

Introduction to Attention

Danna Gurari

University of Colorado Boulder

Fall 2022



<https://home.cs.colorado.edu/~DrG/Courses/NeuralNetworksAndDeepLearning/AboutCourse.html>

Review

- Last lecture:
 - Introduction to natural language processing
 - Text representation
 - Neural word embeddings
 - Programming tutorial
- Assignments (Canvas):
 - Lab assignment 3 due next week
- Questions?

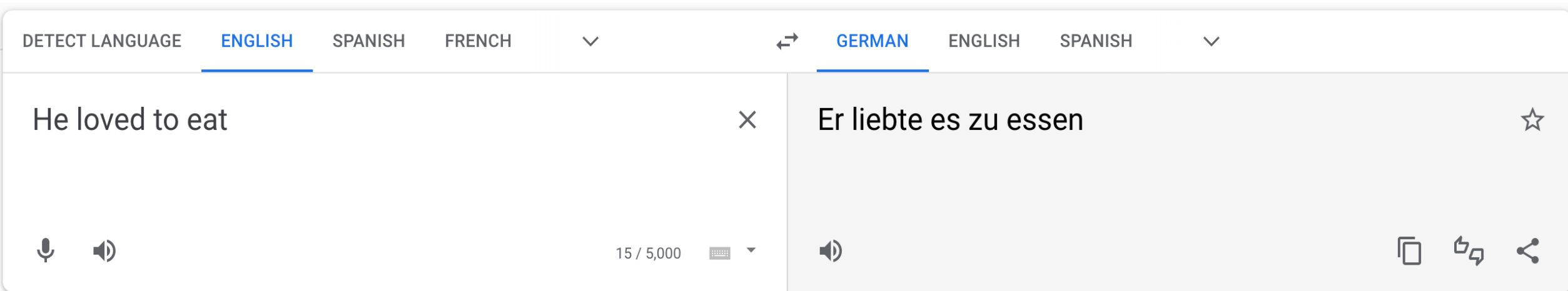
Today's Topics

- Motivation: machine neural translation for long sentences
- Encoder
- Decoder: attention
- Performance evaluation

Today's Topics

- Motivation: machine neural translation for long sentences
- Encoder
- Decoder: attention
- Performance evaluation

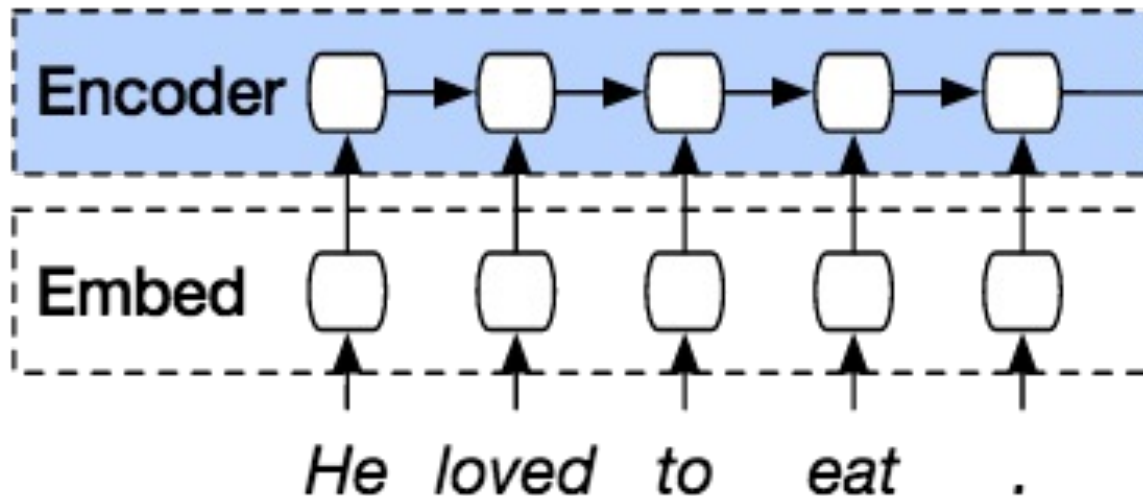
Task: Machine Translation



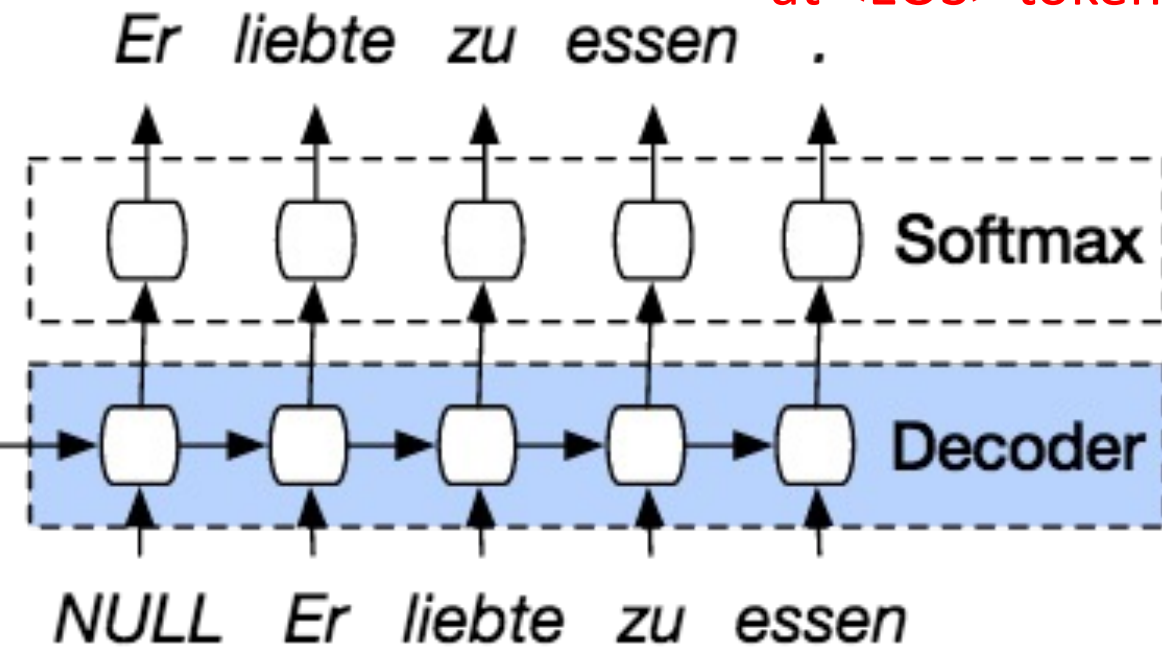
Which type of sequence problem is this: one-to-many, many-to-one, or many-to-many?

Pioneering Neural Network Approach

Input encoded into a
fixed-size vector



S

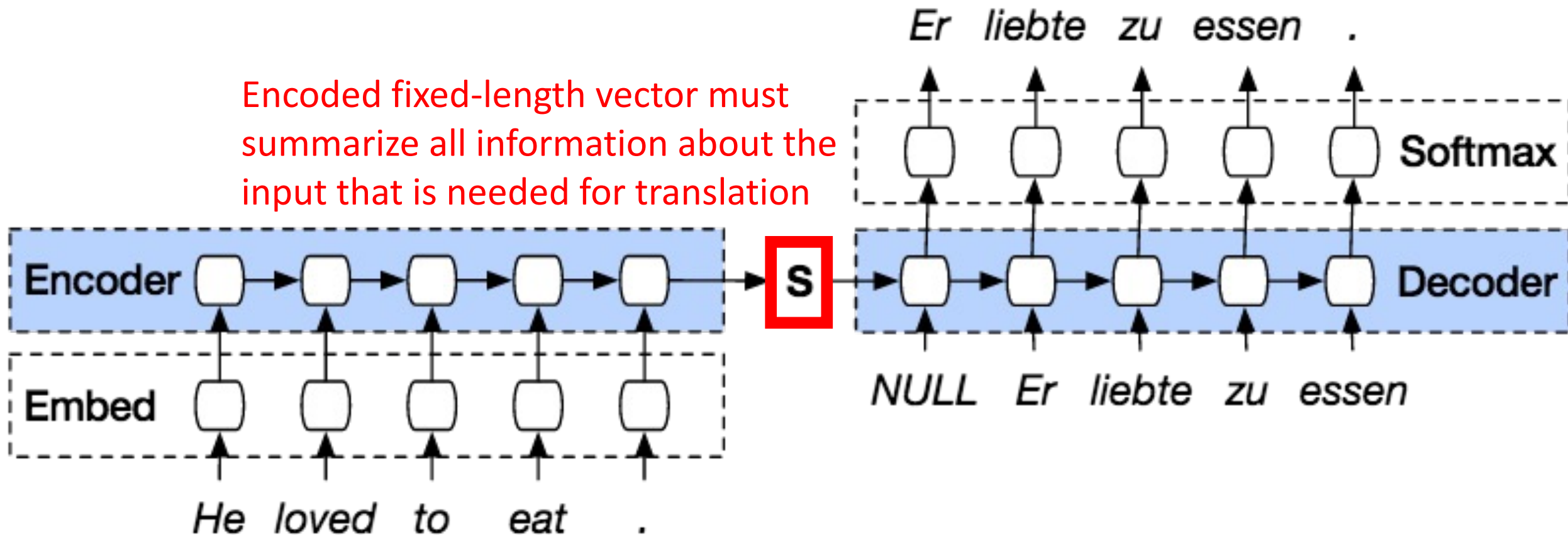


Predictions stop
at <EOS> token

Vector decoded
into a translation

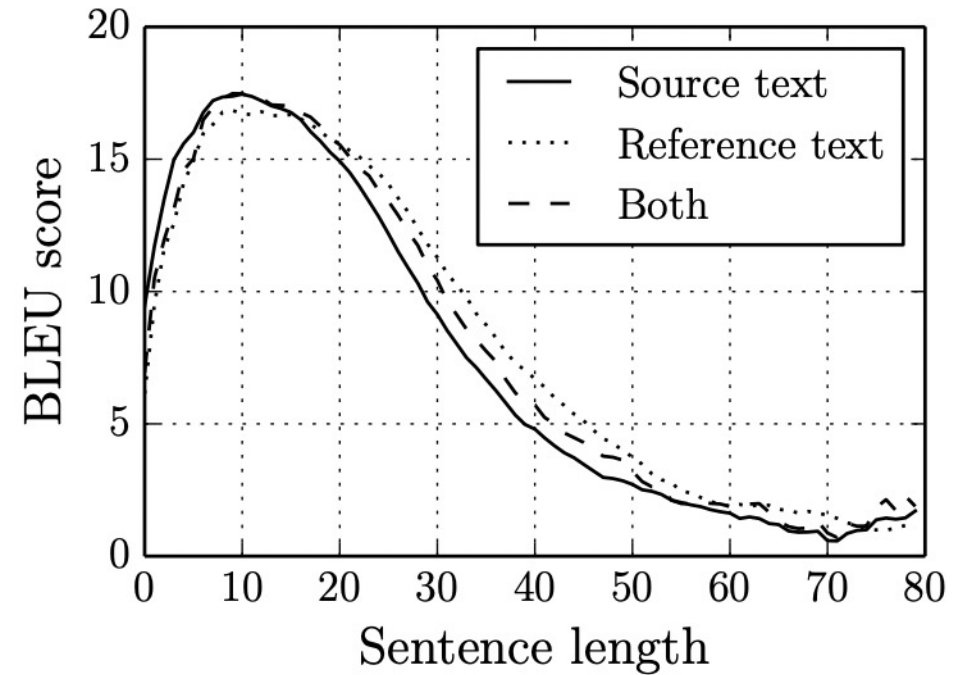
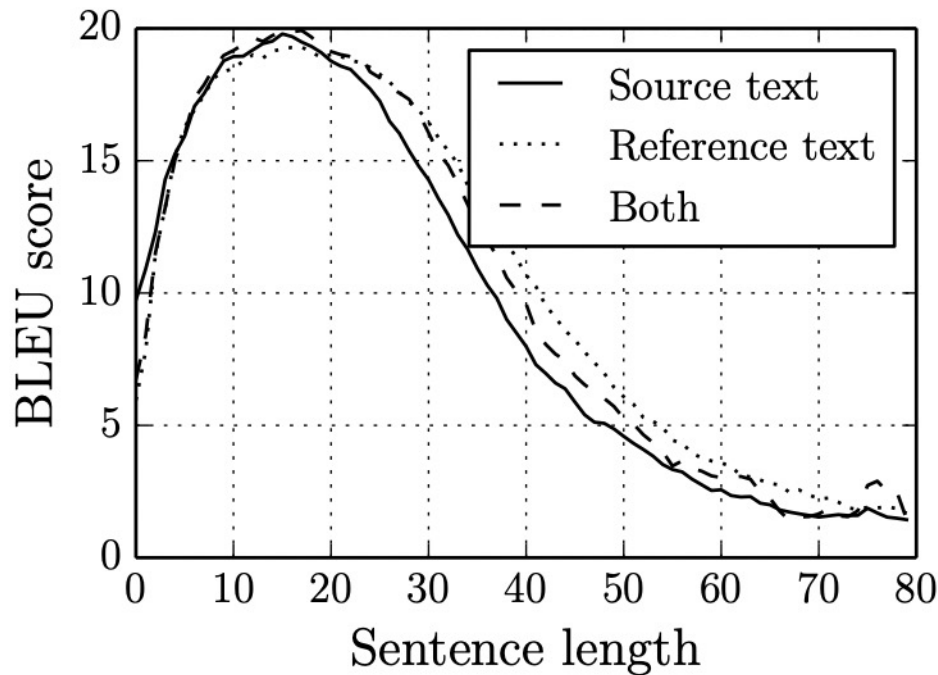
Pioneering Neural Network Approach

Encoded fixed-length vector must summarize all information about the input that is needed for translation



Analysis of Two Models

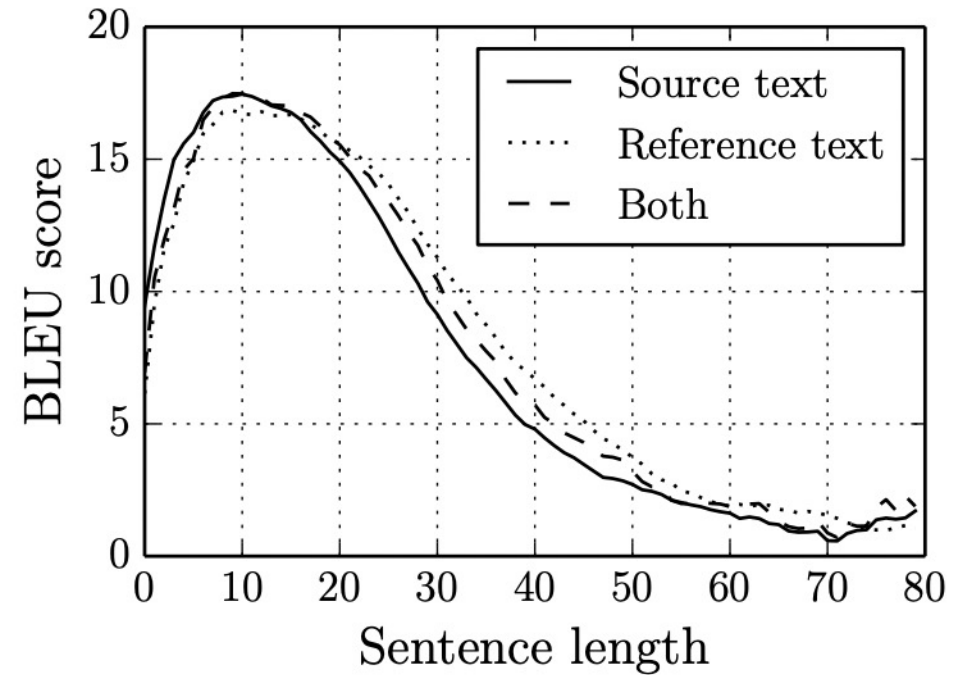
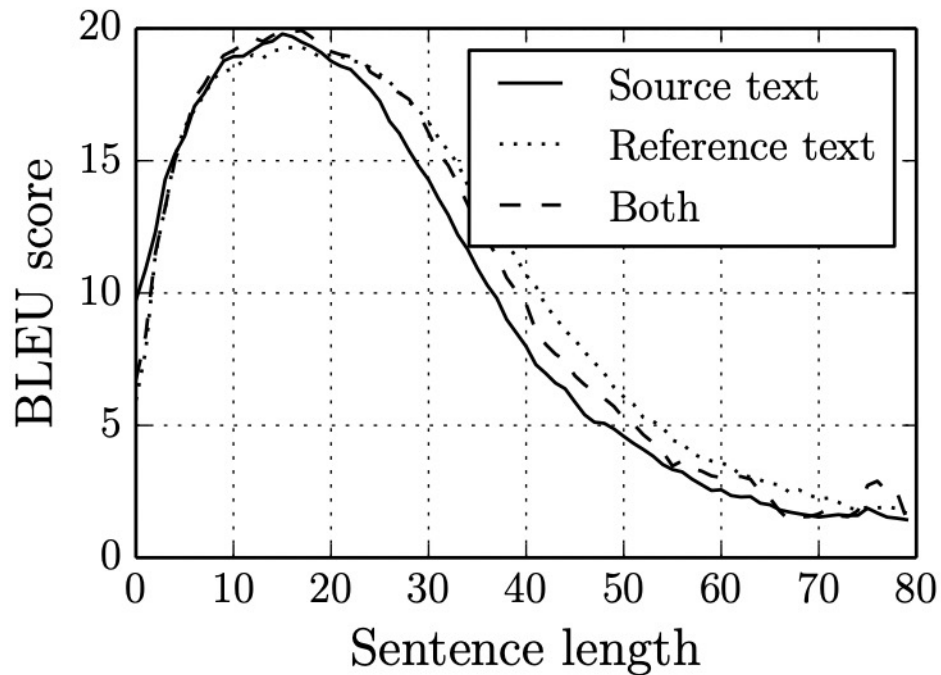
(larger scores are better)



What performance trend is observed for inputs (source) and outputs (reference) as the number of words in each sentence grows?

Analysis of Two Models

(larger scores are better)



Performance drops for longer sentences!

Problem: Performance Drops As Sentence Length Grows

Hypothesis: fixed-length vector lacks sufficient capacity to capture all relevant information for long sentences

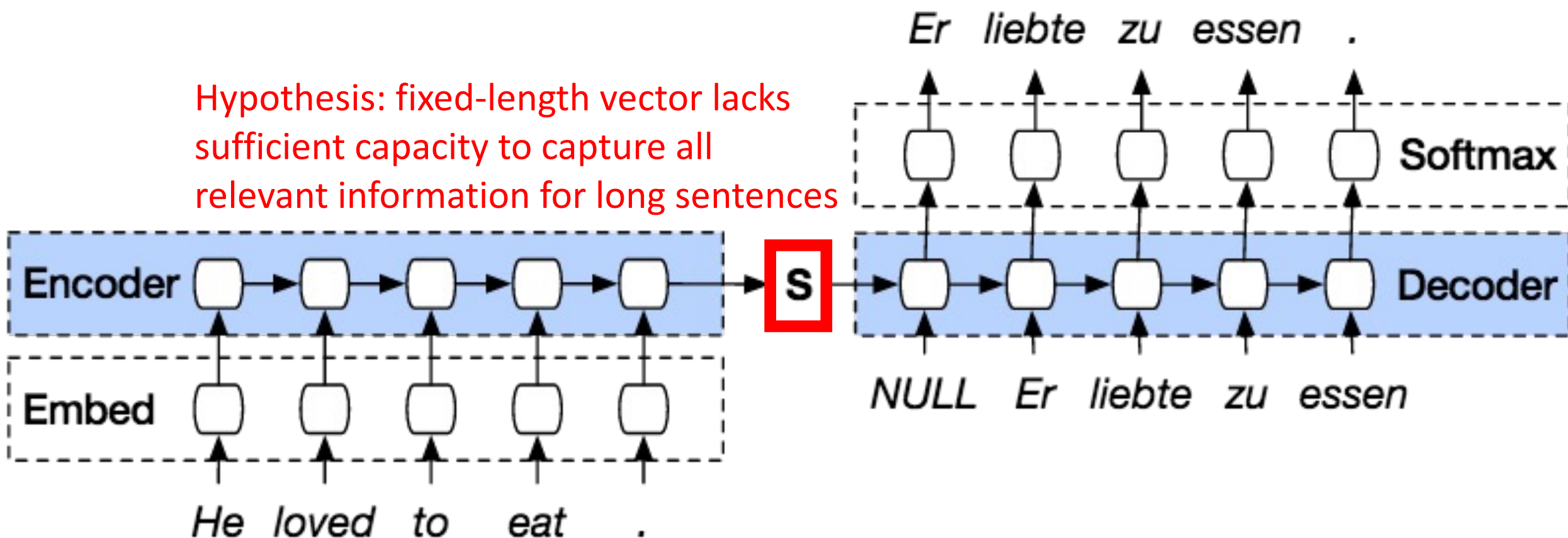
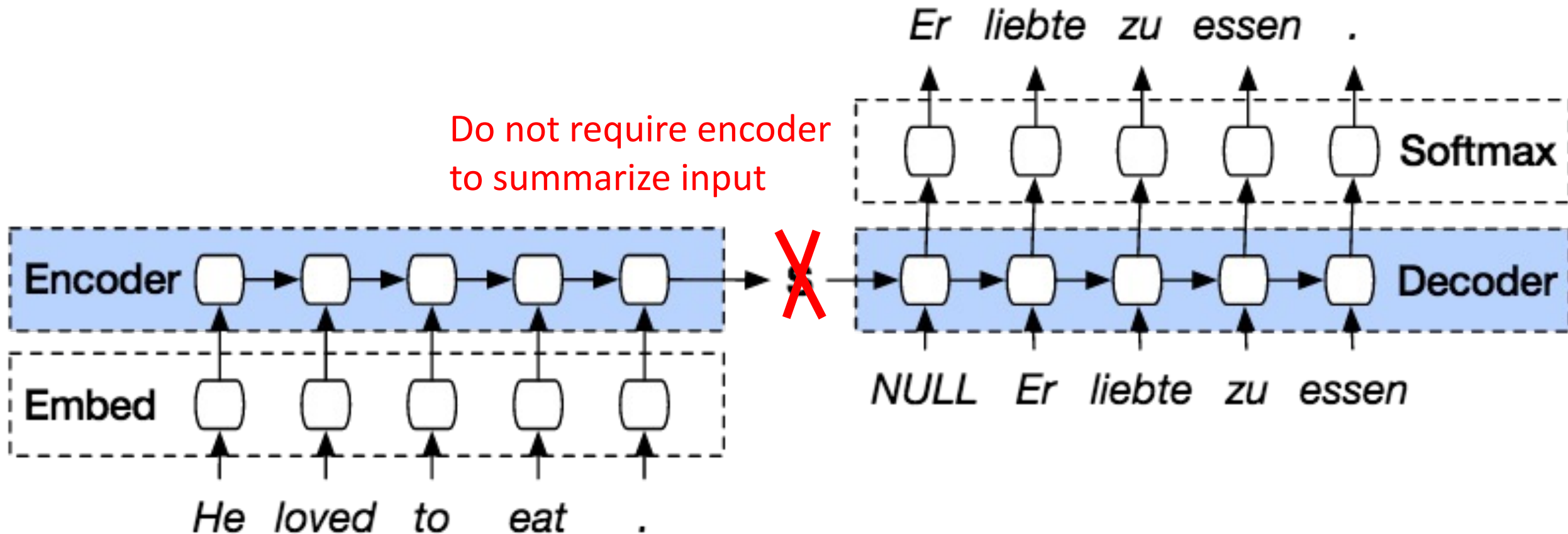


