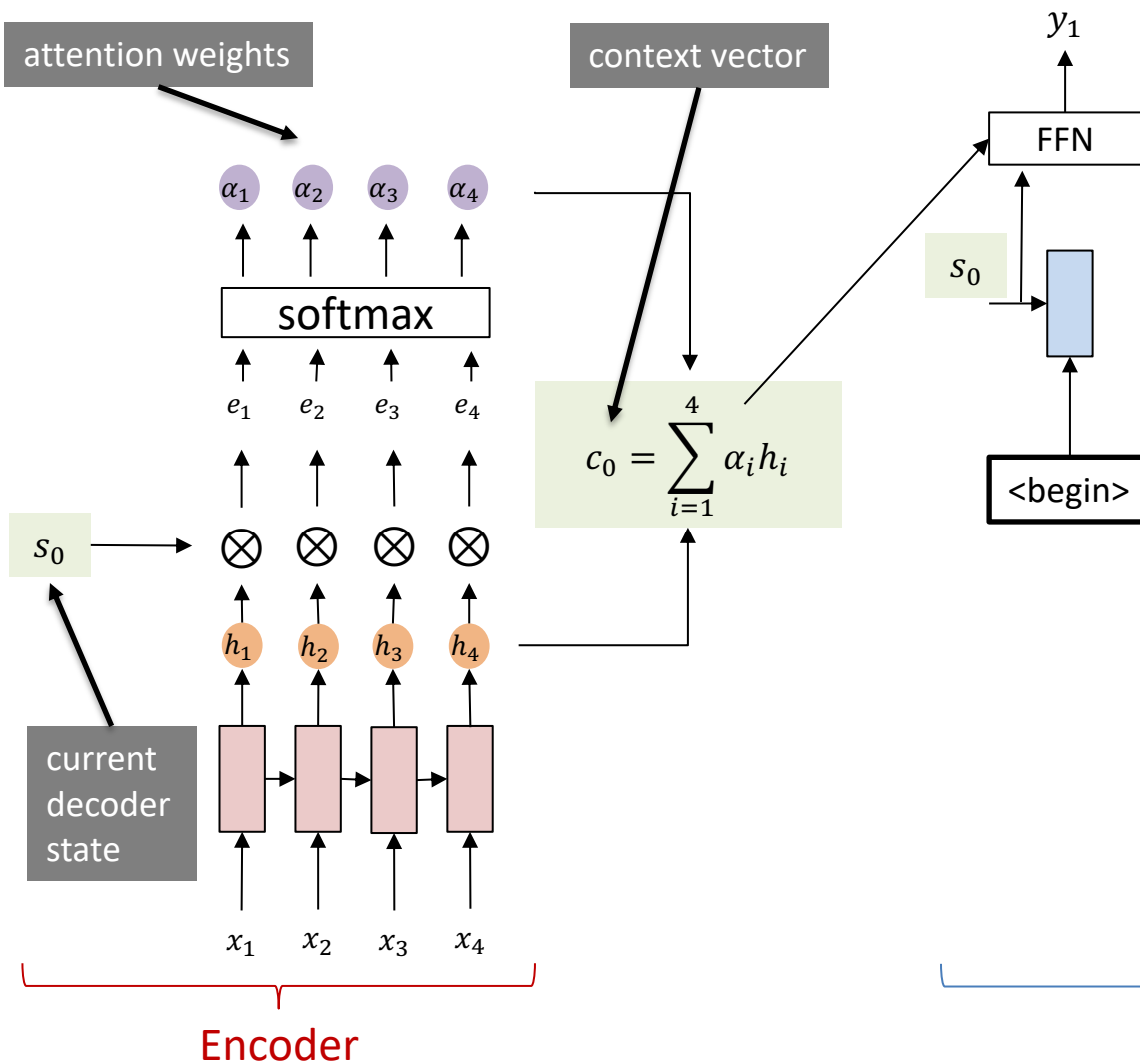
These weights are then used to calculate a weighted sum of the Encoder hidden states, called the **context vector** for the current step