Image source: https://smerity.com/articles/2016/google_nmt_arch.html

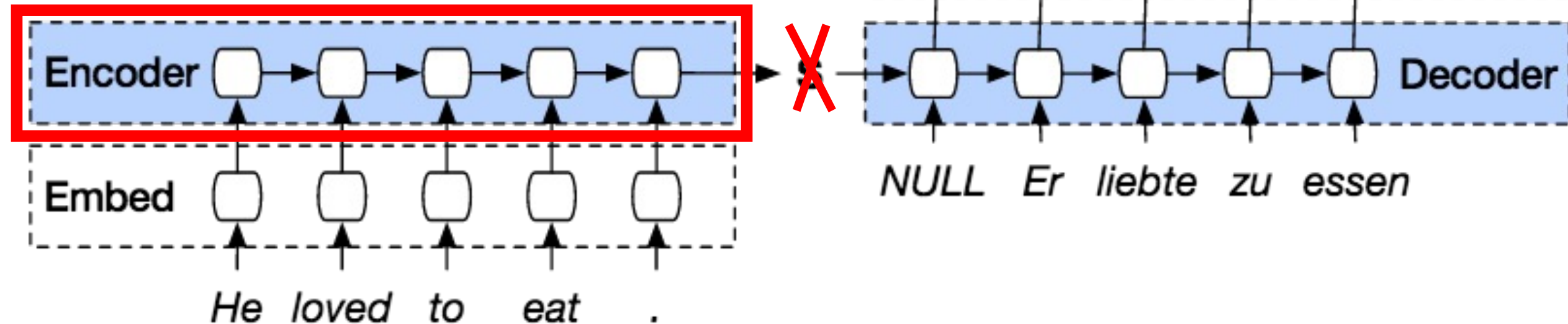
Cho et al. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. SSST 2014.

Idea to Preserve Performance for Long Sentences: **Attention**



Idea to Preserve Performance for Long Sentences: Attention

Instead, have the encoder pass **all** input's hidden states to the decoder to decide which to use for prediction at each time step



Idea to Preserve Performance for Long Sentences: Attention

Decoder decides which inputs are needed for prediction at each time step; e.g., “hard attention” focuses on one input



Note: while word order between the input and target align in this example, it can differ

Idea to Preserve Performance for Long Sentences: **Attention**

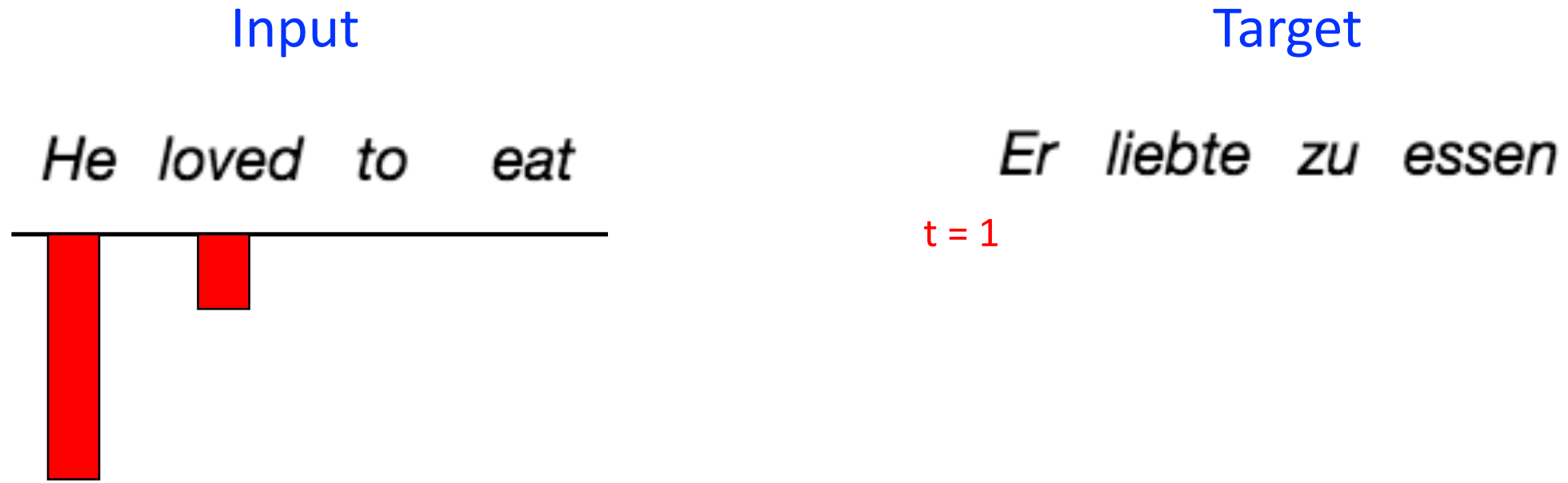
Decoder decides which inputs are needed for prediction at each time step; e.g., “hard attention” focuses on one input



Limitations: a target word relies on information about one input word and “hard attention” is not differentiable

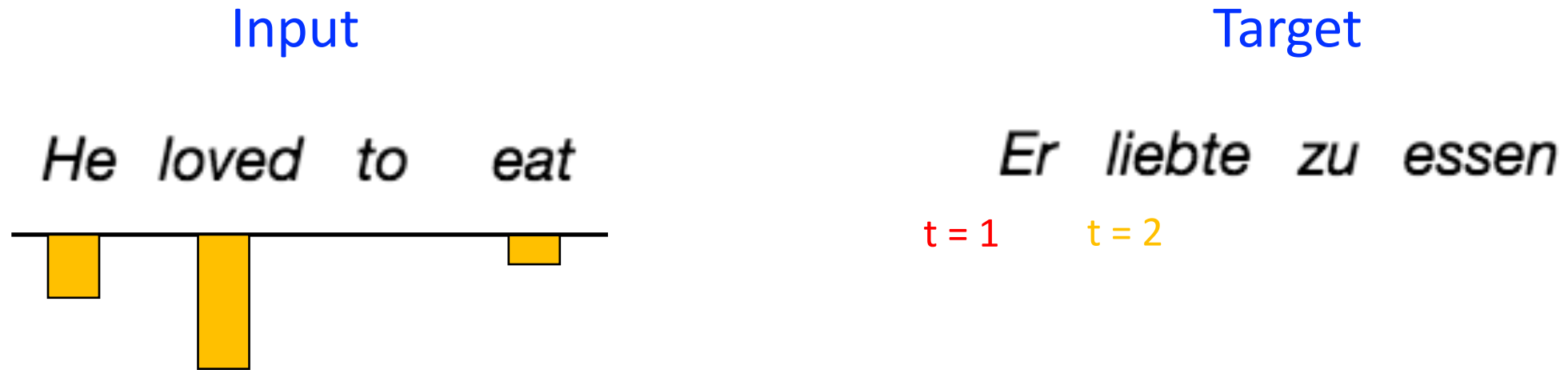
Idea to Preserve Performance for Long Sentences: **Attention**

Decoder decides which inputs are needed for prediction at each time step; e.g., “soft attention” uses a weighted combination of the input



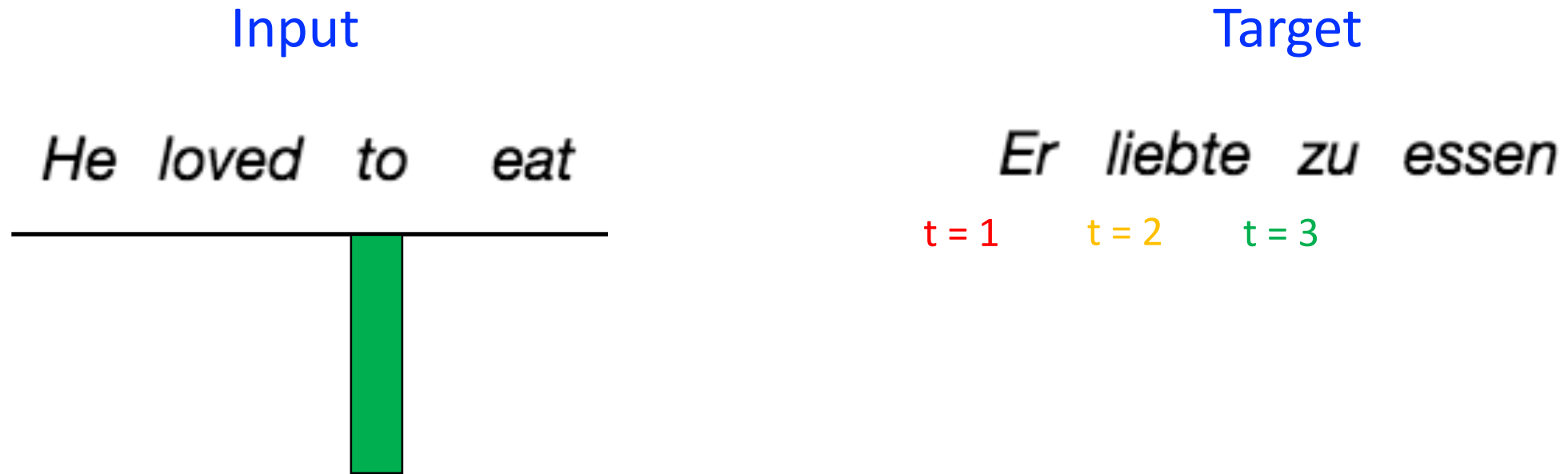
Idea to Preserve Performance for Long Sentences: **Attention**

Decoder decides which inputs are needed for prediction at each time step; e.g., “soft attention” uses a weighted combination of the input



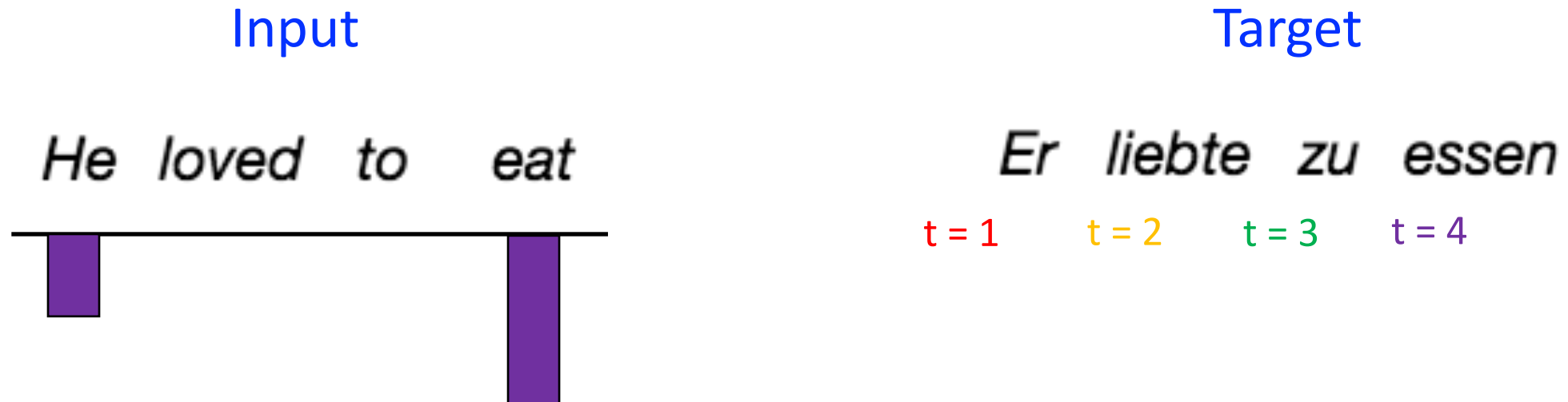
Idea to Preserve Performance for Long Sentences: **Attention**

Decoder decides which inputs are needed for prediction at each time step; e.g., “soft attention” uses a weighted combination of the input



Idea to Preserve Performance for Long Sentences: **Attention**

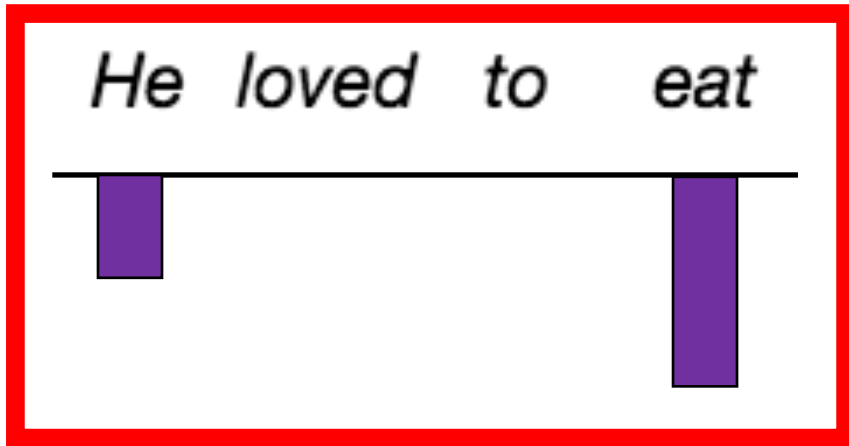
Decoder decides which inputs are needed for prediction at each time step; e.g., “soft attention” uses a weighted combination of the input



“Soft” Attention: Challenge

Decoder decides which inputs are needed for prediction at each time step; e.g., “soft attention” uses a weighted combination of the input

Input



Target

Er liebte zu essen

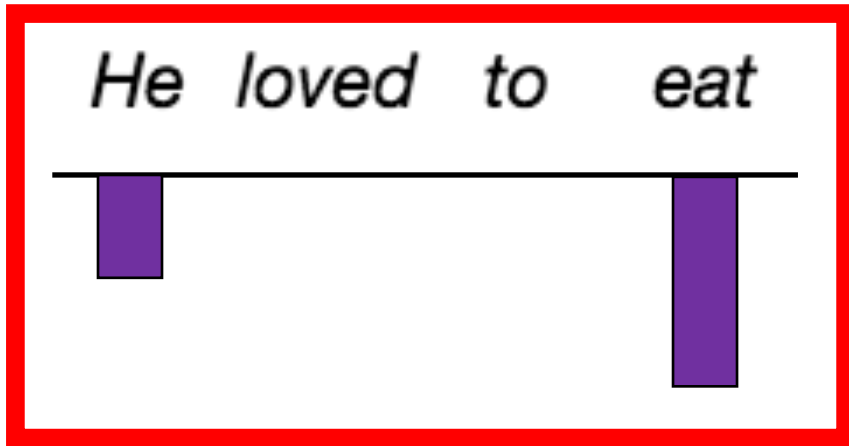
$t = 1$ $t = 2$ $t = 3$ $t = 4$

How should weights be chosen for each input?

“Soft” Attention: Challenge

Decoder decides which inputs are needed for prediction at each time step;
e.g., “soft attention” uses a weighted combination of the input

Input



Target

Er liebte zu essen

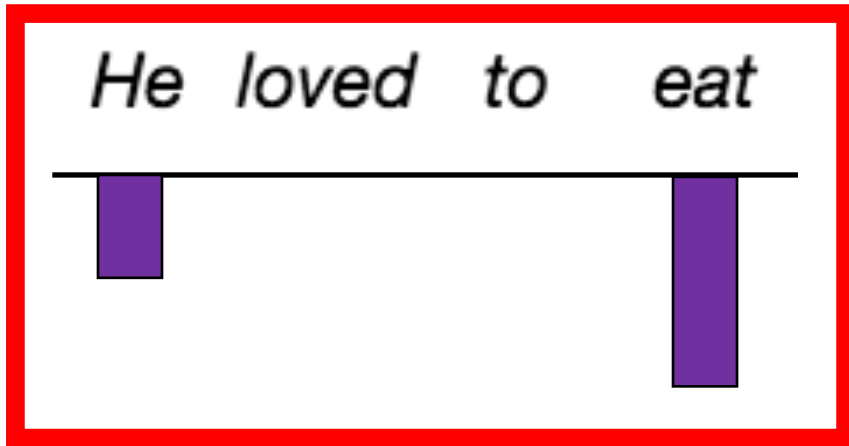
t = 1 t = 2 t = 3 t = 4

Could collect manual annotations and then incorporate into the loss function that predicted weights should match ground truth weights... but this approach is impractical

“Soft” Attention: Challenge

Decoder decides which inputs are needed for prediction at each time step;
e.g., “soft attention” uses a weighted combination of the input

Input



Target

Er liebte zu essen

t = 1 t = 2 t = 3 t = 4

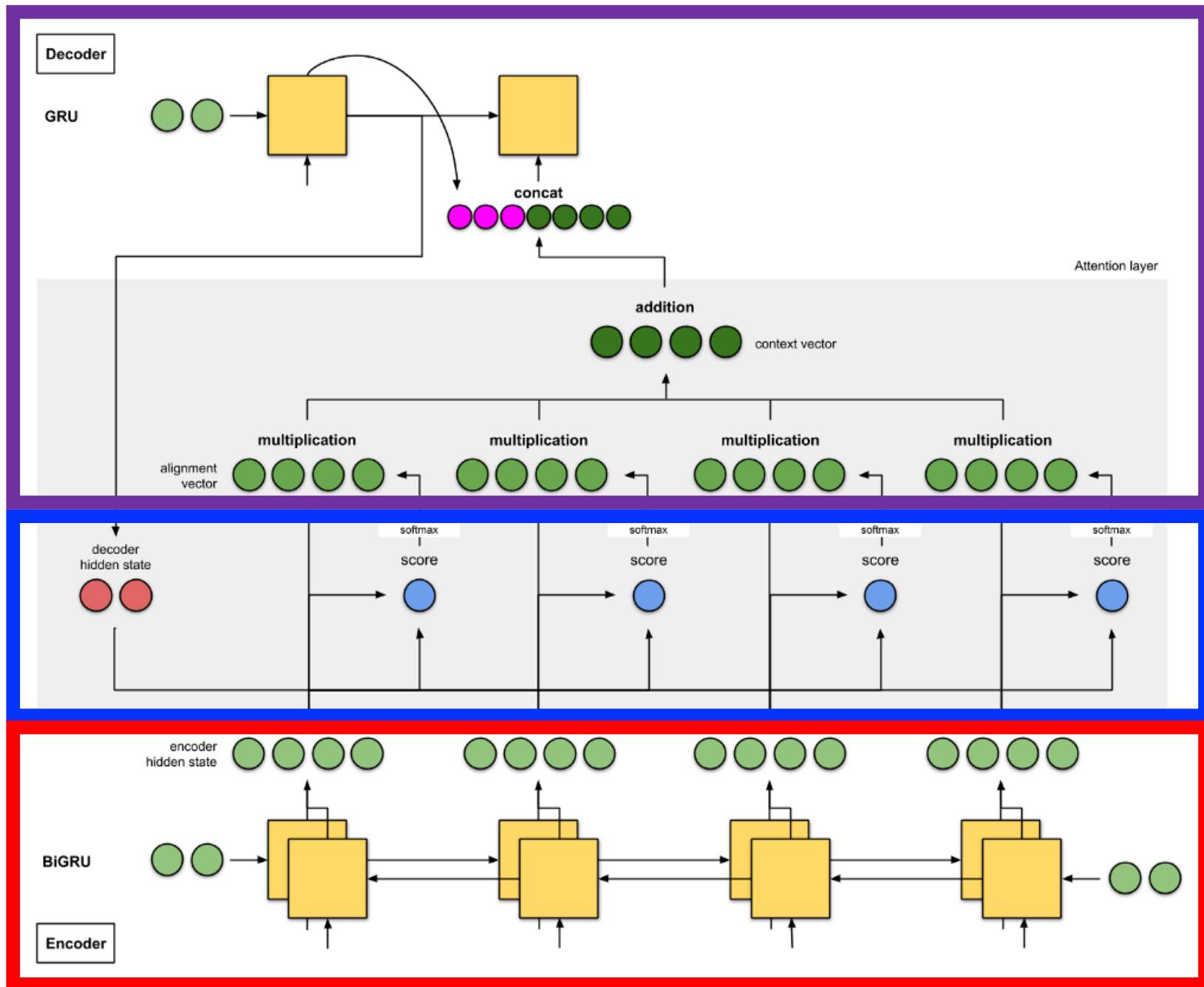
Instead, have the model learn
how to weight each input!

Solution

3. At each decoder time step, a prediction is made based on the weighted sum of the inputs

2. At each decoder time step, attention weights are computed that determine each input's relevance for the prediction

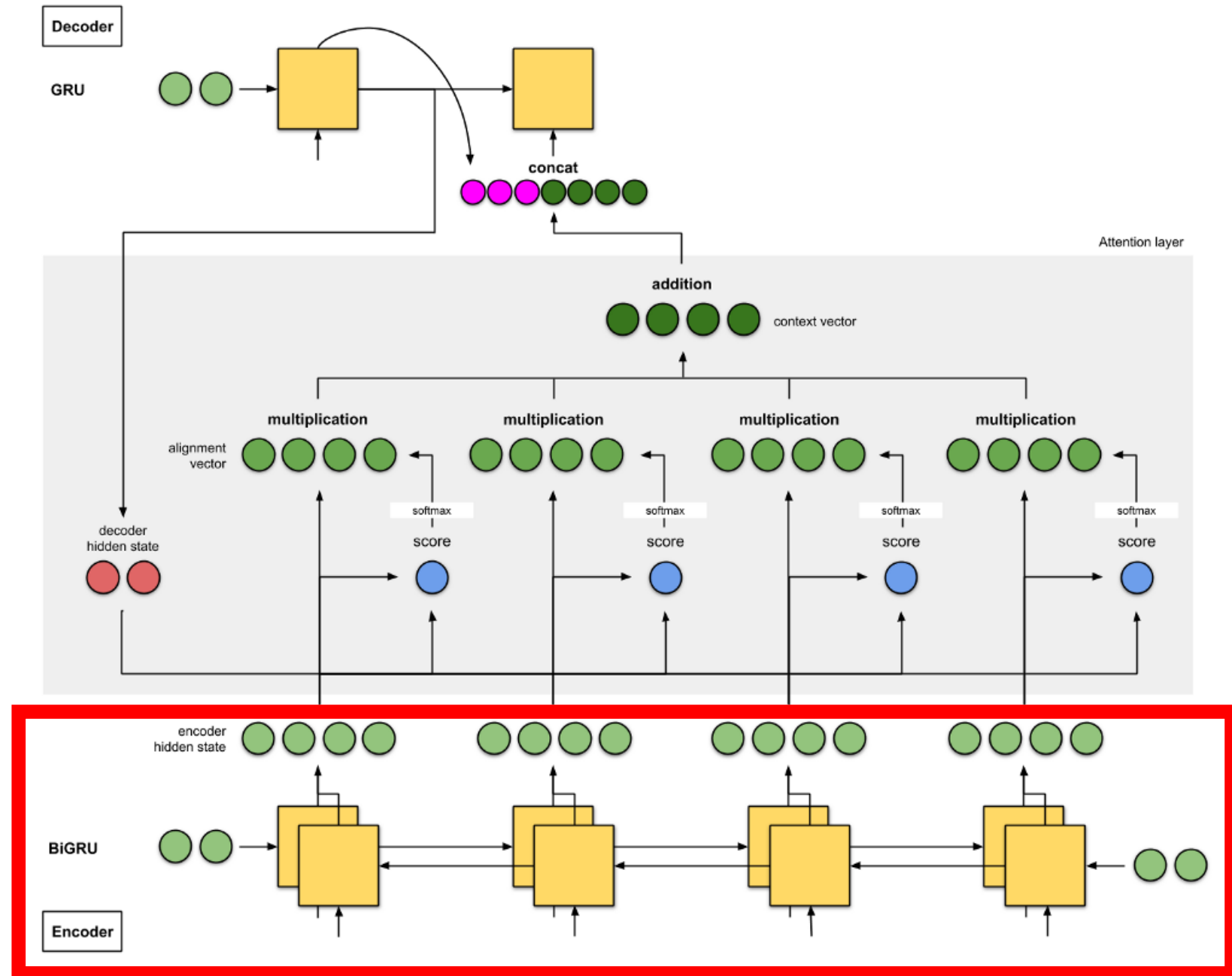
1. Encoder produces hidden state for every input



Today's Topics

- Motivation: machine neural translation for long sentences
- Encoder
- Decoder: attention
- Performance evaluation

Solution

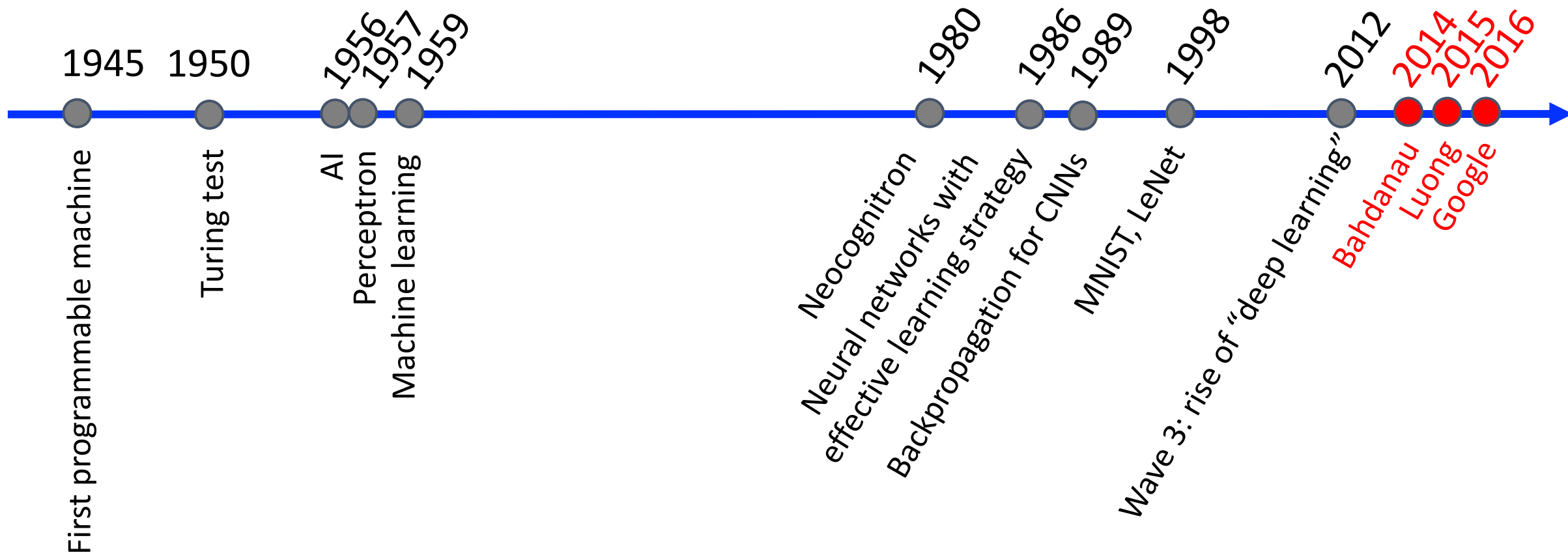


1. Encoder produces hidden state for every input

Popular Choices for Encoding Input

- Bi-directional RNN (Bahdanau)
- Stacked RNNs (Luong)
- Bi-directional and Stacked RNN (Google)

Historical Context

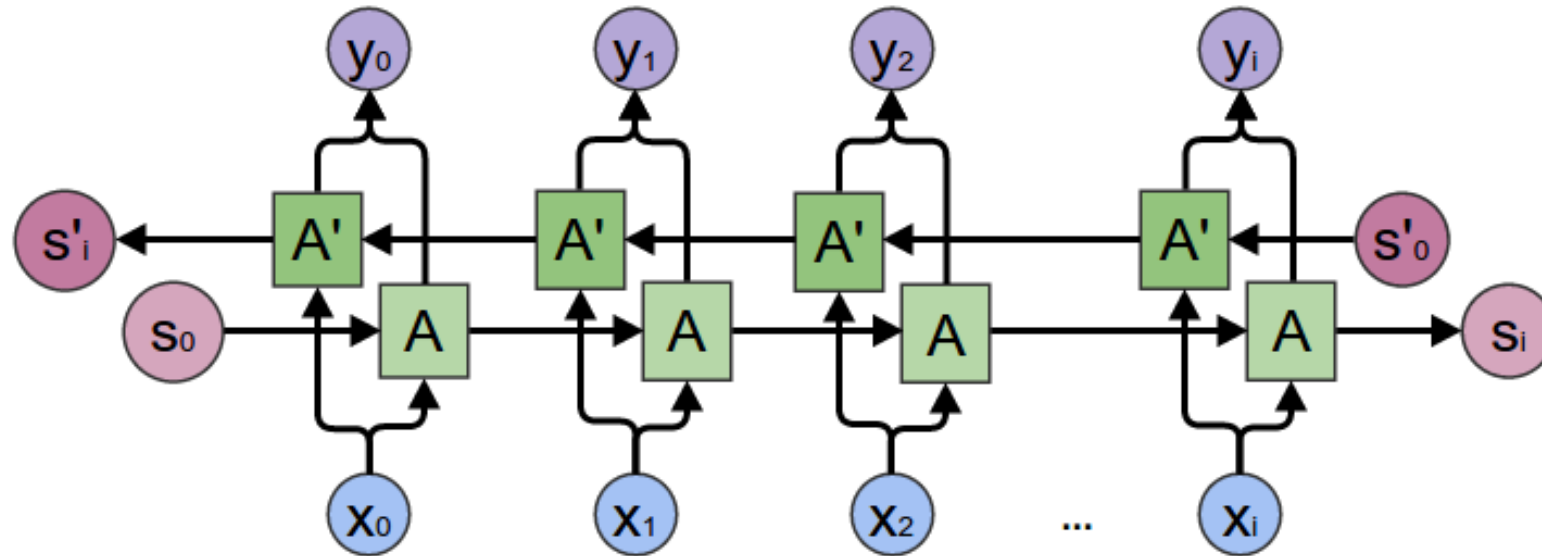


Popular Choices for Encoding Input

- Bi-directional RNN (Bahdanau)
- Stacked RNNs (Luong)
- Bi-directional and Stacked RNN (Google)

Bahdanau's Neural Machine Translation: Encoder

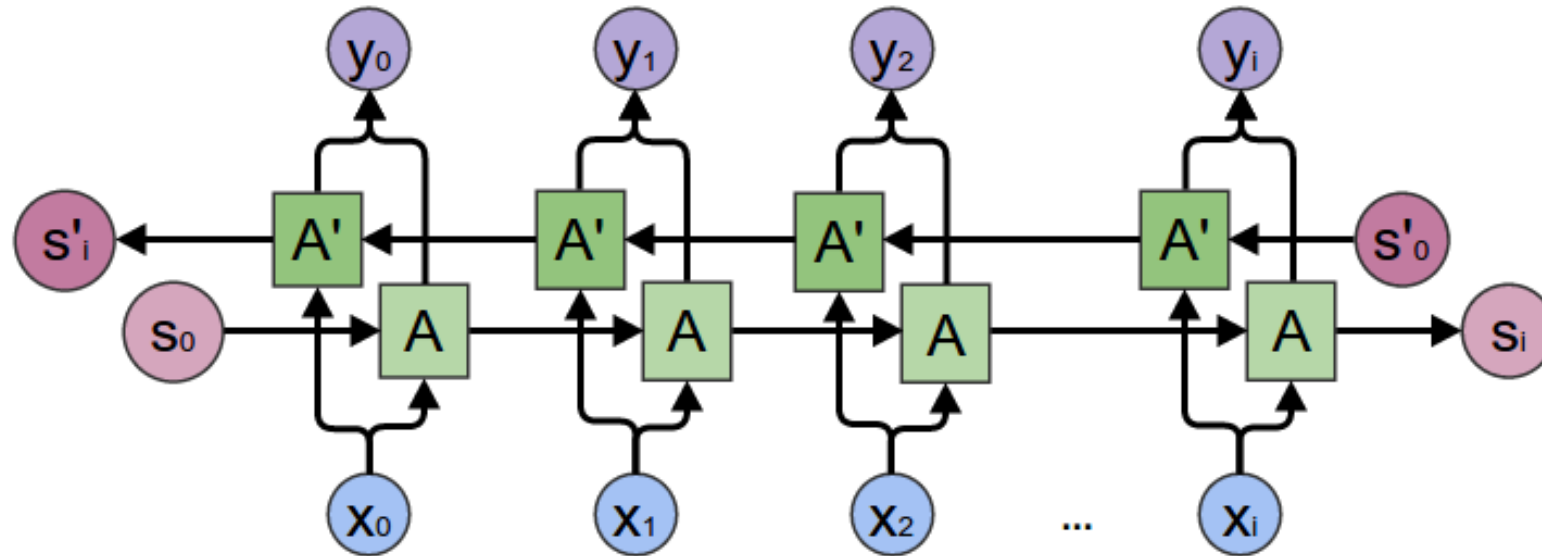
- Two RNNs where input is fed forward and backward respectively and then the hidden states (typically) are concatenated into a hidden state



What are advantages of a bi-directional RNN compared to a single RNN?

Bahdanau's Neural Machine Translation: Encoder

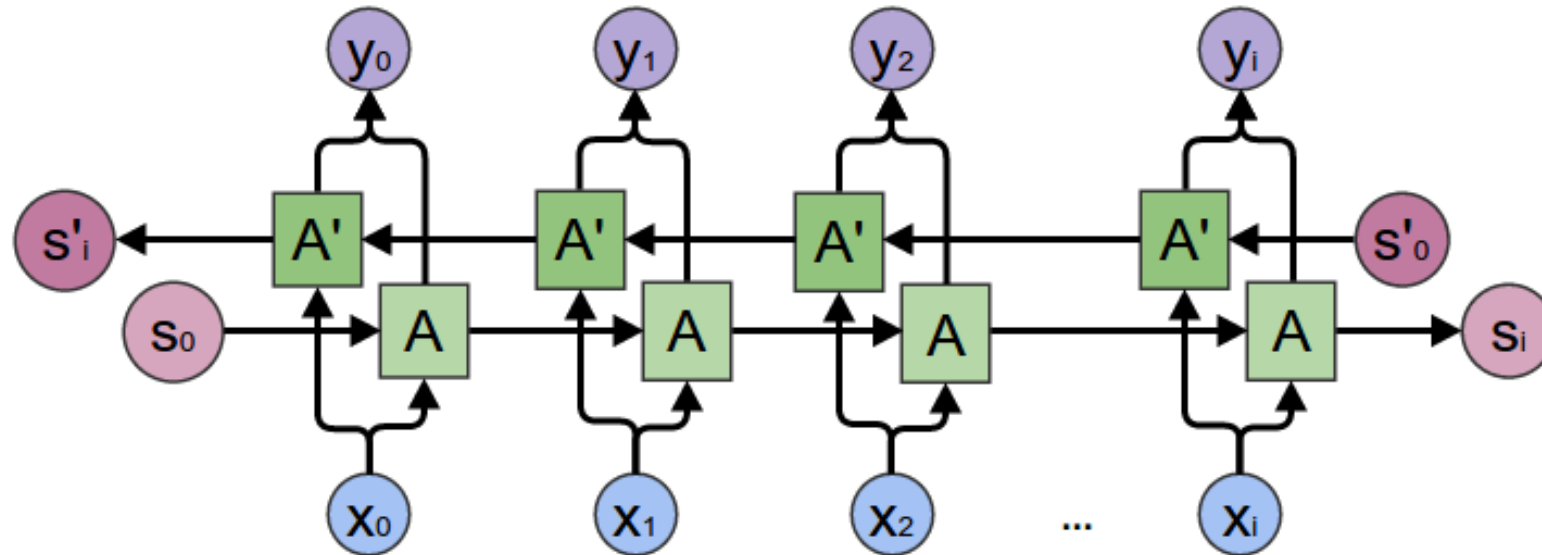
- Two RNNs where input is fed forward and backward respectively and then the hidden states (typically) are concatenated into a hidden state



Can use information from the past and **future** to make predictions: e.g., can resolve for "Teddy is a ...?" if Teddy refers to a "bear" or former US President Roosevelt

Bahdanau's Neural Machine Translation: Encoder

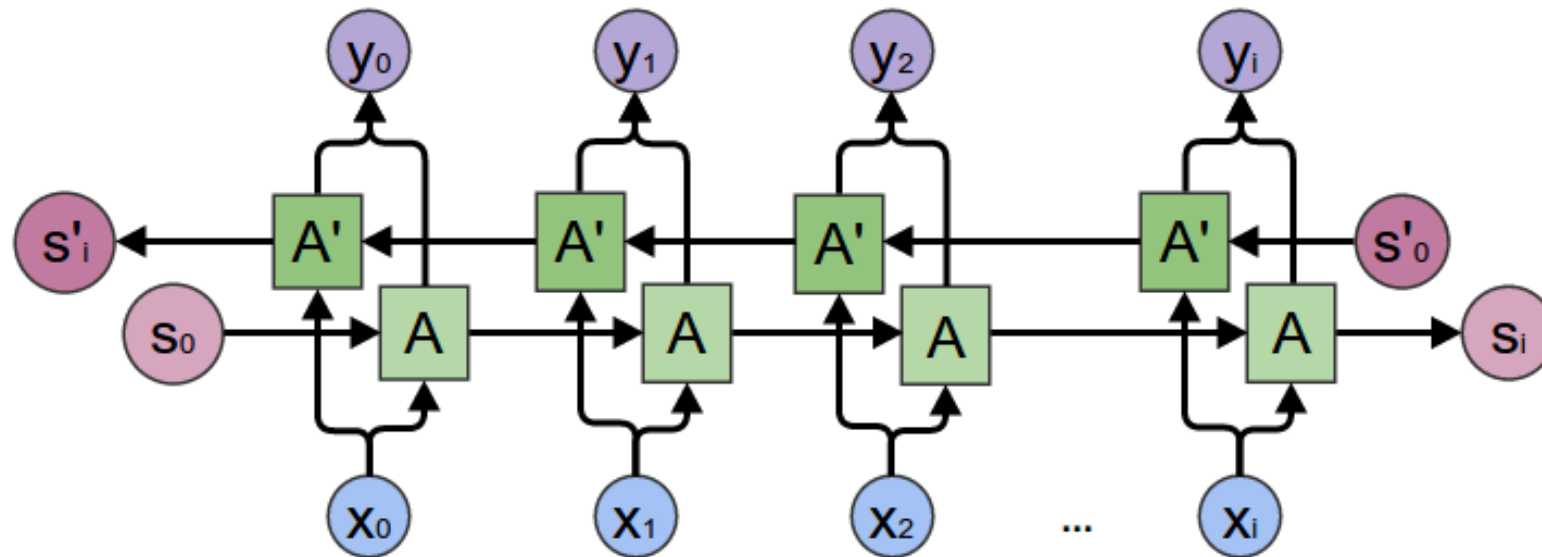
- Two RNNs where input is fed forward and backward respectively and then the hidden states (typically) are concatenated into a hidden state



What are disadvantages of a bi-directional RNN compared to a single RNN?