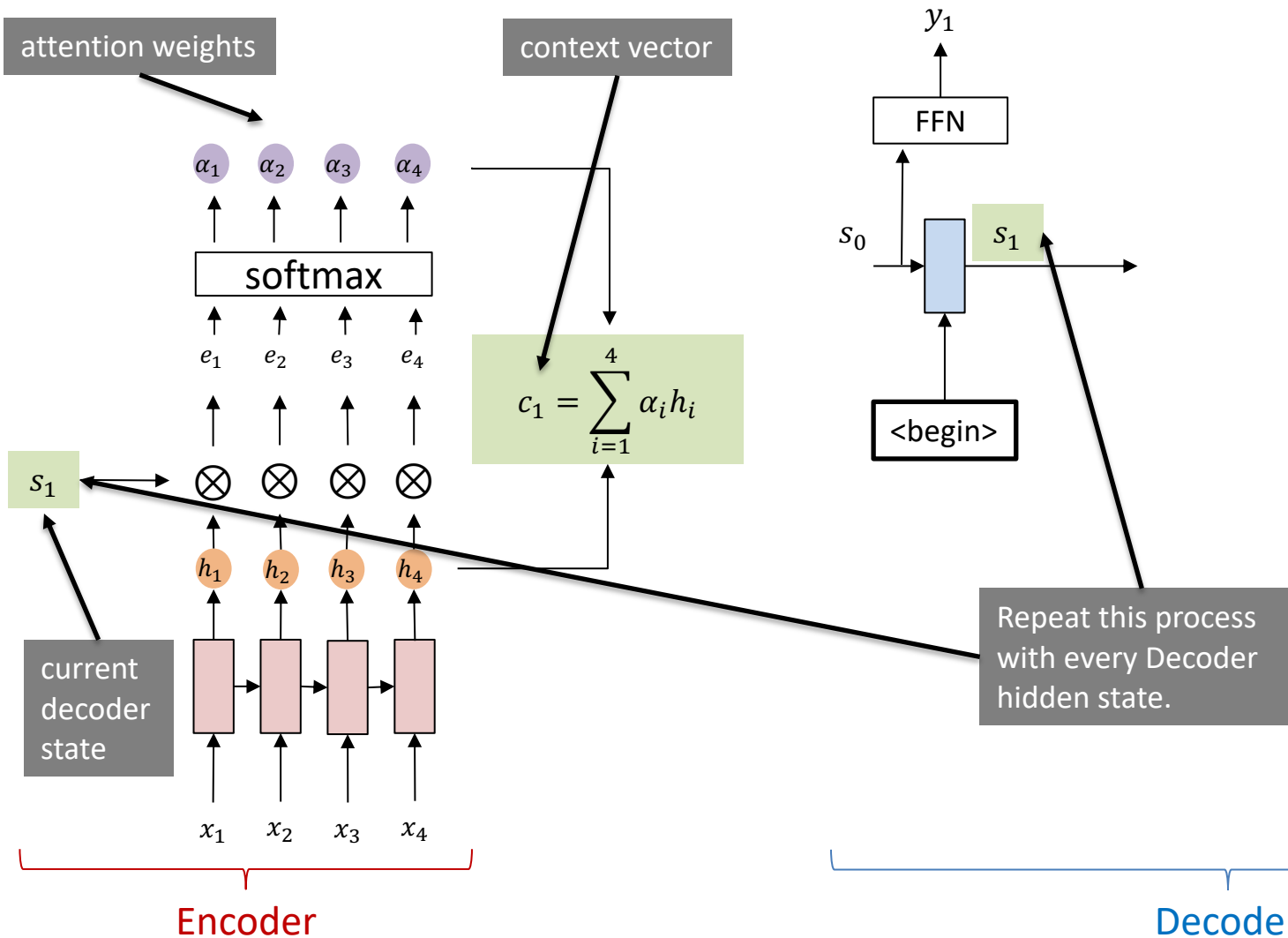
# Attention



# Attention

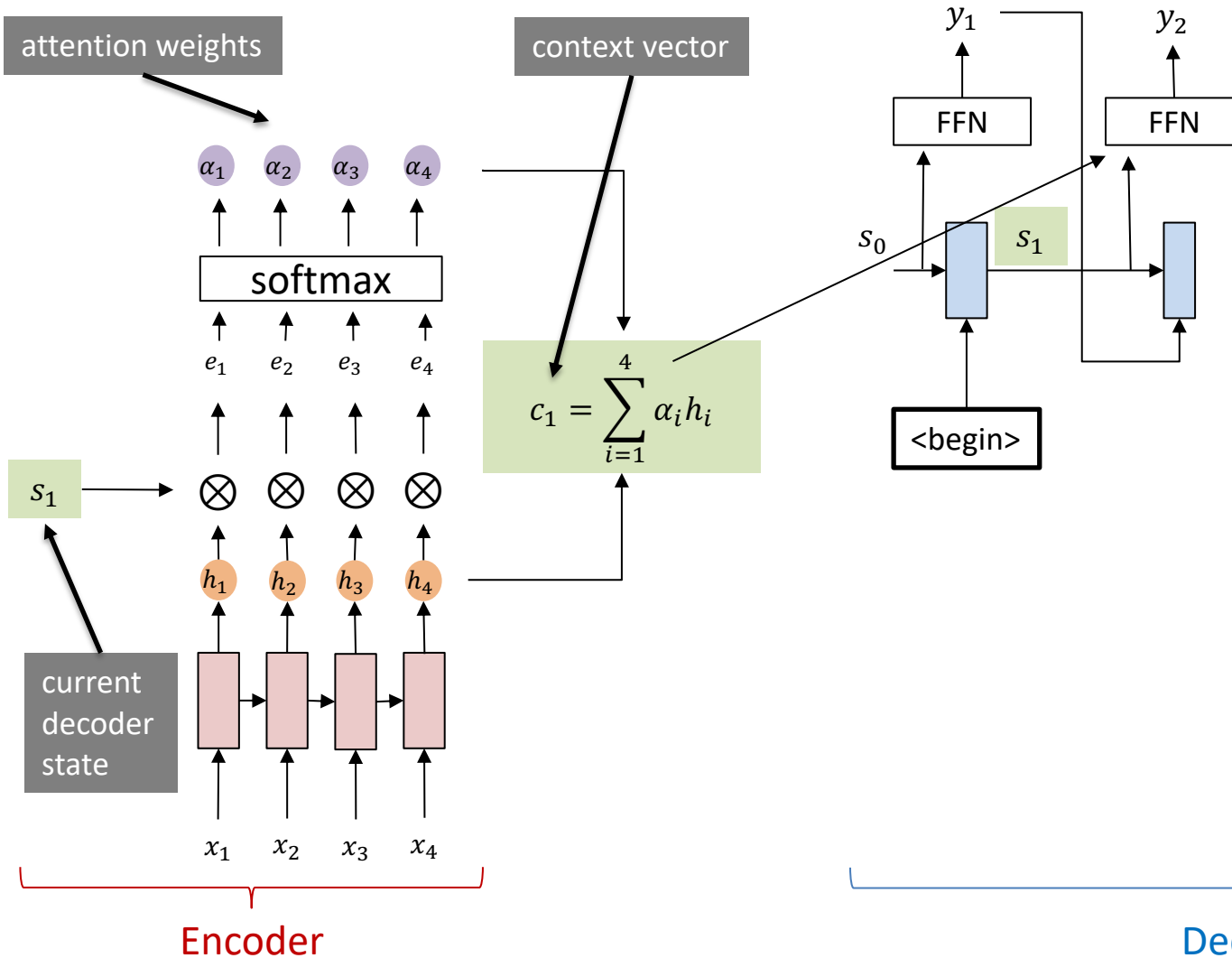


# Attention

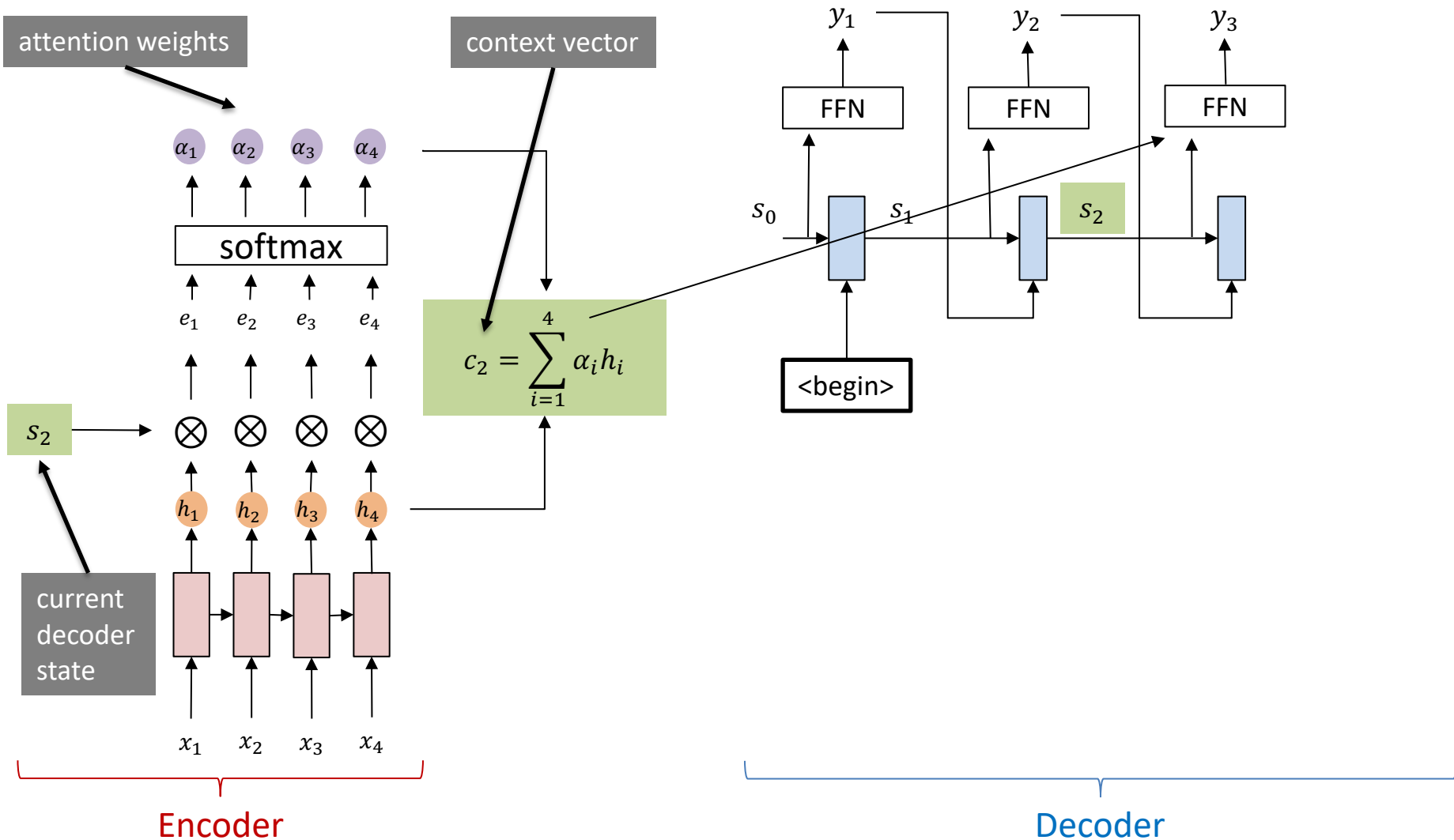




# Attention



# Attention



# Attention

- A context vector  $c_i$  is computed as a weighted sum over  $k$  encoder states at every decoding step  $i$ :

$$c_i = \sum_{j=1}^k \alpha_{ij} h_j$$

- Where  $\alpha_{ij}$  is a scalar weighting factor computed as:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{l=1}^k \exp(e_{il})}$$

- And  $e_{ij}$  is an alignment model:

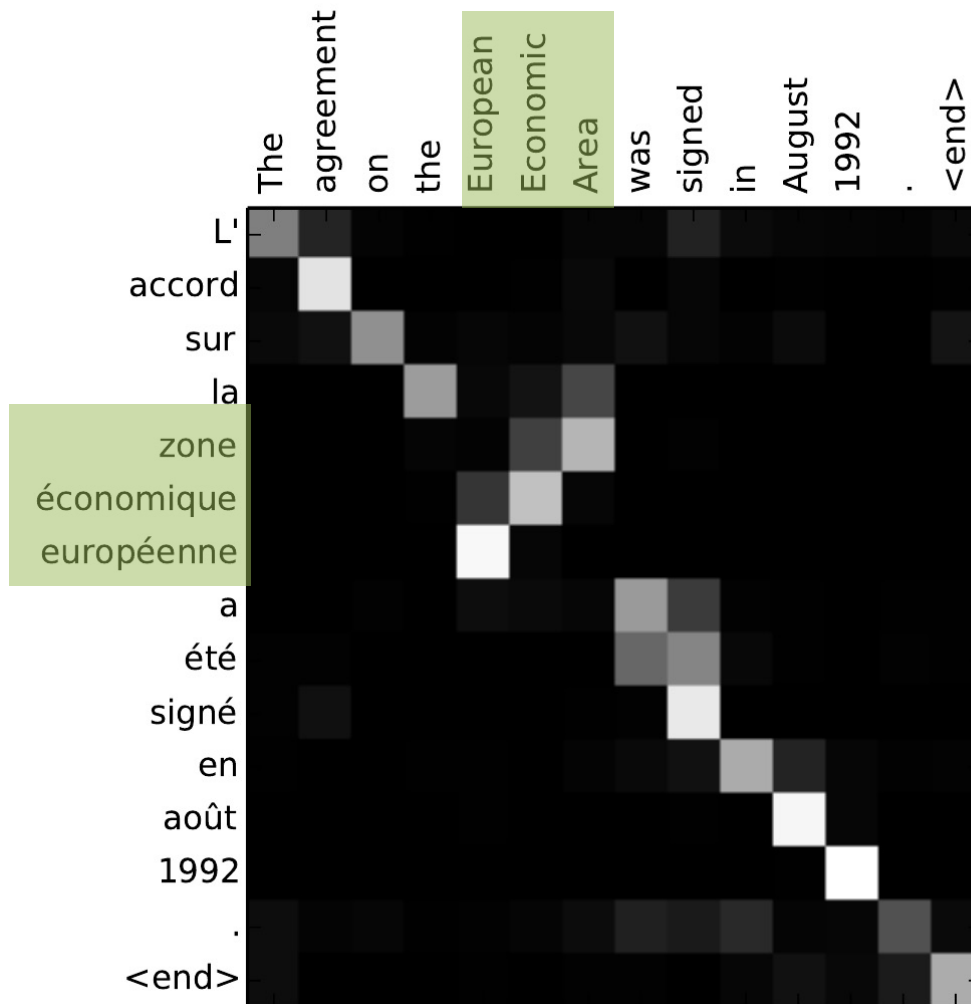
$$e_{ij} = a(s_{i-1}, h_j)$$

where  $a$  can be any transformation (e.g. a dot product)

# Attention

- Allows the model to distribute information across encoder states and extract only necessary bits at every decoder step.
- The attention weights can be visualized to allow insights into the models behaviour.

# Visualising Attention



Here some words are in a different order.

The plotted weights show how the attention shifts accordingly.

# Was that useful?

- We now know a lot about techniques for machine translation
  - We also know many „improvements“ over the vanilla model
  - But are they really useful?  
How well do our models perform?
- We need some way to measure translation quality!

# Evaluating Machine Translation

# Evaluating Machine Translation

- Needed: Some measure for the quality of machine translations
- Problem: **No „one true answer“**

During the meal, he read the newspaper



Während des Essens las er Zeitung

Während er aß, las er Zeitung

Er las Zeitung, während er aß

Er las während des Essens Zeitung

Beim Essen las er Zeitung



# Evaluating Machine Translation

- First attempt: **Unigram Precision**

- How many of the words in the machine translation appear in a target translation?

$$s = \frac{m}{w_t}$$

- $m$  = number of words appearing in target and machine translation
- $w_t$  = length of machine translation

During the meal, he read the newspaper

Target Translation:

Während des Essens las er Zeitung



Während des Essens las er Zeitung  
100%

Er las Zeitung, während e  
83%

Essen Essen Essen  
Essen Essen 100%

des Essens Zeitung 100%

las er Zeitung 80%

# Evaluating Machine Translation — BLEU

- Papineni et al, 2002: **Bilingual evaluation understudy (BLEU)**
  - Basically **the** evaluation method for machine translation (much to the authors' surprise)
  - Very high correlation with human judgement
  - Modifies Unigram Precision in two ways
    - Limit occurrences of the same word
    - Use n-grams instead of unigrams
  - Usually evaluates a translation against multiple target translations and averages the scores
  - For brevity, we use only one target translation

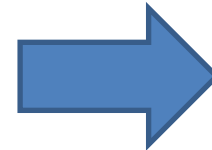
# Evaluating Machine Translation — BLEU

- Papineni et al, 2002: **Bilingual evaluation understudy (BLEU)**
  - Modification 1: Limit occurrences of the same word
  - For each word  $w$  in the machine translation, cap its count  $m_w$  to the count in the reference translation(s)

Source Sentence: During the meal, he read the newspaper

Gold Translation: Während des Essens las er Zeitung

Machine Translation: Essen Essen Essen Essen Essen



$$s = \frac{m_{essen}}{w_t} = \frac{1}{5} = 20\%$$

# Evaluating Machine Translation — BLEU

- Papineni et al, 2002: **Bilingual evaluation understudy (BLEU)**

- Modification 2: Replace unigrams by n-grams

- Example: Bigram Precision

Source Sentence: During the meal, he read the newspaper

Gold Translation: Während des Essens las er Zeitung

➡ „Während des“, „des Essens“, „Essens las“, „las er“, „er Zeitung“

Machine Translation: Zeitung er las während des Essens

➡ „Zeitung er“, „er las“, „las während“, „während des“, „des Essens“

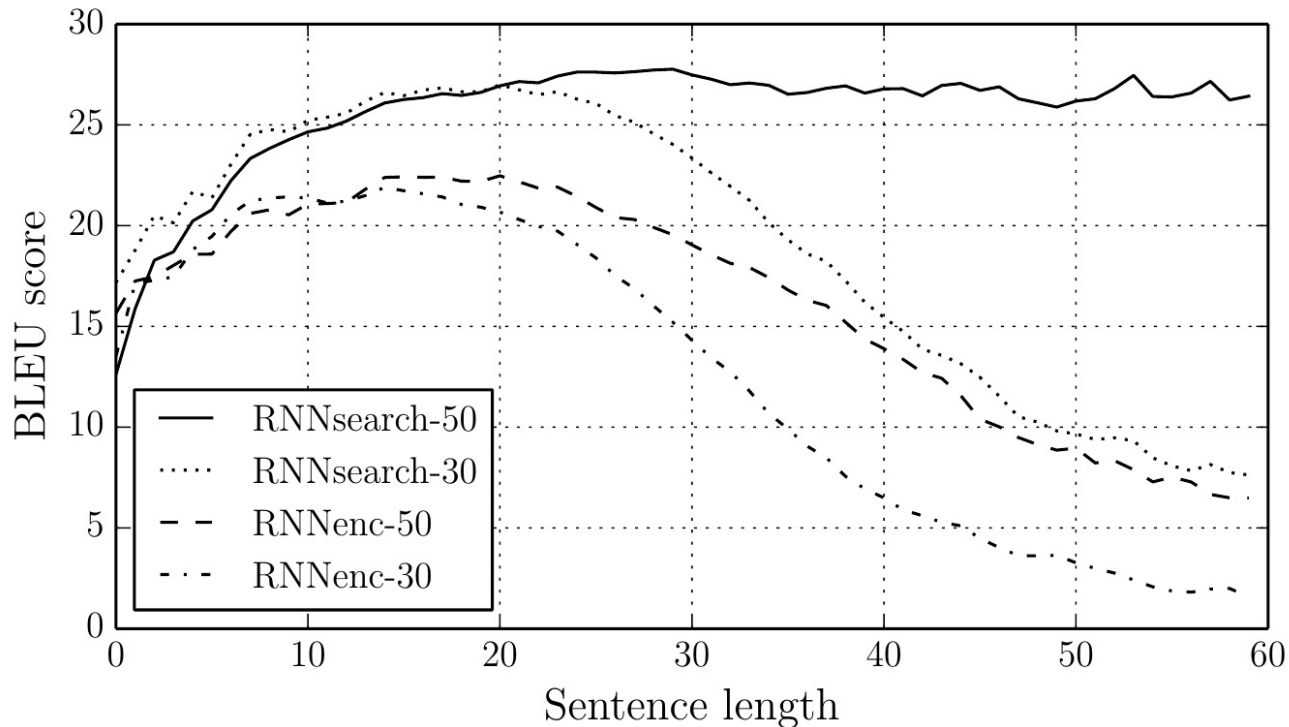
$b_1$        $b_2$        $b_3$        $b_4$        $b_5$

➡ 
$$s = \frac{m_{b_1} + m_{b_2} + m_{b_3} + m_{b_4} + m_{b_5}}{w_t} = \frac{0 + 0 + 0 + 1 + 1}{5} = 40\%$$

# Evaluating Machine Translation — BLEU

- Papineni et al, 2002: **Bilingual evaluation understudy (BLEU)**
  - In practice: Use Fourgram Precision
  - Use lots of target translations
  - **Do not evaluate single sentences!**  
BLEU only works well for large corpora
- Now how well does the Attention model perform?

# Attention: Results (Translation / Bahdanau et. al.)



All models are trained to translate sentences of length 30/50

- **RNNsearch-k** is an attention model trained on sentences of length k
- **RNNenc-k** is a normal seq2seq model trained on sentences of length k