Bahdanau's Neural Machine Translation: Encoder

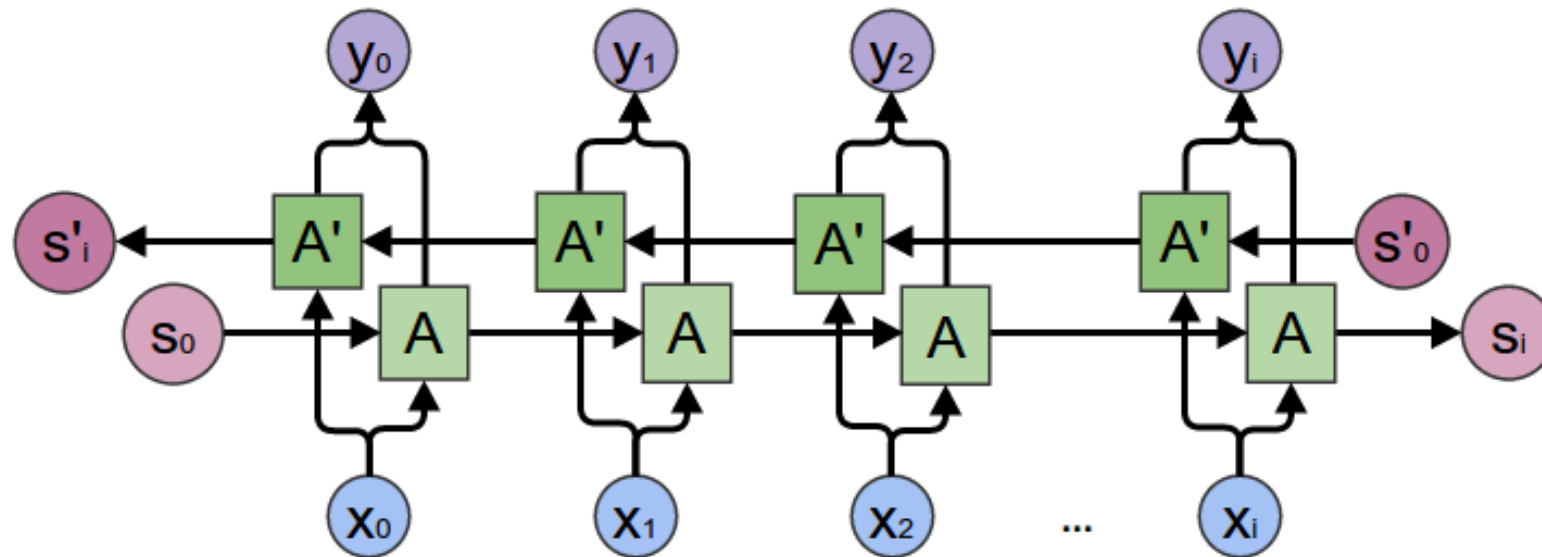
- Two RNNs where input is fed forward and backward respectively and then the hidden states (typically) are concatenated into a hidden state



Entire sequence must be observed to make a prediction (e.g., unsuitable for text prediction)

Bahdanau's Neural Machine Translation: Encoder

- Two RNNs where input is fed forward and backward respectively and then the hidden states (typically) are concatenated into a hidden state

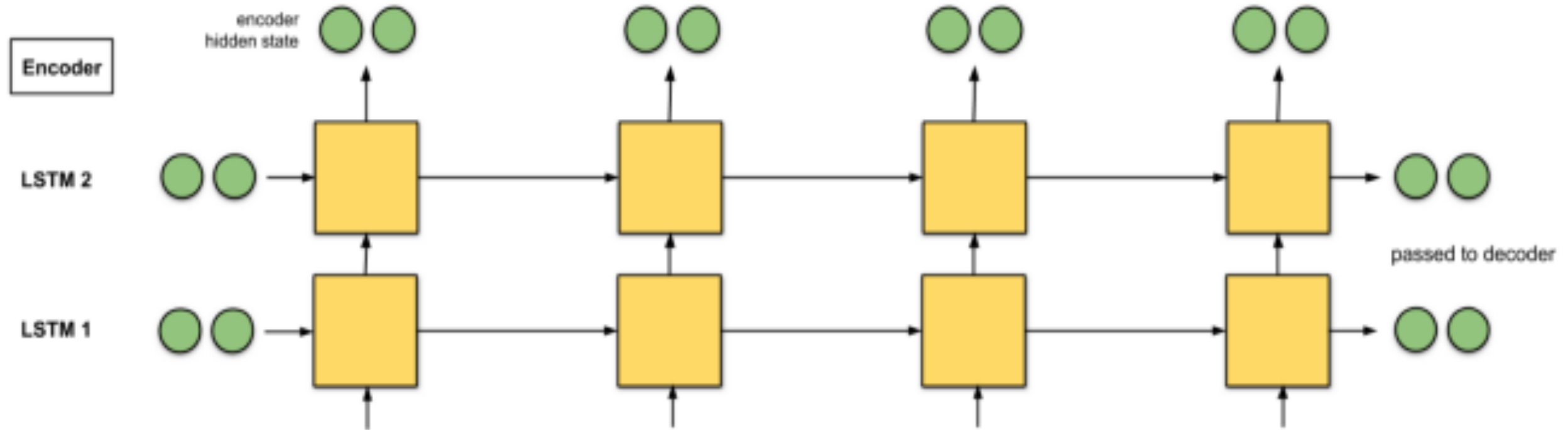


Bahdanau's method encodes input with a bidirectional GRU

Popular Choices for Encoding Input

- Bi-directional RNN (Bahdanau)
- **Stacked RNNs (Luong)**
- Bi-directional and Stacked RNN (Google)

Luong's Neural Machine Translation: Encoder



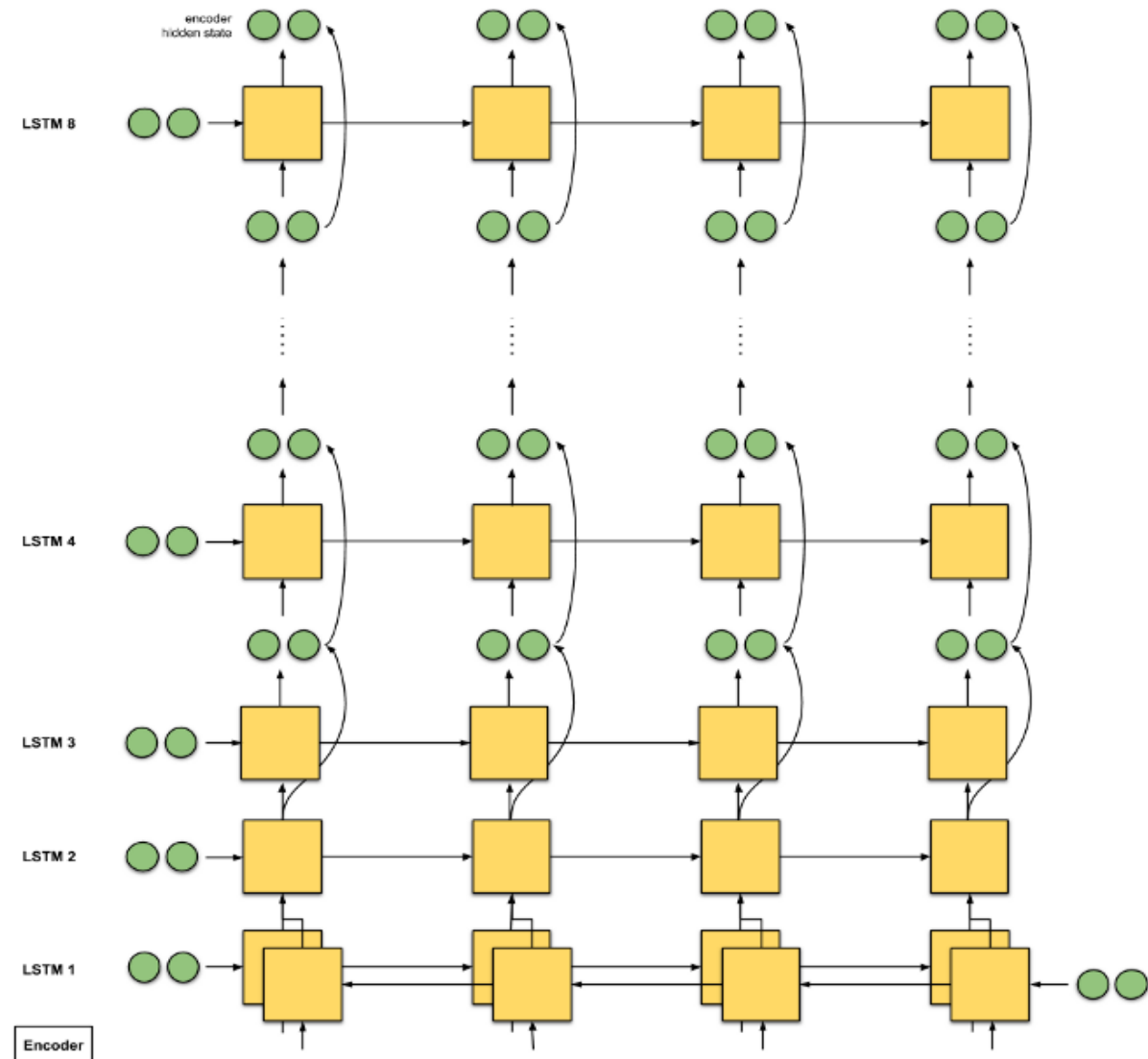
Luong's method encodes input with a 2-layer stacked LSTM

Popular Choices for Encoding Input

- Bi-directional RNN (Bahdanau)
- Stacked RNNs (Luong)
- Bi-directional and Stacked RNN (Google)

Google's Neural Machine Translation: Encoder

8 layers with 1st layer bi-directional
and skip connections between layers
(greater level of abstraction for input)



Wu et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv 2016.

<https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3#df28>

Popular Choices for Encoding Input

- Bi-directional RNN (Bahdanau)
- Stacked RNNs (Luong)
- Bi-directional and Stacked RNN (Google)

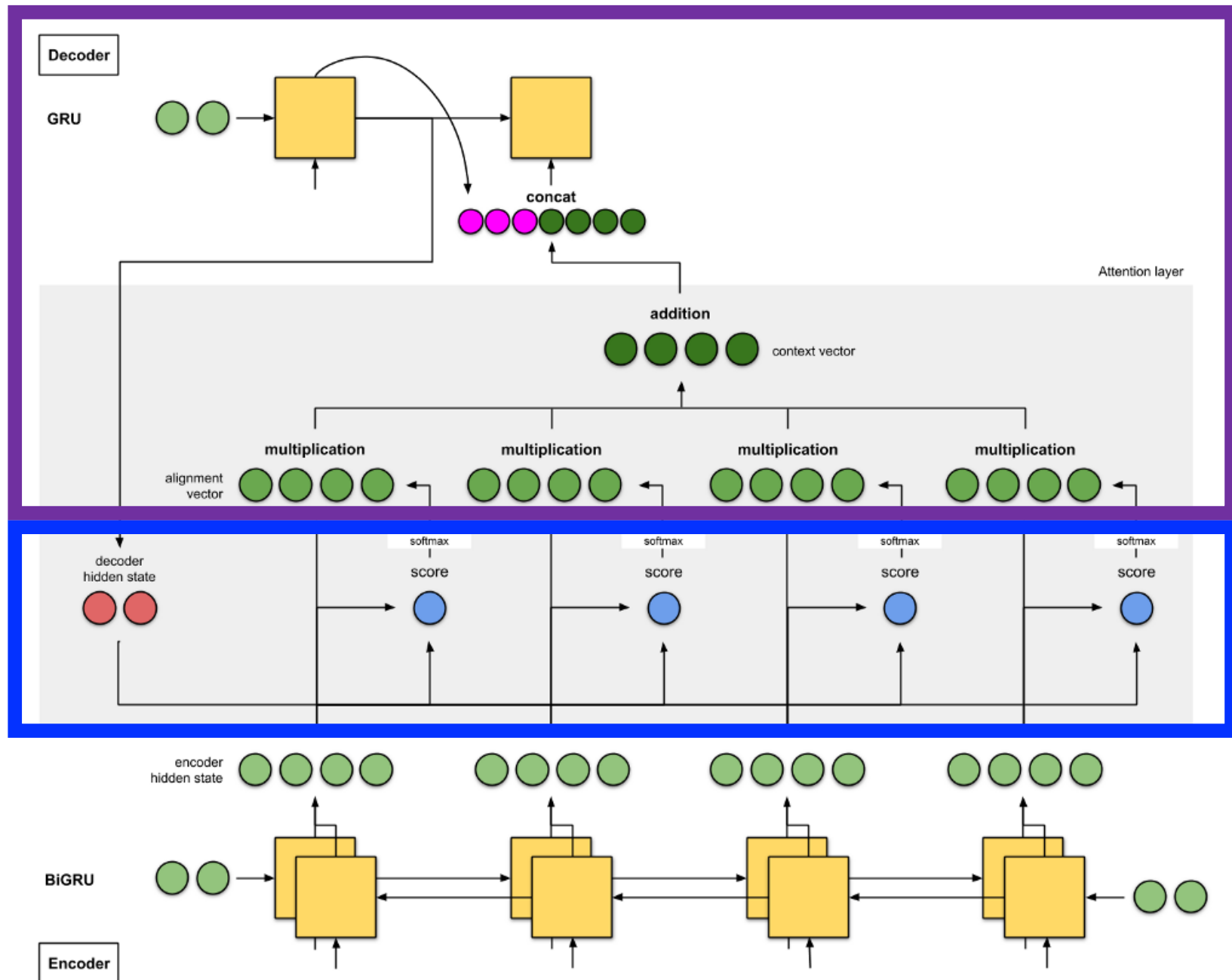
Today's Topics

- Motivation: machine neural translation for long sentences
- Encoder
- **Decoder: attention**
- Performance evaluation

Solution

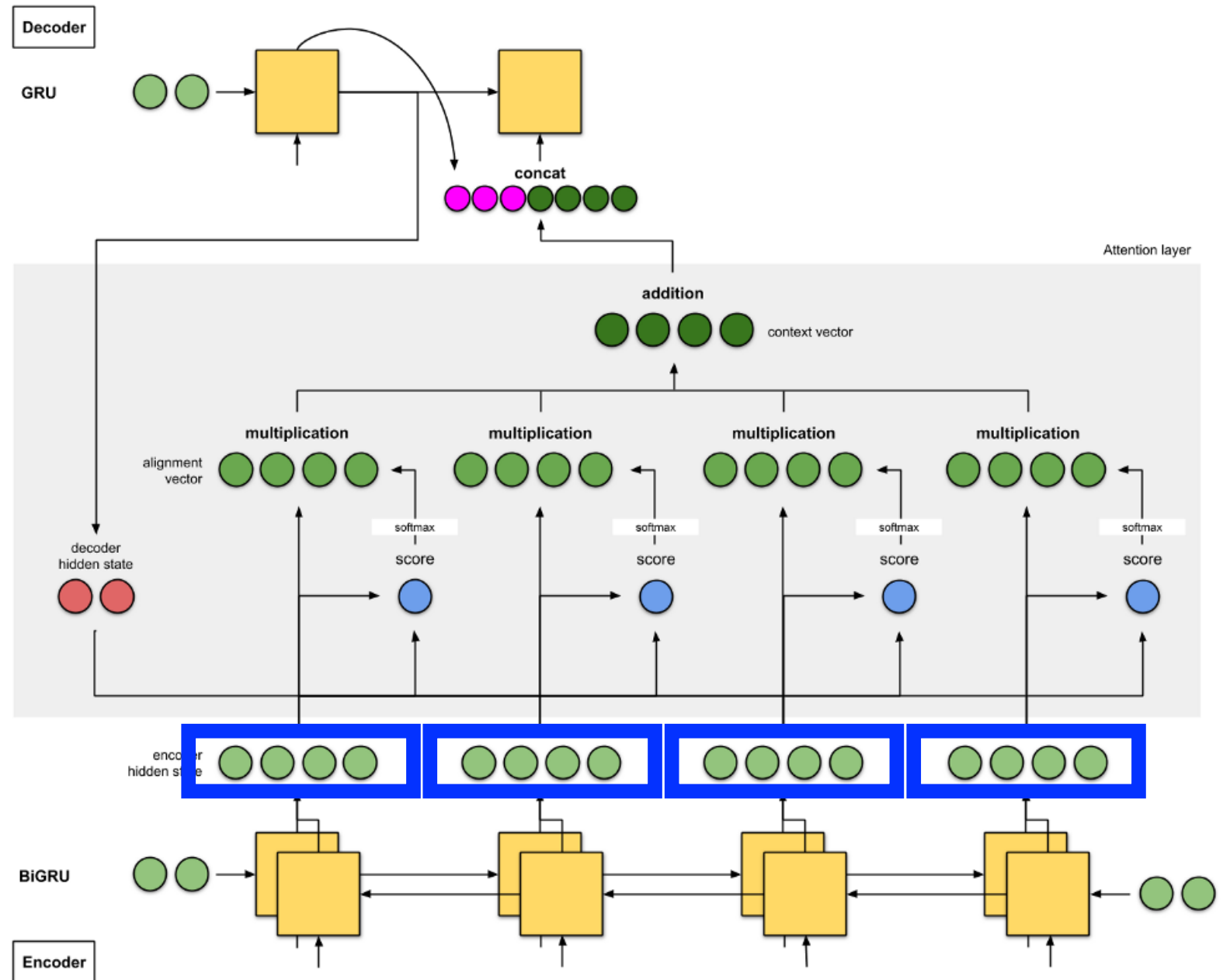
3. At each decoder time step, a prediction is made based on the weighted sum of the inputs

2. At each decoder time step, attention weights are computed that determine each input's relevance for the prediction



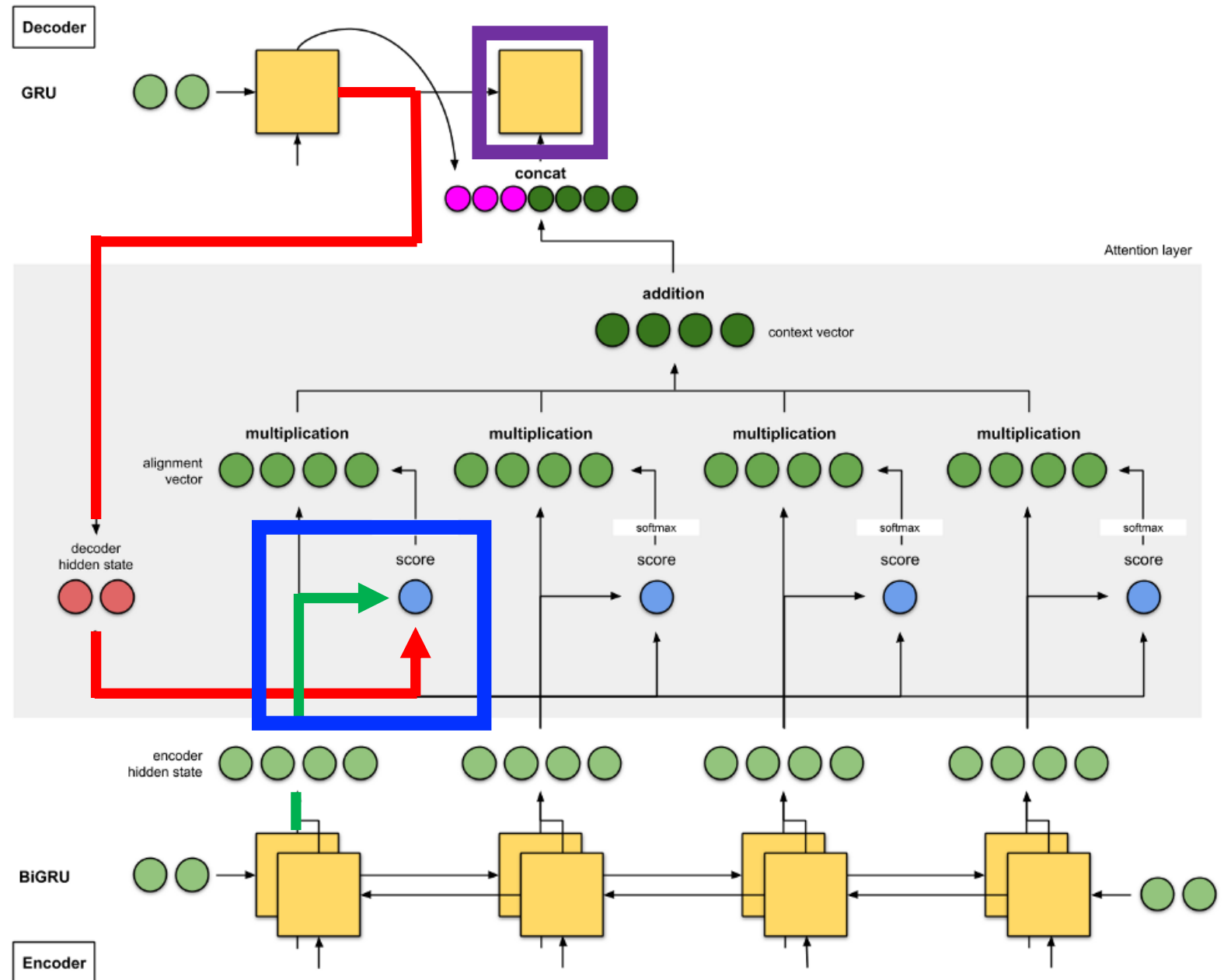
Measuring Each Input's Relevance on the Prediction

How many inputs are in this example?



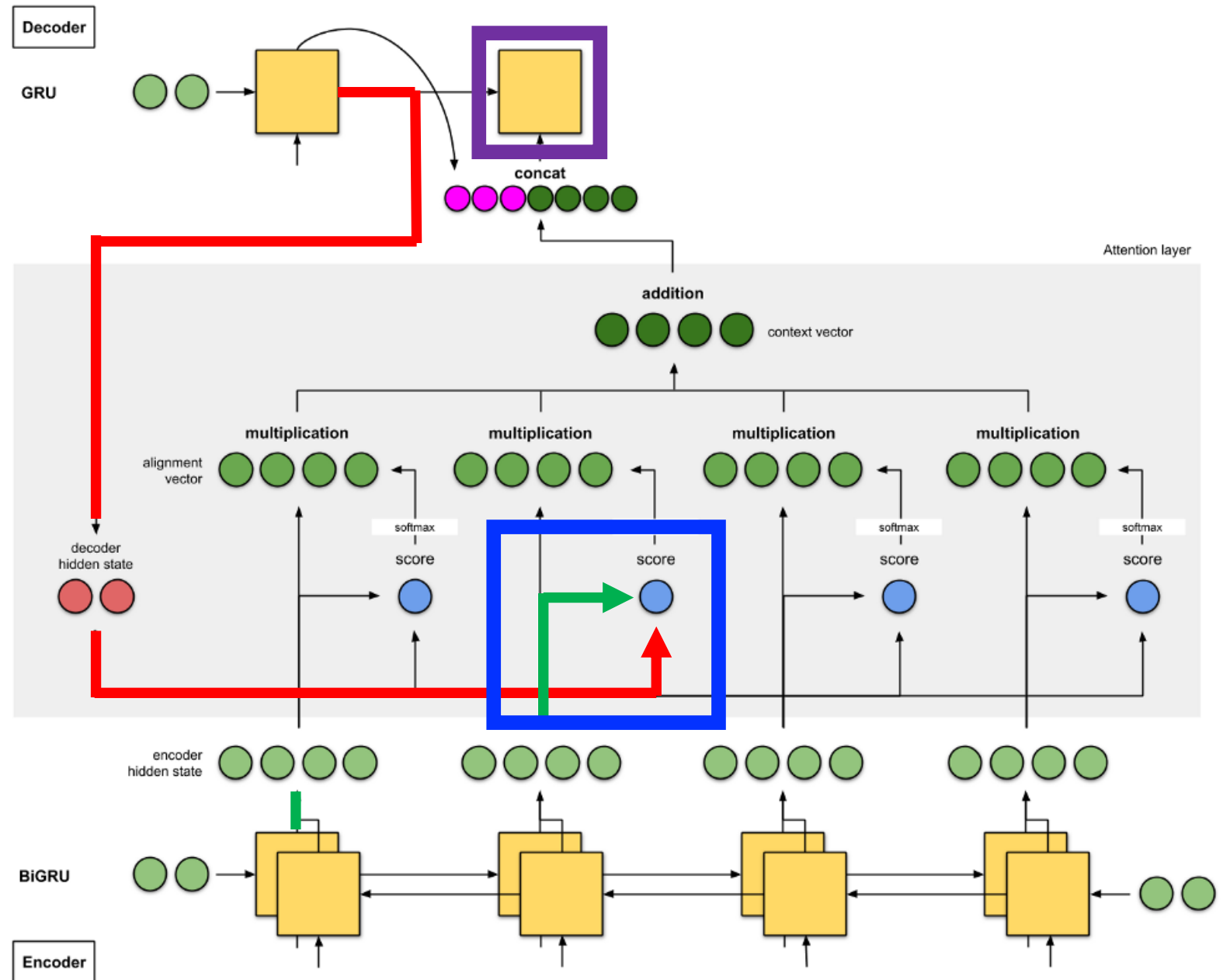
Measuring Each Input's Relevance on the Prediction

At each **decoder time step**, the similarity between the **decoder's hidden state** and each **input's hidden state** is computed to decide each input's score at the time step



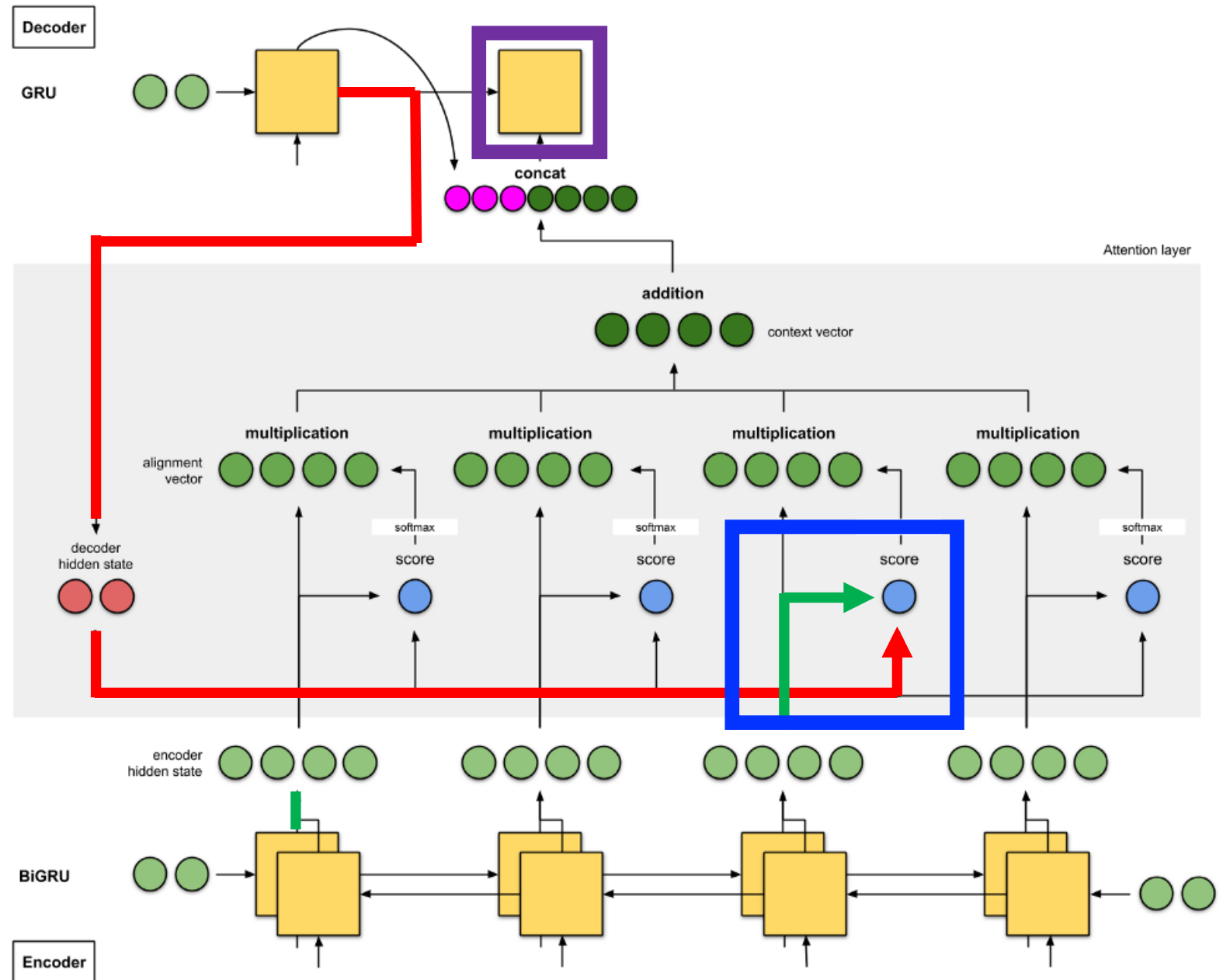
Measuring Each Input's Relevance on the Prediction

At each **decoder time step**, the similarity between the **decoder's hidden state** and each **input's hidden state** is computed to decide each input's score at the time step



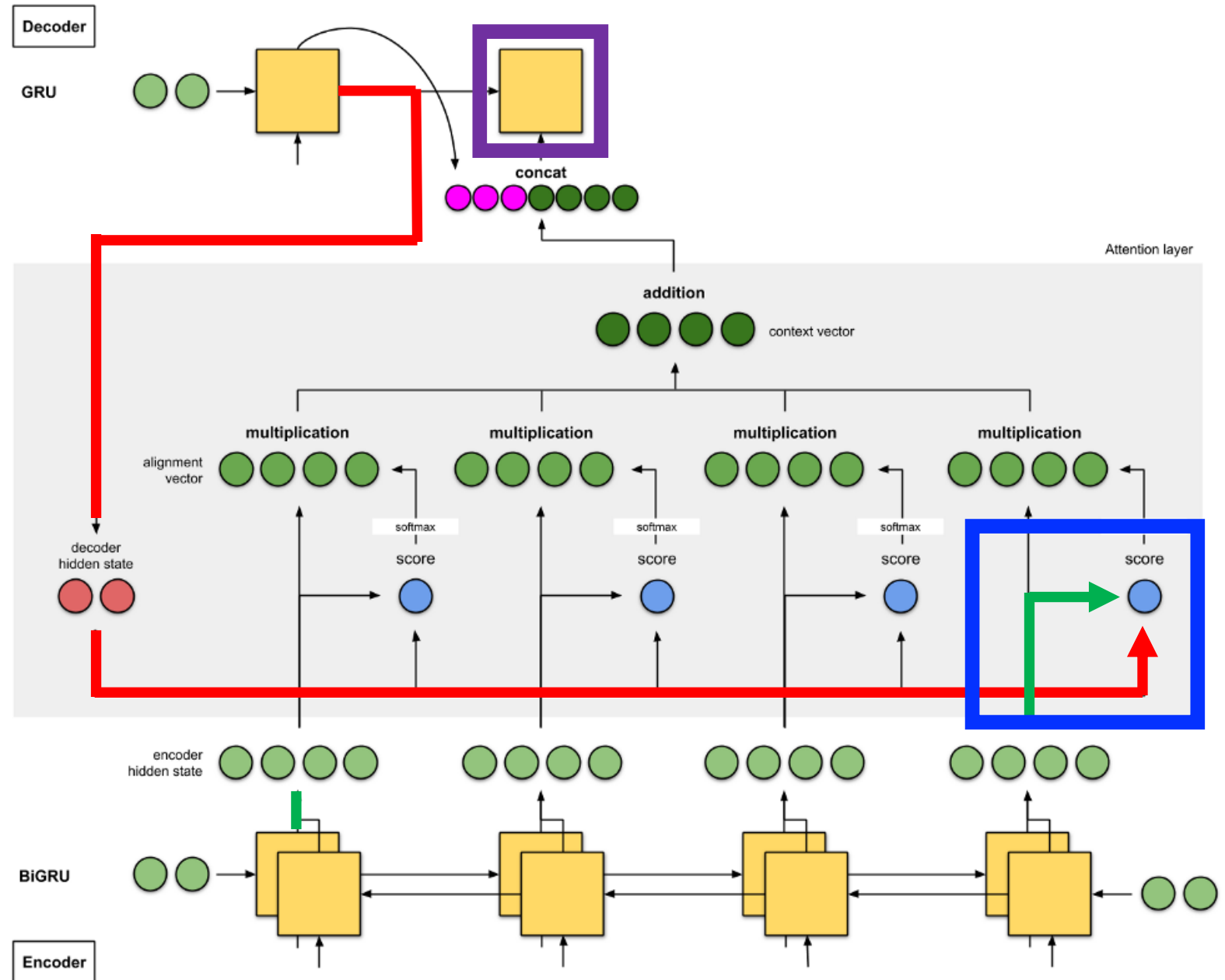
Measuring Each Input's Relevance on the Prediction

At each **decoder time step**, the similarity between the **decoder's hidden state** and each **input's hidden state** is computed to decide each input's score at the time step



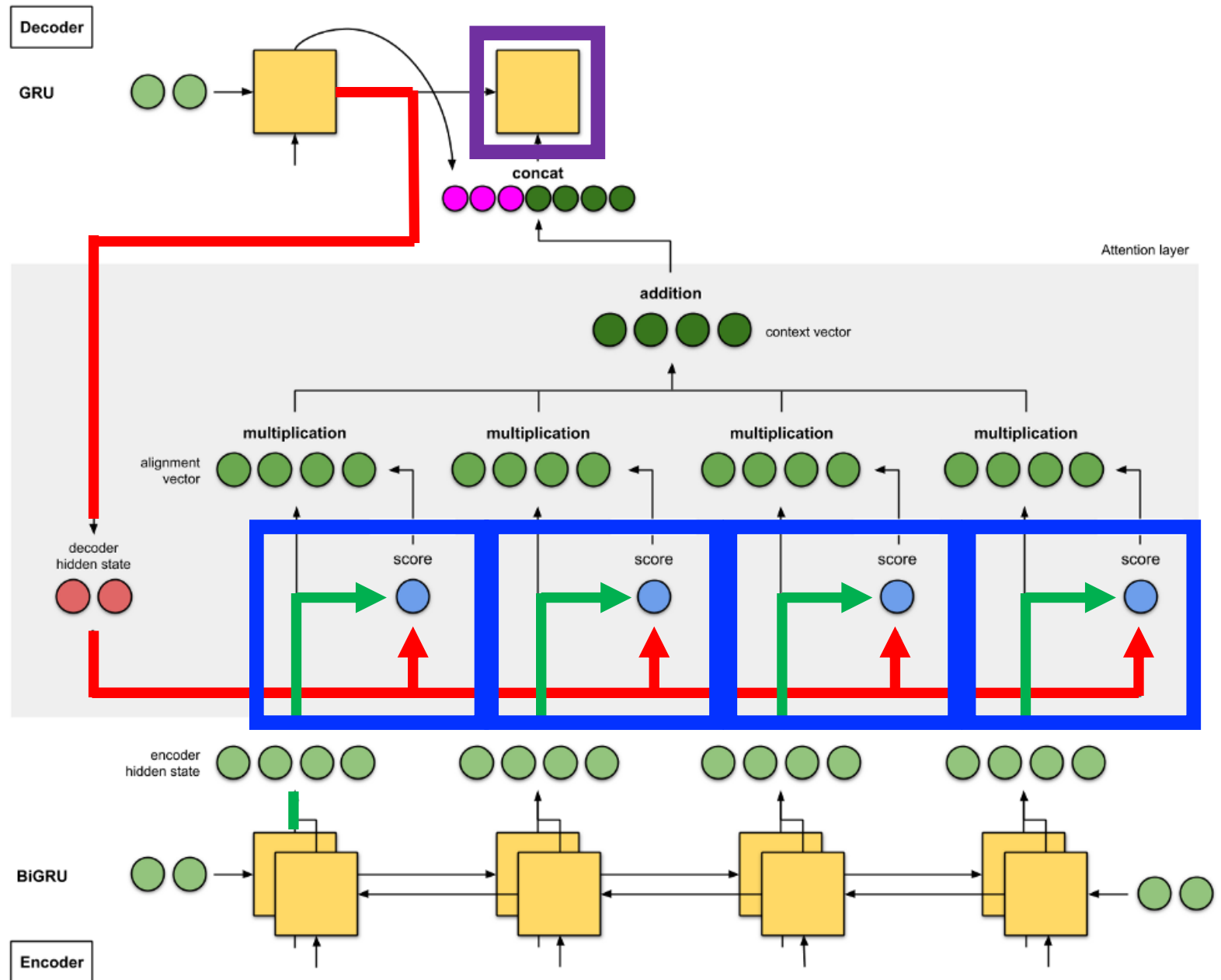
Measuring Each Input's Relevance on the Prediction

At each **decoder time step**, the similarity between the **decoder's hidden state** and each **input's hidden state** is computed to decide each input's score at the time step

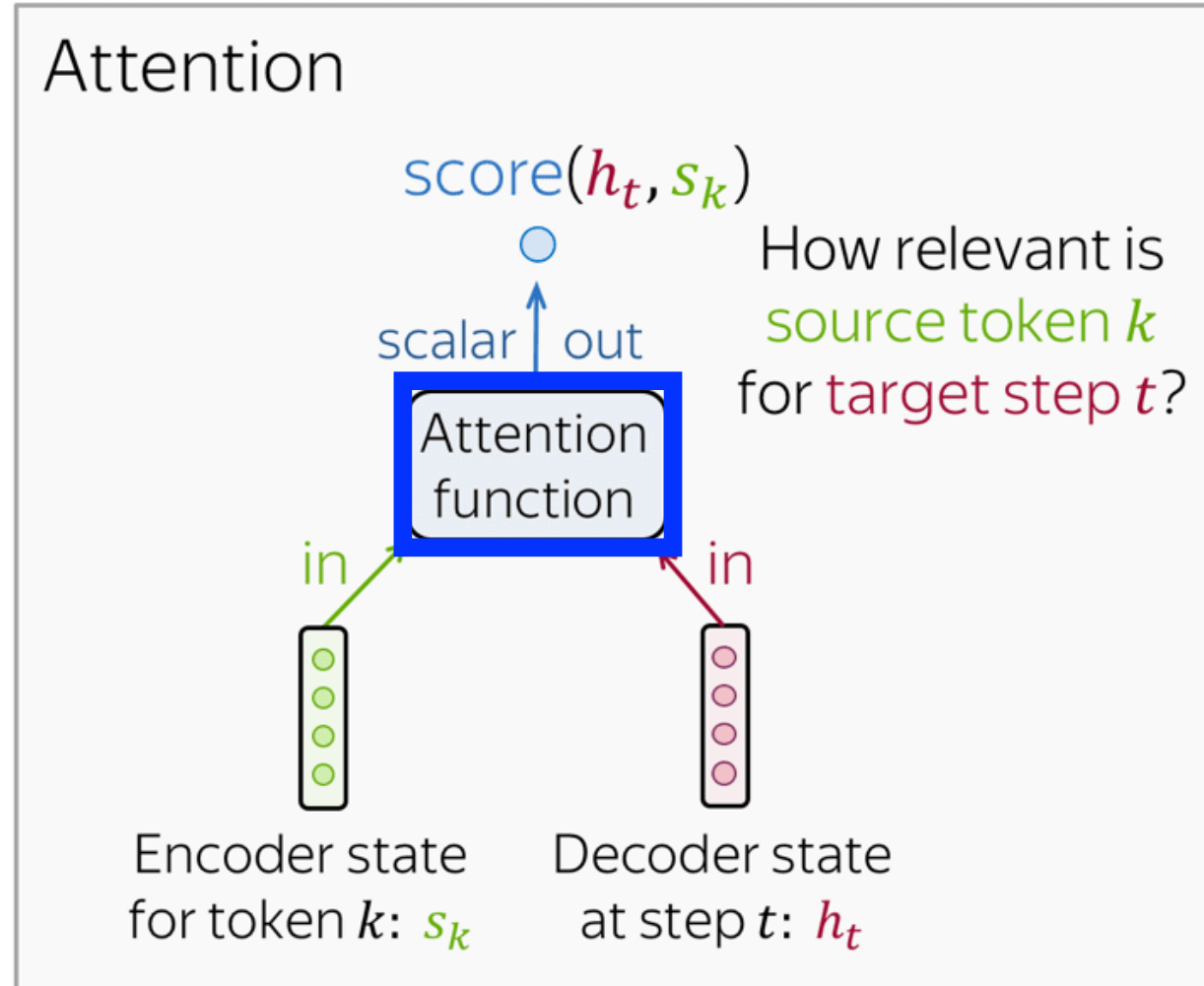


Measuring Each Input's Relevance on the Prediction

How to measure the similarity between hidden states of the **decoder** and **input**?



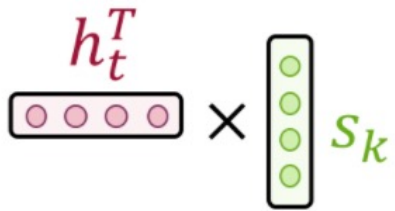
Similarity Measure for Hidden States of the Decoder and Encoder



Similarity Measure for Hidden States of the Decoder and Encoder

- Many options (function should be differentiable)

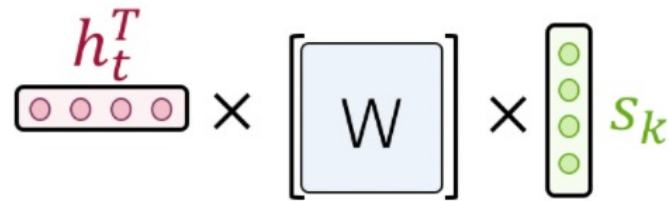
Dot-product



The diagram shows a horizontal vector of four pink circles labeled h_t^T and a vertical vector of four green circles labeled s_k . They are connected by a multiplication symbol \times .

$$\text{score}(h_t, s_k) = h_t^T s_k$$

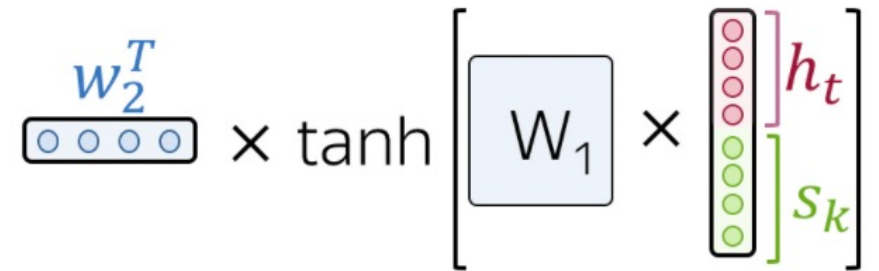
Bilinear



The diagram shows a horizontal vector of four pink circles labeled h_t^T , a light blue square labeled W , and a vertical vector of four green circles labeled s_k . They are connected by multiplication symbols \times .

$$\text{score}(h_t, s_k) = h_t^T W s_k$$

Multi-Layer Perceptron



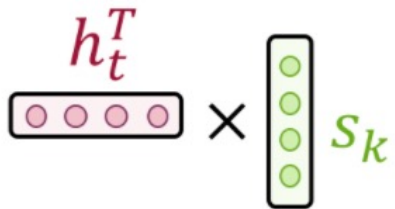
The diagram shows a horizontal vector of four blue circles labeled w_2^T , a multiplication symbol \times , a \tanh function, a light blue square labeled W_1 , another multiplication symbol \times , and a vertical vector of eight circles (four pink labeled h_t and four green labeled s_k) enclosed in large square brackets.

$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$

Similarity Measure for Hidden States of the Decoder and Encoder

- Many options (function should be differentiable)

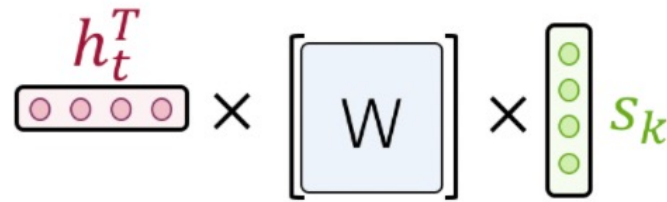
Dot-product



A diagram illustrating the dot-product similarity measure. It shows a horizontal vector h_t^T with four pink circles, followed by a multiplication symbol \times , and then a vertical vector s_k with four green circles.

$$\text{score}(h_t, s_k) = h_t^T s_k$$

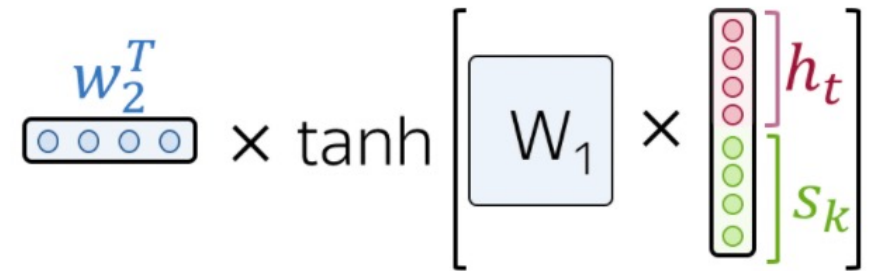
Bilinear



A diagram illustrating the bilinear similarity measure. It shows a horizontal vector h_t^T with four pink circles, followed by a multiplication symbol \times , then a light blue square matrix W , followed by another multiplication symbol \times , and finally a vertical vector s_k with four green circles.

$$\text{score}(h_t, s_k) = h_t^T W s_k$$

Multi-Layer Perceptron



A diagram illustrating the Multi-Layer Perceptron similarity measure. It shows a horizontal vector w_2^T with four blue circles, followed by a multiplication symbol \times , then the word \tanh , followed by a large square bracket containing a light blue square matrix W_1 multiplied by a vertical vector. The vertical vector is split into two parts: the top part has four pink circles and is labeled h_t , and the bottom part has four green circles and is labeled s_k .

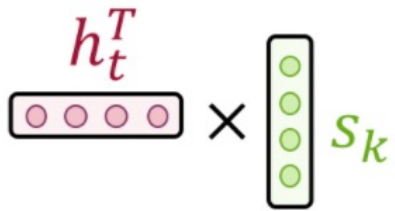
$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$

What model parameters must be learned when using dot-product?

Similarity Measure for Hidden States of the Decoder and Encoder

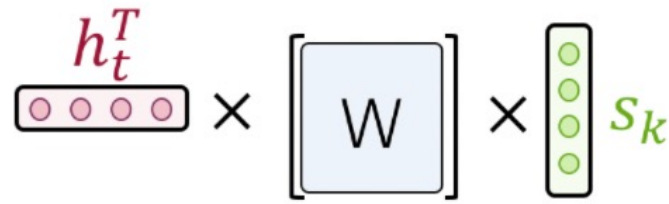
- Many options (function should be differentiable)

Dot-product



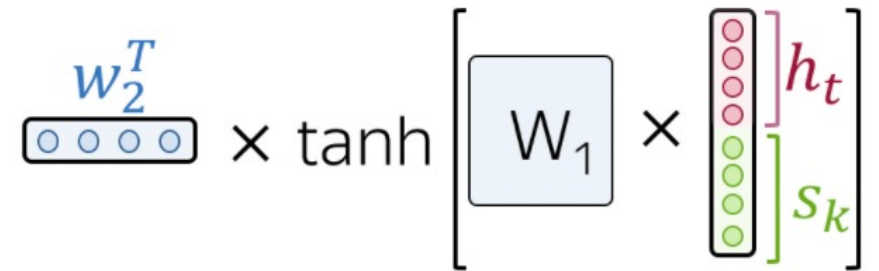
$$\text{score}(h_t, s_k) = h_t^T s_k$$

Bilinear



$$\text{score}(h_t, s_k) = h_t^T W s_k$$

Multi-Layer Perceptron



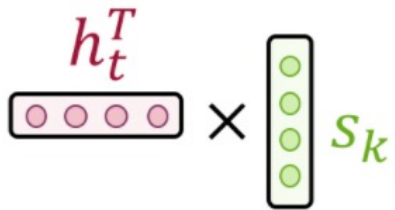
$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$

What model parameters must be learned when using bilinear?

Similarity Measure for Hidden States of the Decoder and Encoder

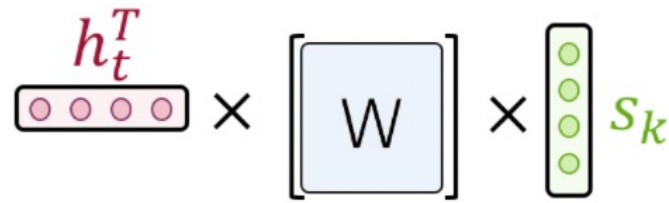
- Many options (function should be differentiable)

Dot-product



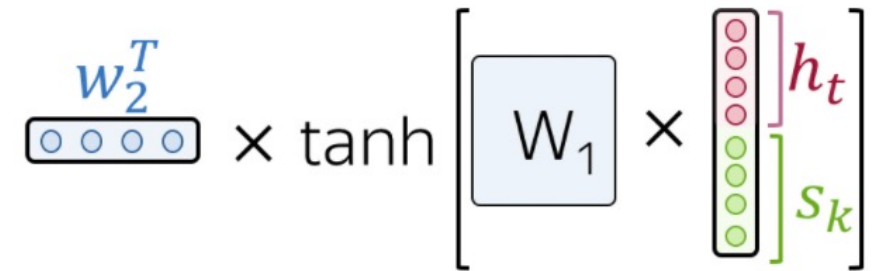
$$\text{score}(h_t, s_k) = h_t^T s_k$$

Bilinear



$$\text{score}(h_t, s_k) = h_t^T W s_k$$

Multi-Layer Perceptron



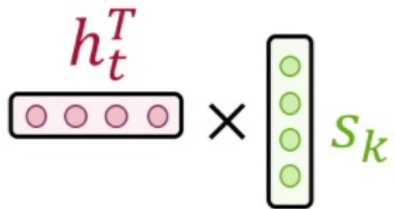
$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh(W_1 [h_t, s_k])$$

What model parameters must be learned when using multi-layer perceptron?

Similarity Measure for Hidden States of the Decoder and Encoder

- Many options (function should be differentiable)

Dot-product

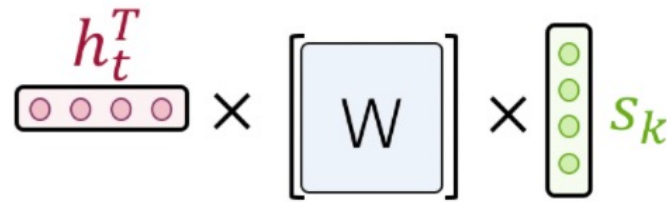


A diagram showing a horizontal vector h_t^T with four pink circles and a vertical vector s_k with four green circles. They are separated by a multiplication symbol \times .

$$\text{score}(h_t, s_k) = h_t^T s_k$$

(no parameters)

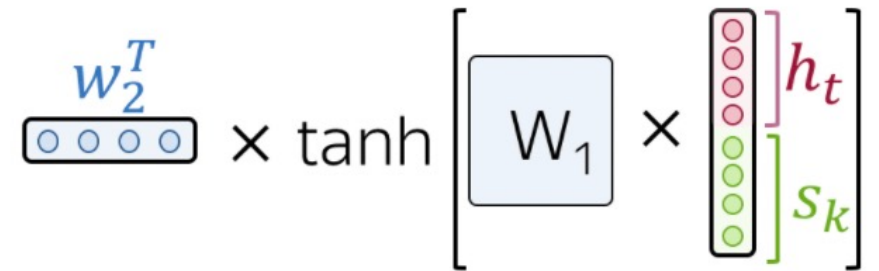
Bilinear



A diagram showing a horizontal vector h_t^T with four pink circles, a square matrix W with a light blue background, and a vertical vector s_k with four green circles. They are connected by multiplication symbols \times .

$$\text{score}(h_t, s_k) = h_t^T W s_k$$

Multi-Layer Perceptron



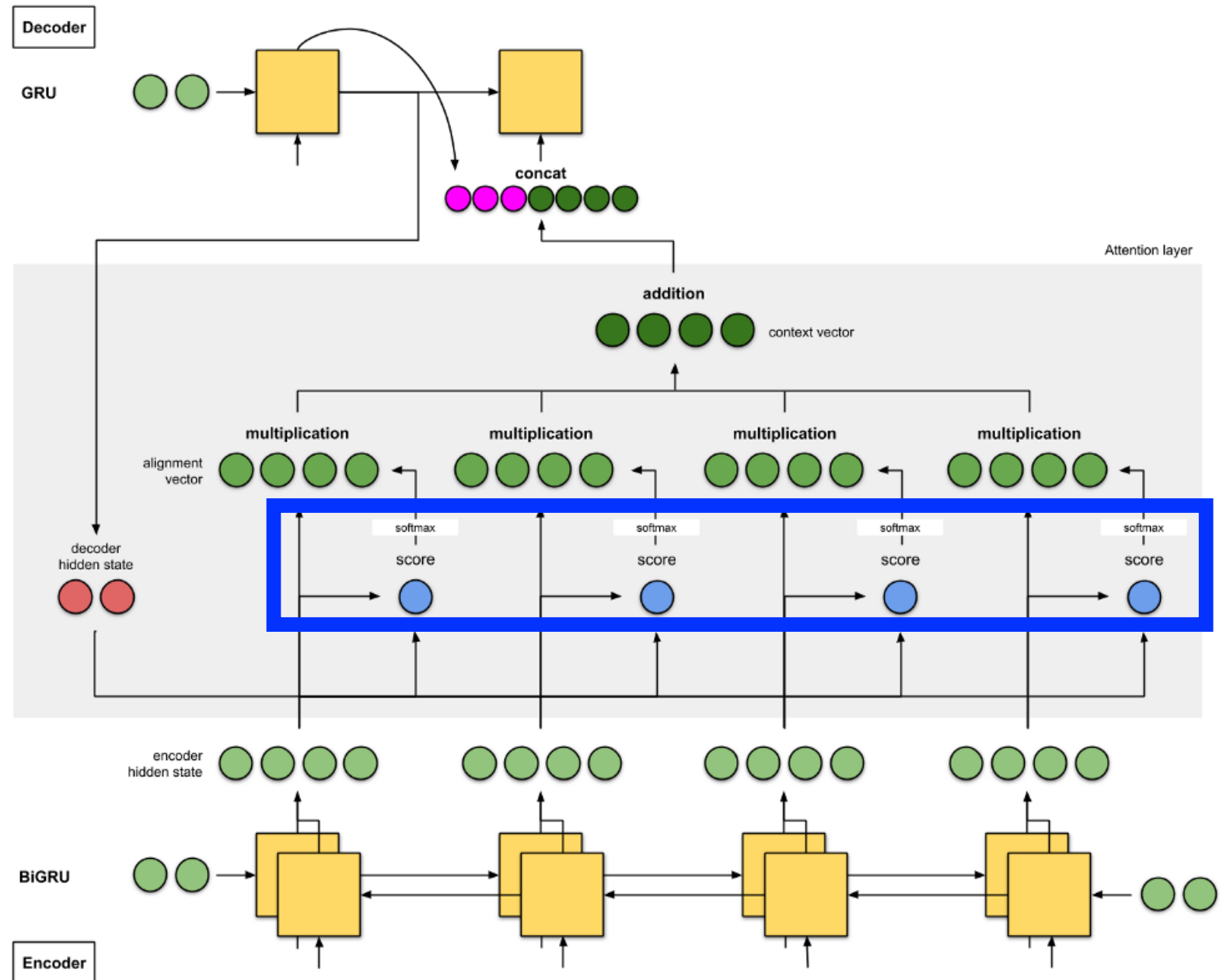
A diagram showing a horizontal vector w_2^T with four blue circles, a multiplication symbol \times , a \tanh activation function, a square matrix W_1 with a light blue background, another multiplication symbol \times , and a vertical vector with four pink circles (labeled h_t) and four green circles (labeled s_k) stacked vertically. The entire expression is enclosed in large square brackets.

$$\text{score}(h_t, s_k) = w_2^T \cdot \tanh[W_1 h_t, s_k]$$

Model parameters that must be learned

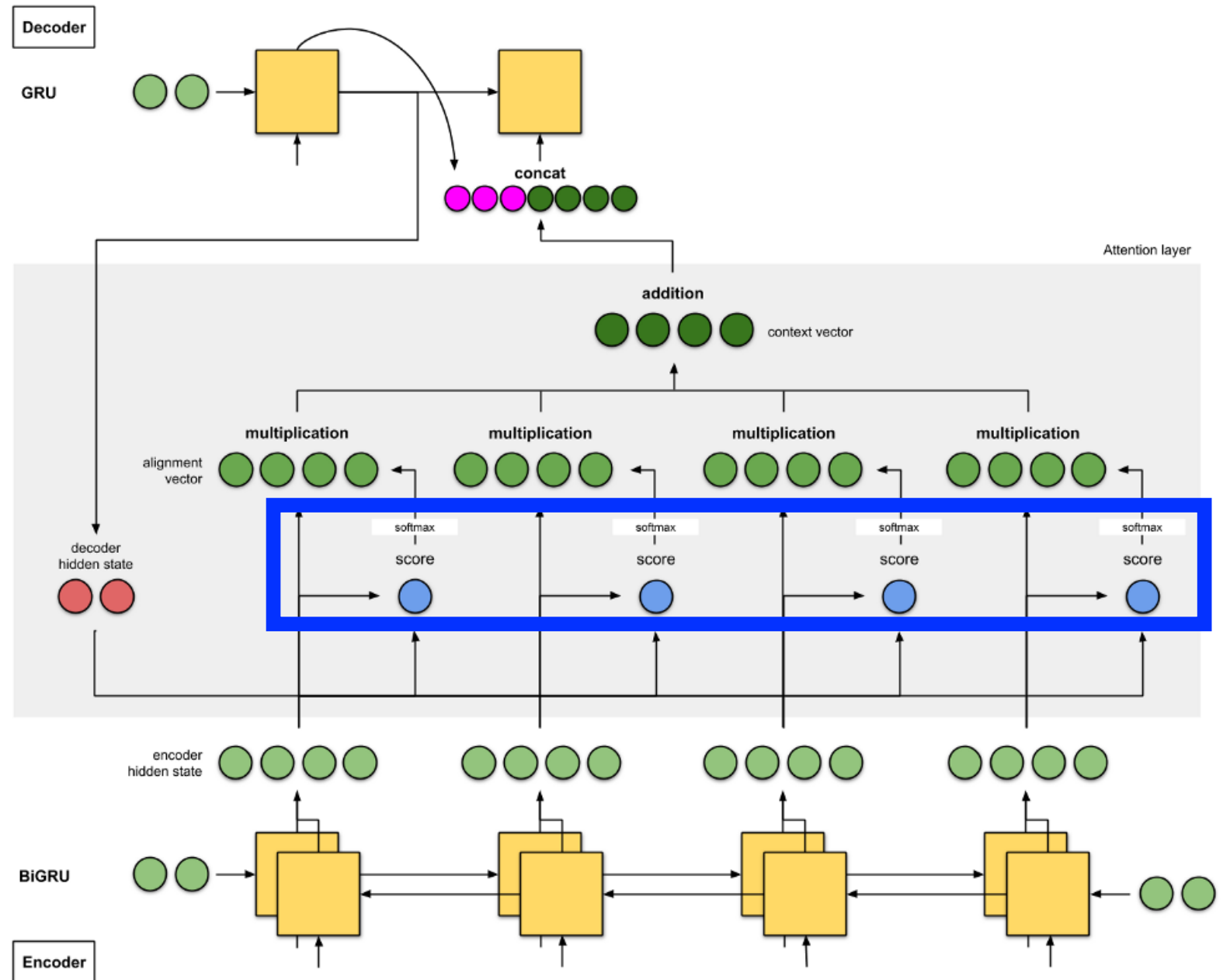
Measuring Each Input's Influence on the Prediction

After computing the similarity scores for each input, then apply softmax to all scores so all inputs' weights sum to 1



Measuring Each Input's Influence on the Prediction

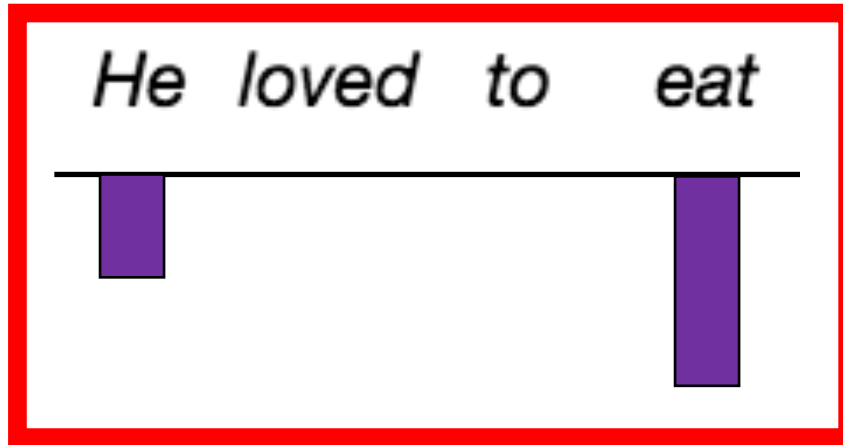
We now have our attention weights!



Measuring Each Input's Influence on the Prediction

Intuitively:

Input



The model can weight each input at each time step!

Target

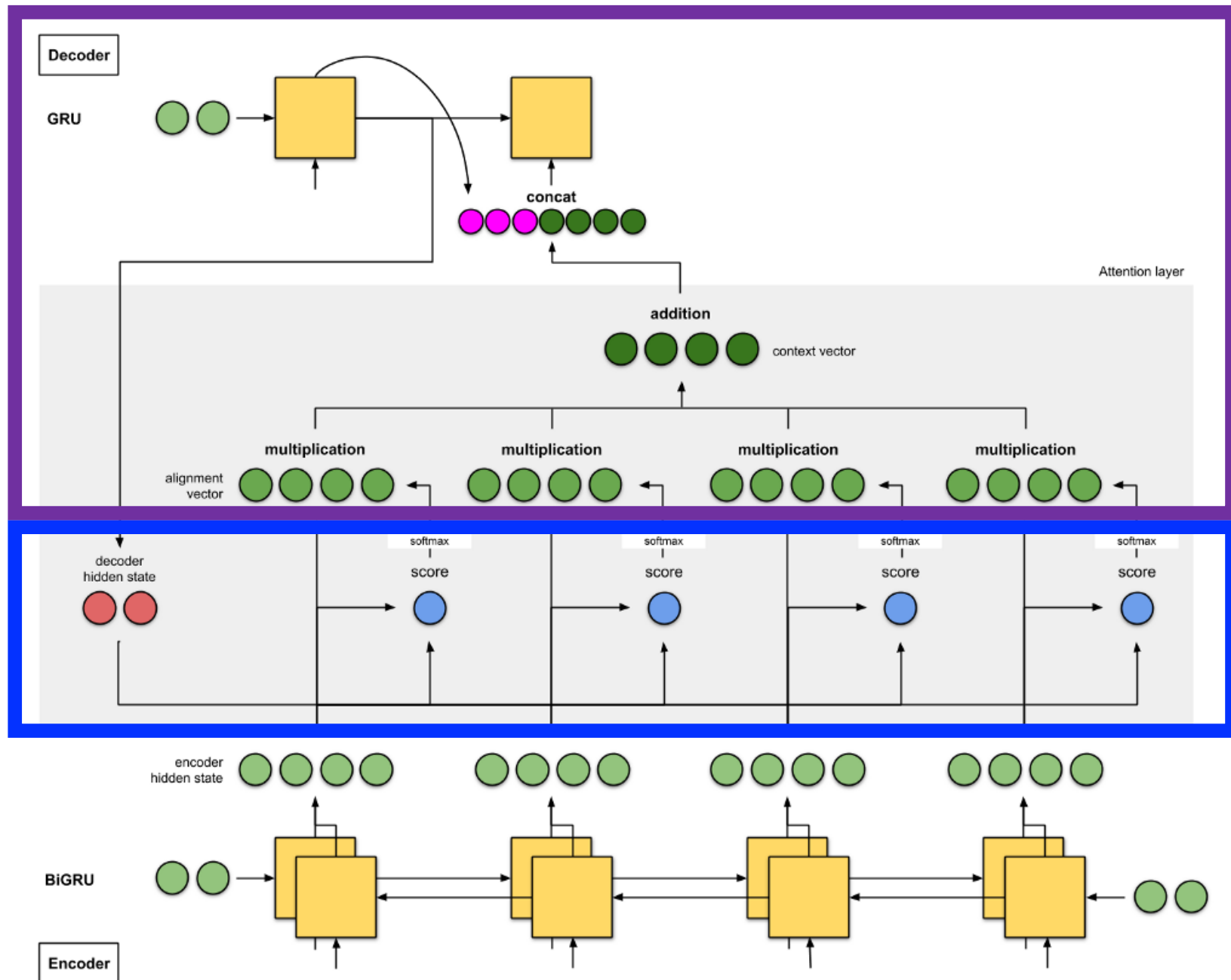
Er liebte zu essen

$t = 4$

Solution

3. At each decoder time step, a prediction is made based on the weighted sum of the inputs

2. At each decoder time step, attention weights are computed that determine each input's relevance for the prediction

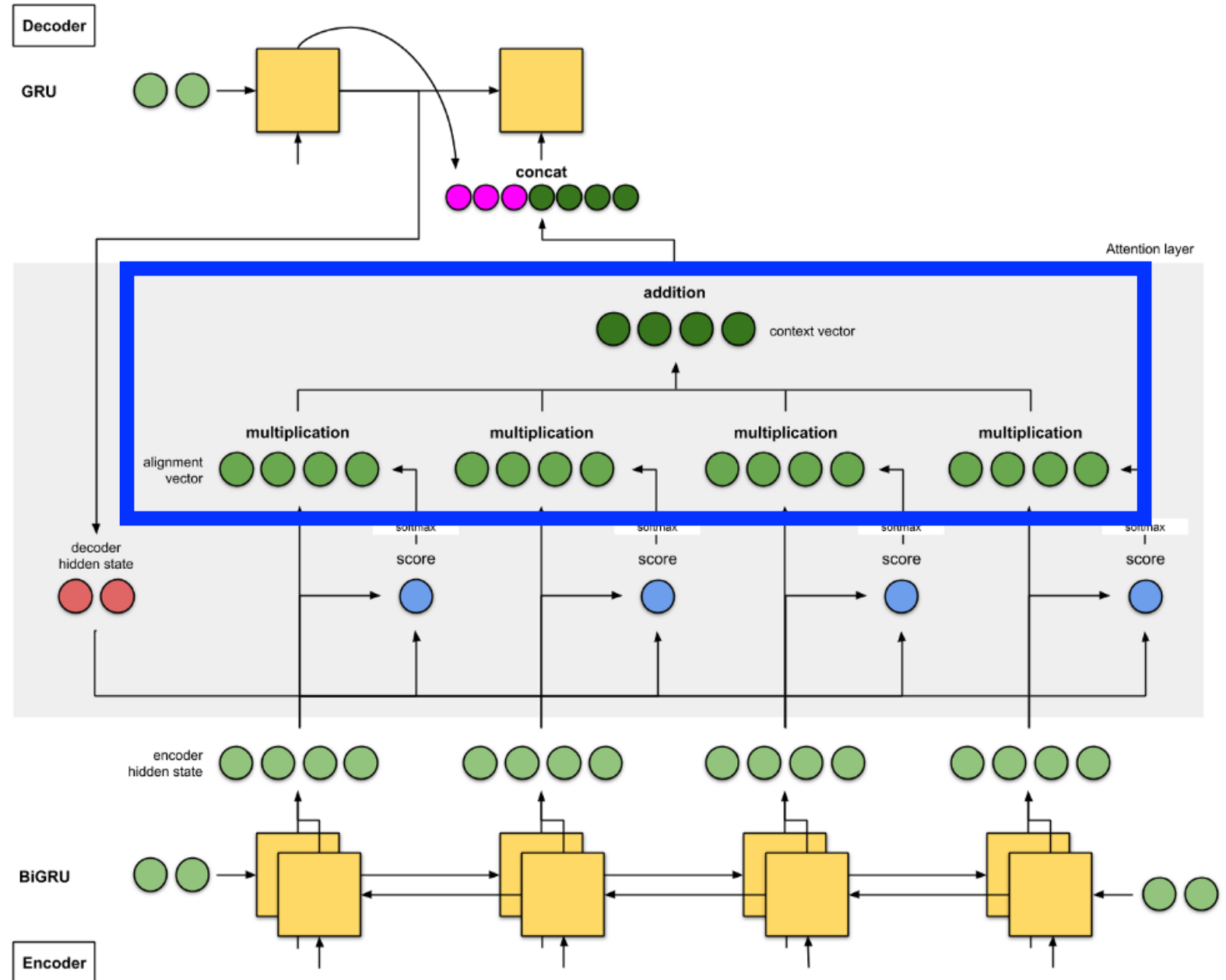


Word Prediction

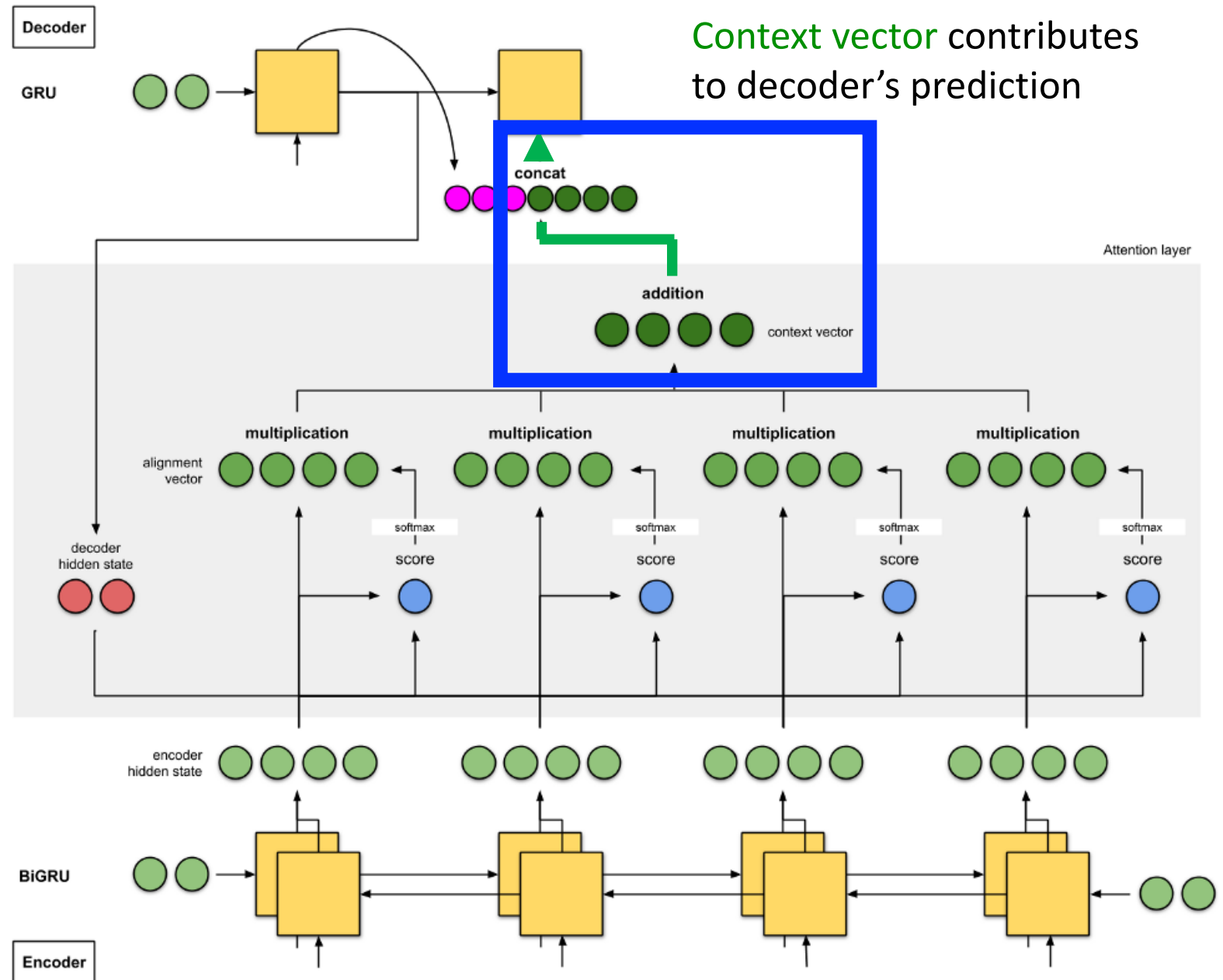
We compute at time step t for all n inputs a weighted sum:

$$\mathbf{c}_t = \sum_{i=1}^n \alpha_{t,i} \mathbf{h}_i$$

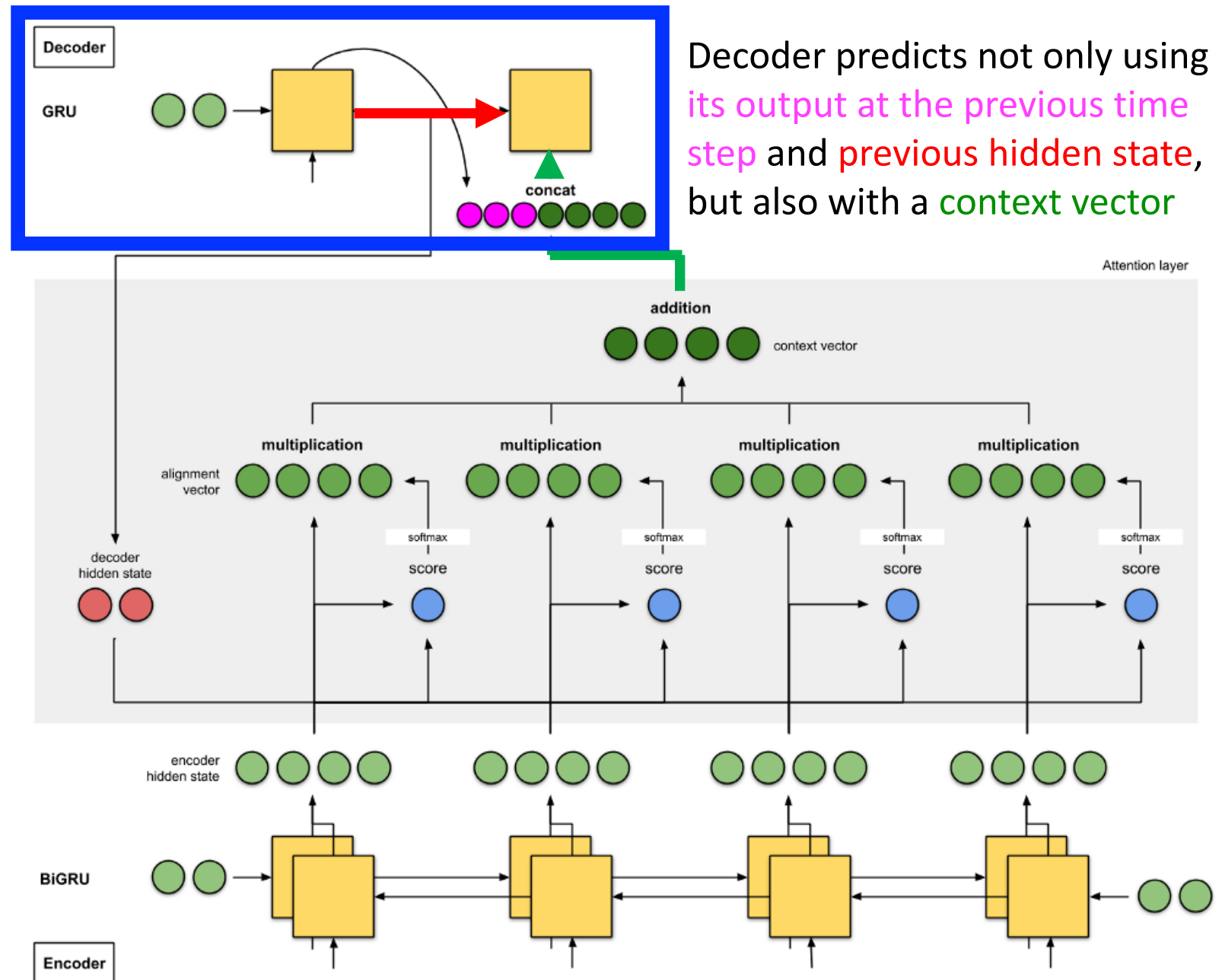
The influence of inputs are **amplified** for large attention weights and repressed otherwise



Word Prediction

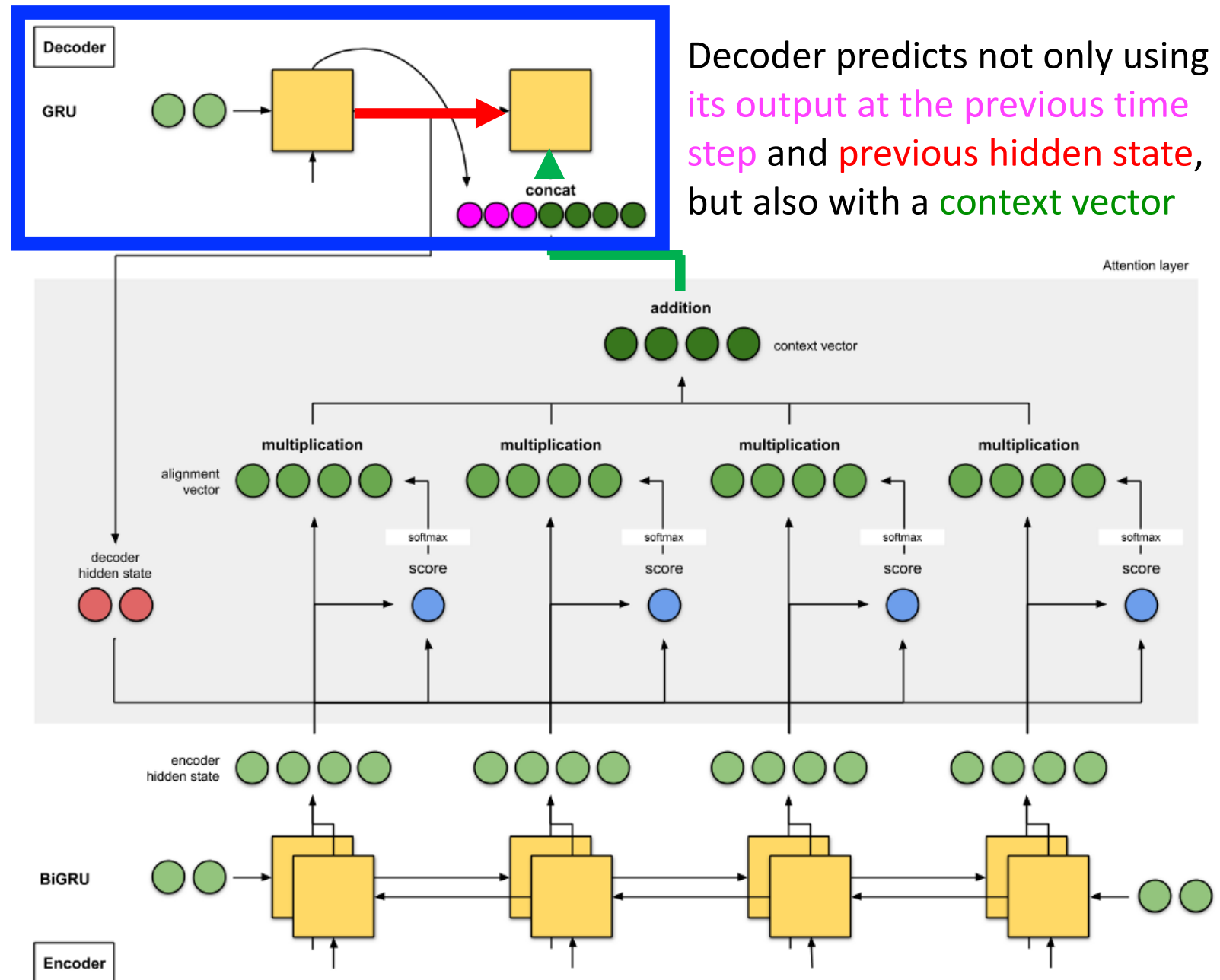


Word Prediction



Bahdanau method

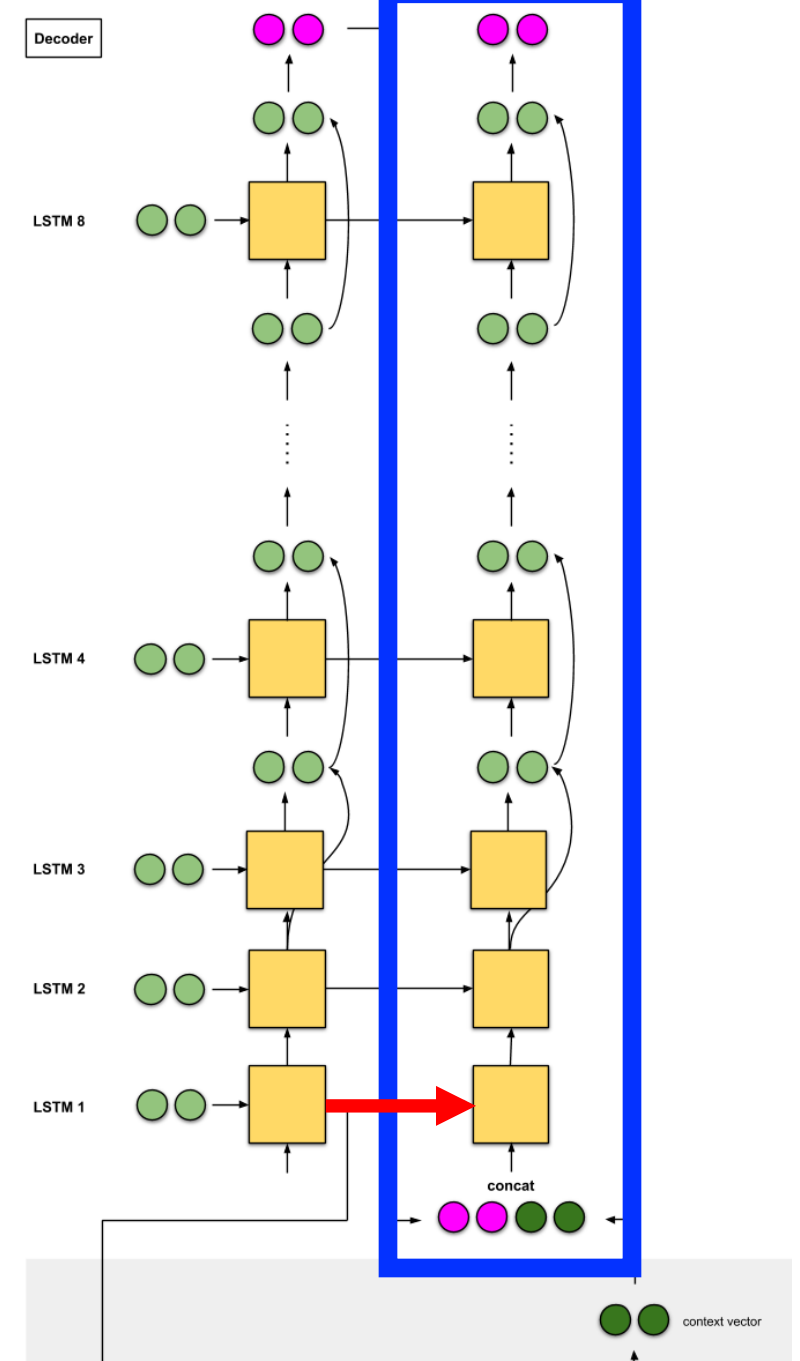
Many options exist for how to use the **context vector** with the **decoder's output at the previous time step** to produce an output at each decoder time step



Google method

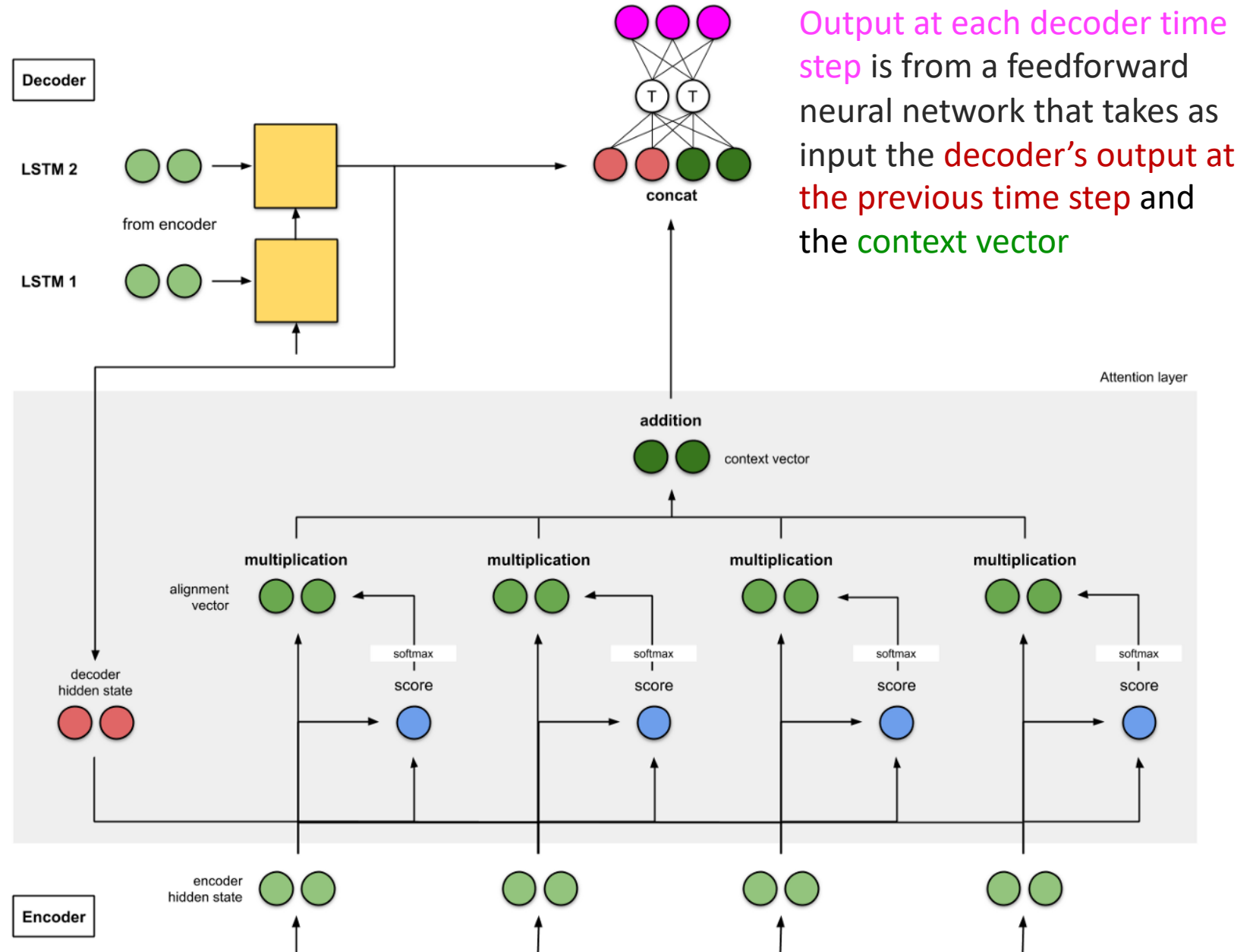
Many options exist for how to use the **context vector** with the **decoder's output at the previous time step** to produce an output at each decoder time step

Decoder predicts not only using **its output at the previous time step** and **previous hidden state**, but also with a **context vector**



Luong method

Many options exist for how to use the **context vector** with the **decoder's output at the previous time step** to produce an output at each decoder time step



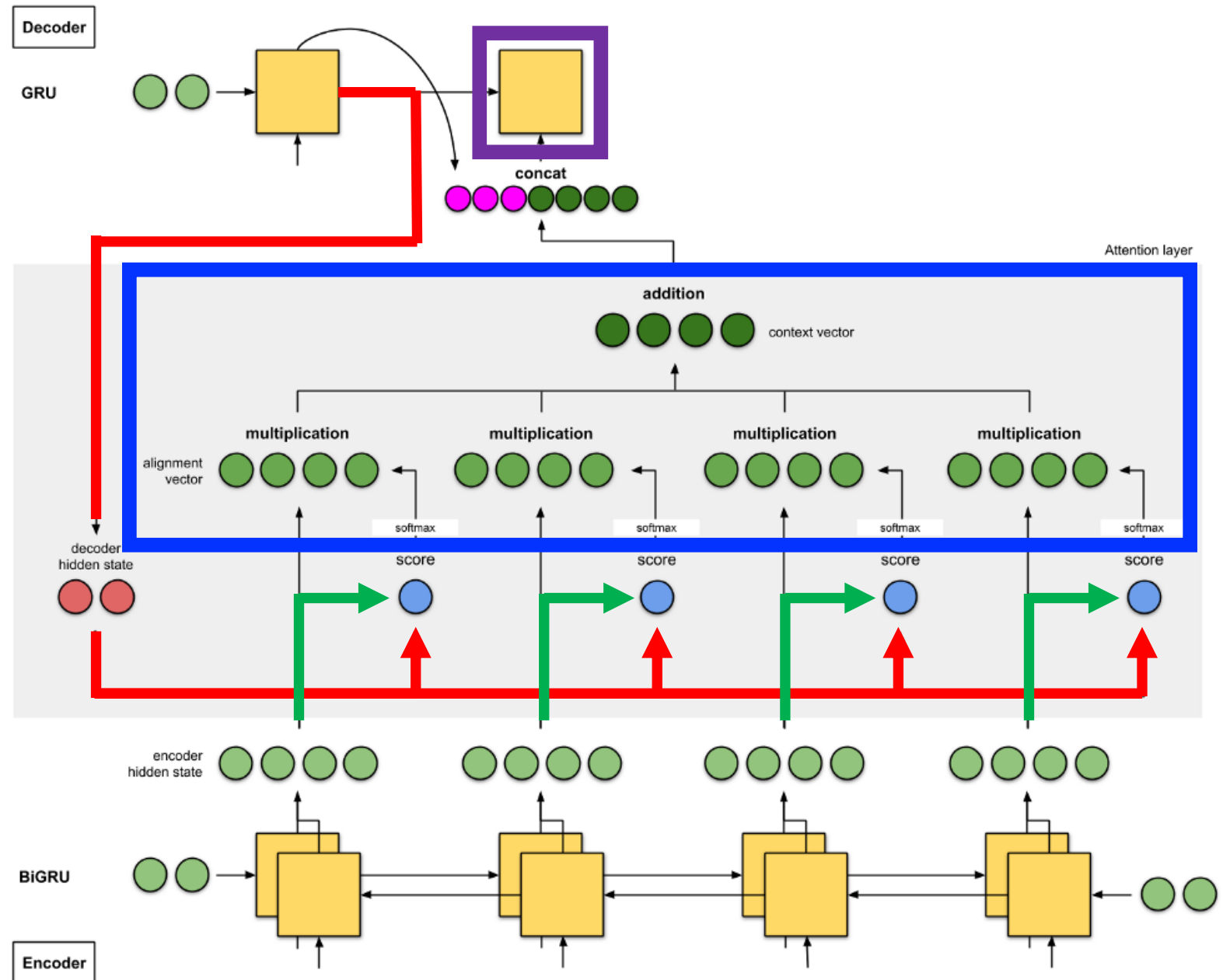
Decoder

What stays the same at each decoder time step?

- input's hidden state

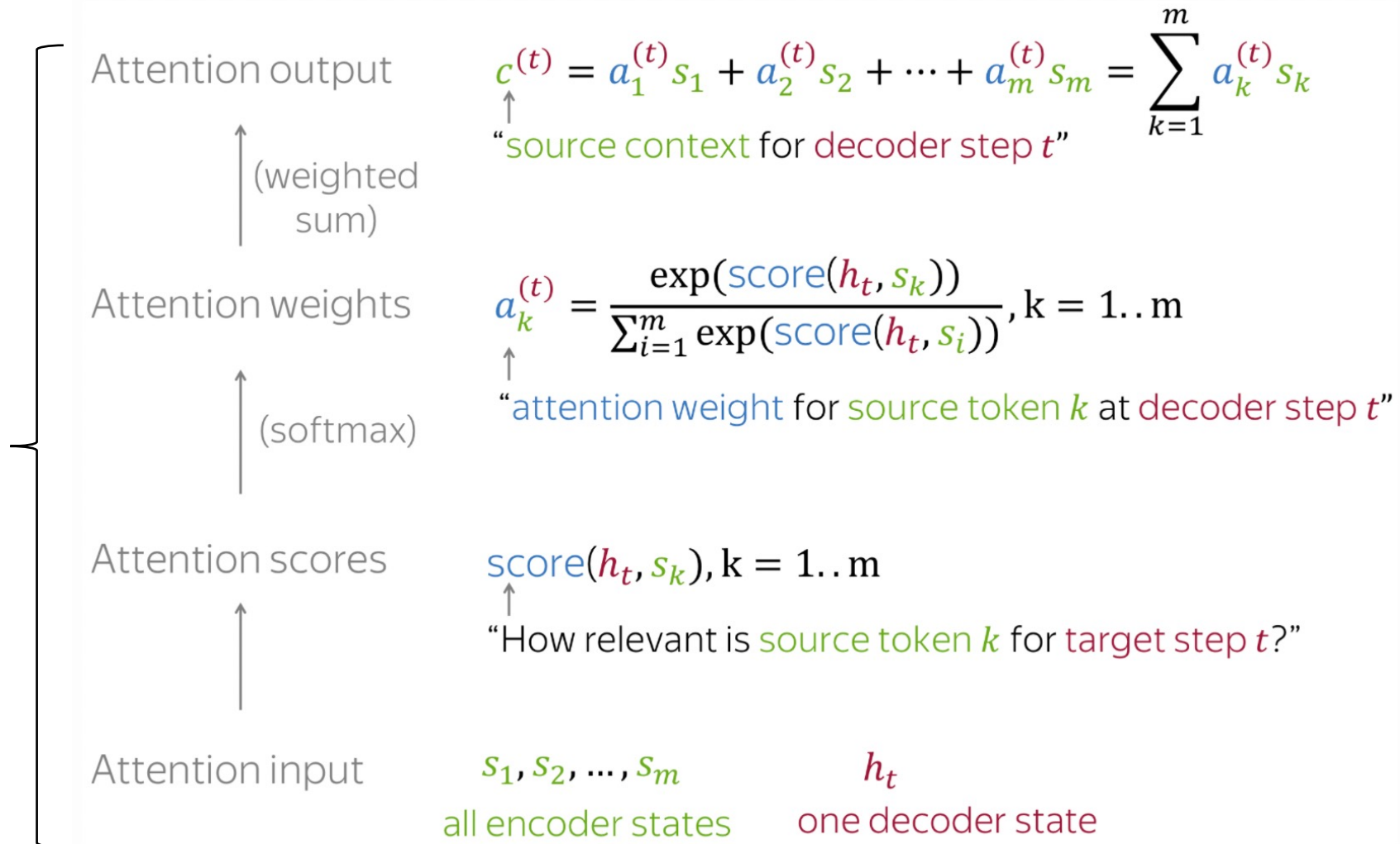
What changes at each decoder time step?

- decoder's hidden state
- (and so) attention weights and context vector
- decoder's output word at the previous time step



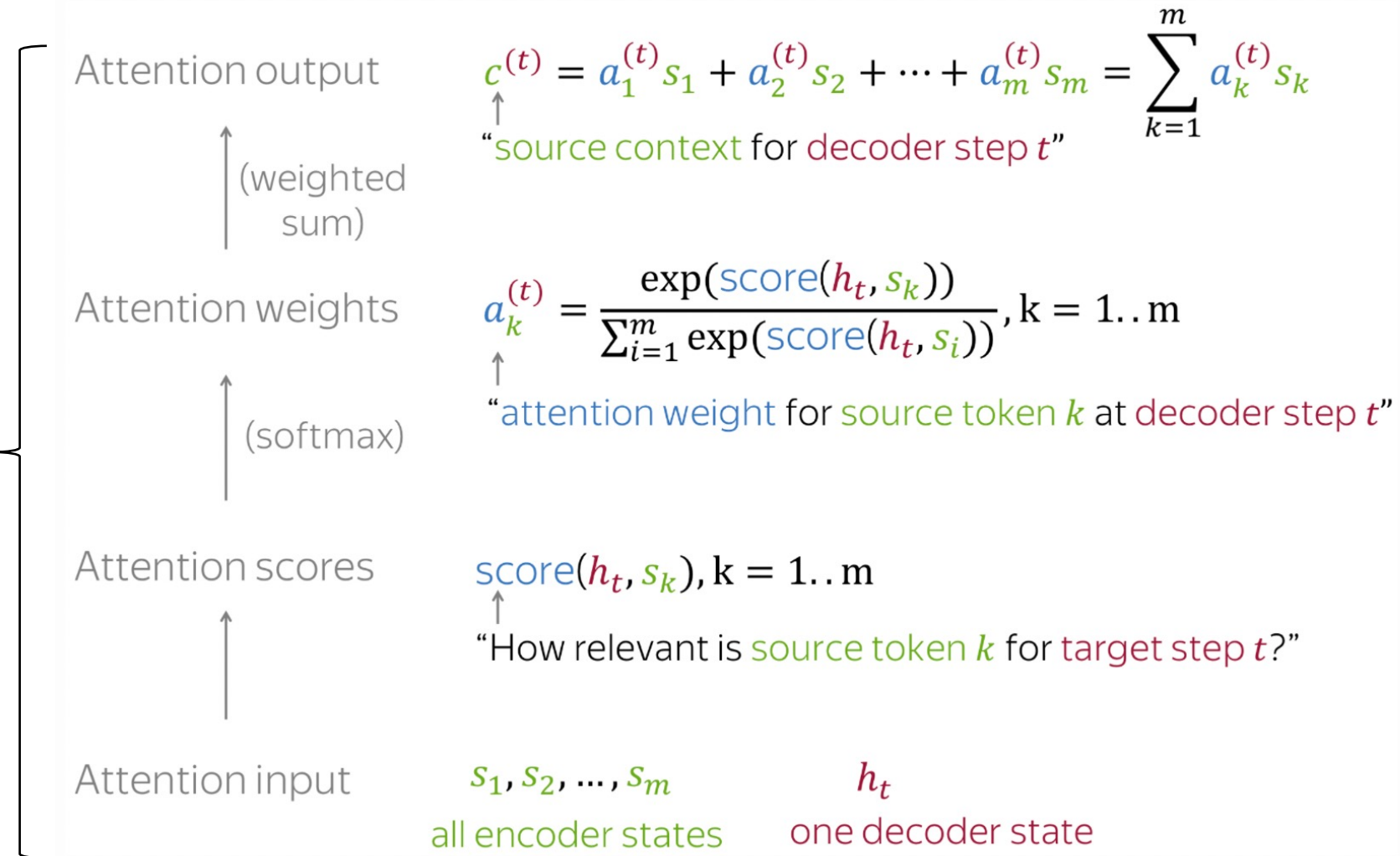
Summary: Attention (Computations at Each Decoder Step)

Decoder decides which inputs are needed for prediction at each time step with “soft attention”, which results in a weighted combination of the input



Summary: Attention (Computations at Each Decoder Step)

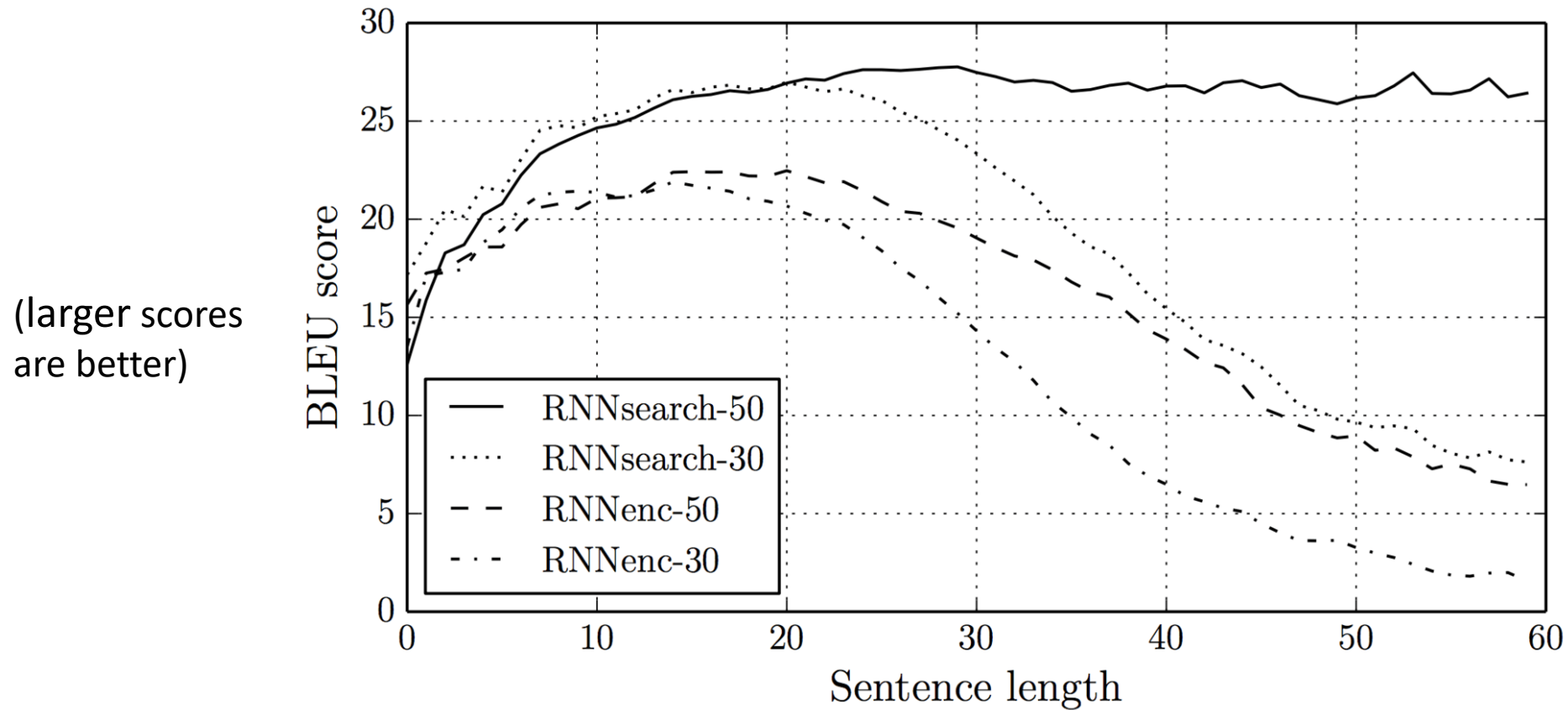
All parts are differentiable
which means end-to-end
training is possible



Today's Topics

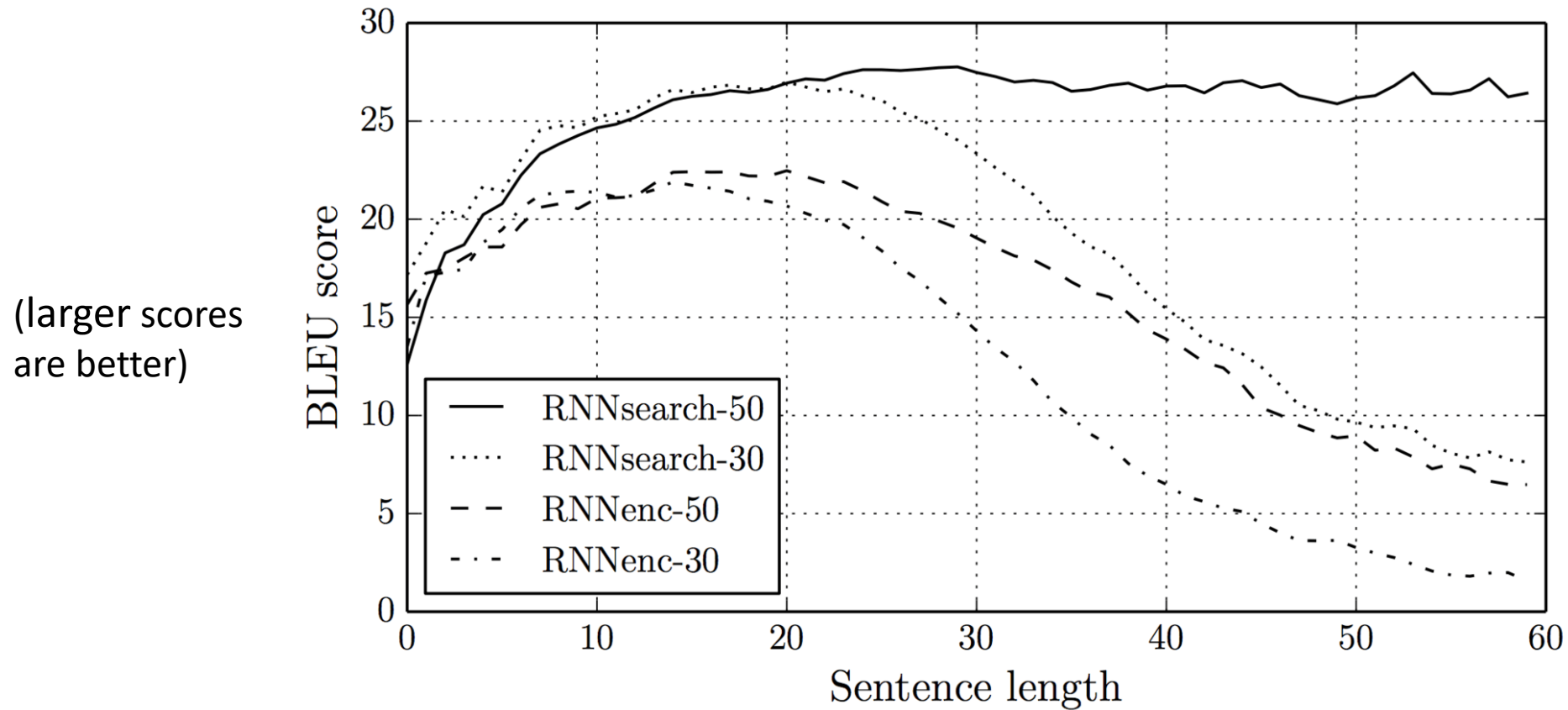
- Motivation: machine neural translation for long sentences
- Encoder
- Decoder: attention
- Performance evaluation

Analysis of Attention Models



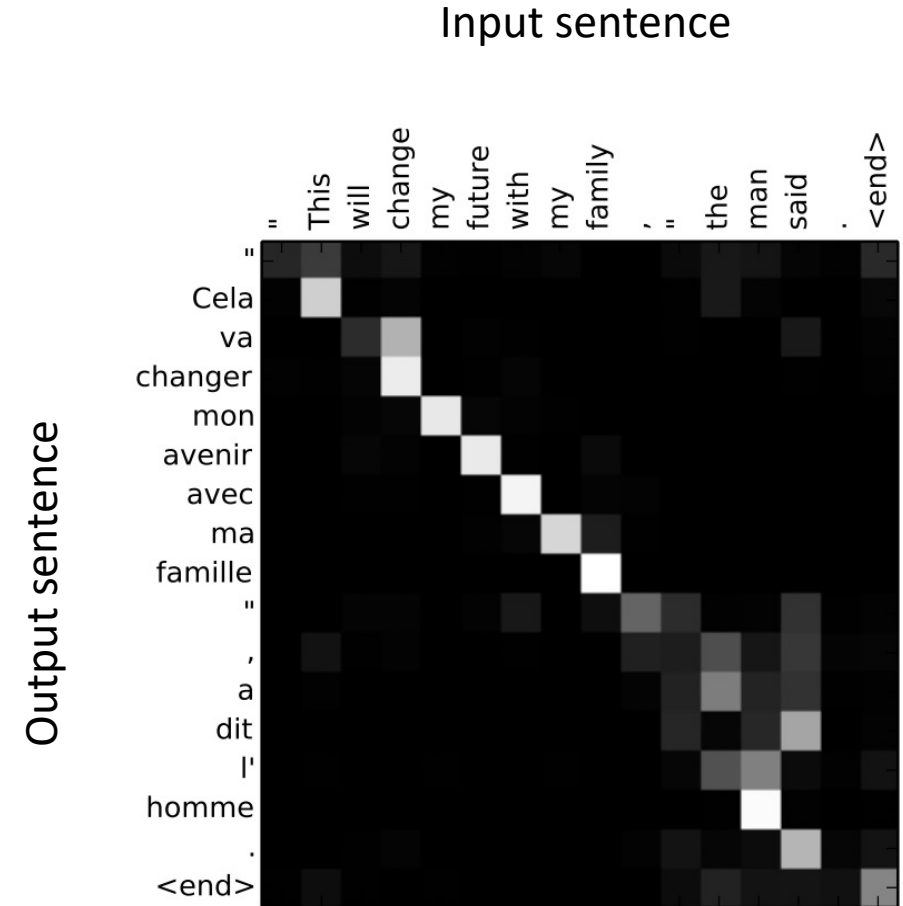
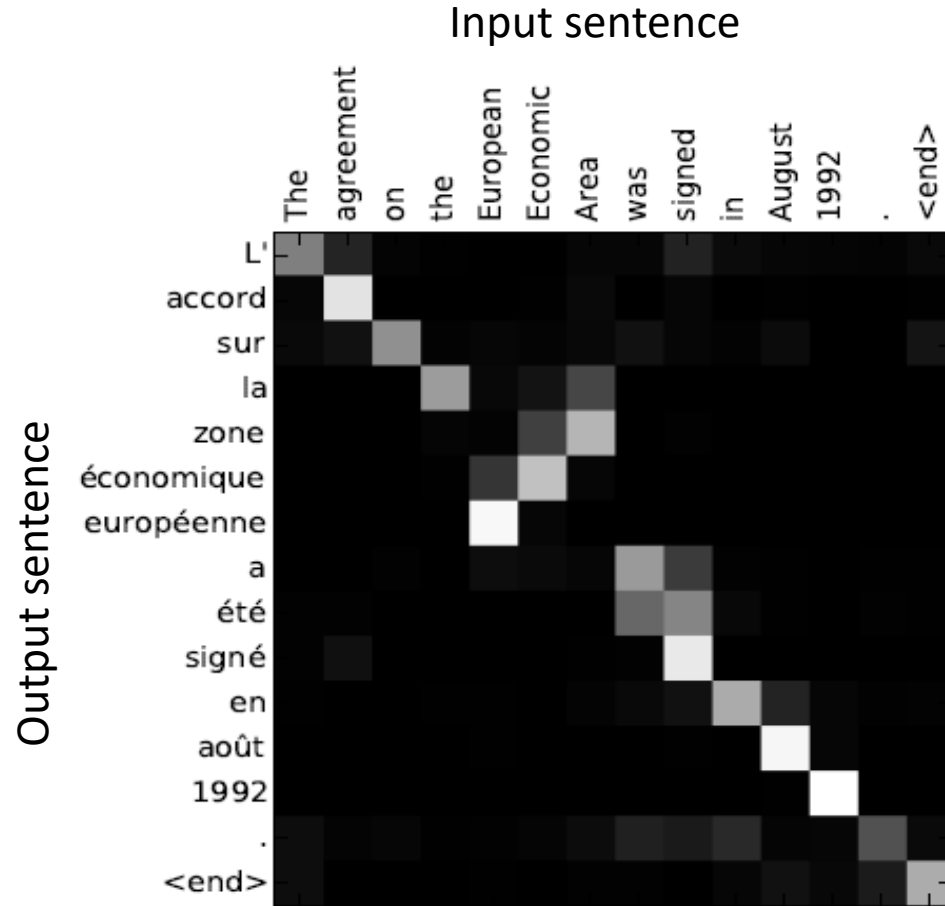
What performance trend is observed as the number of words in the input sentence grows?

Analysis of Attention Models



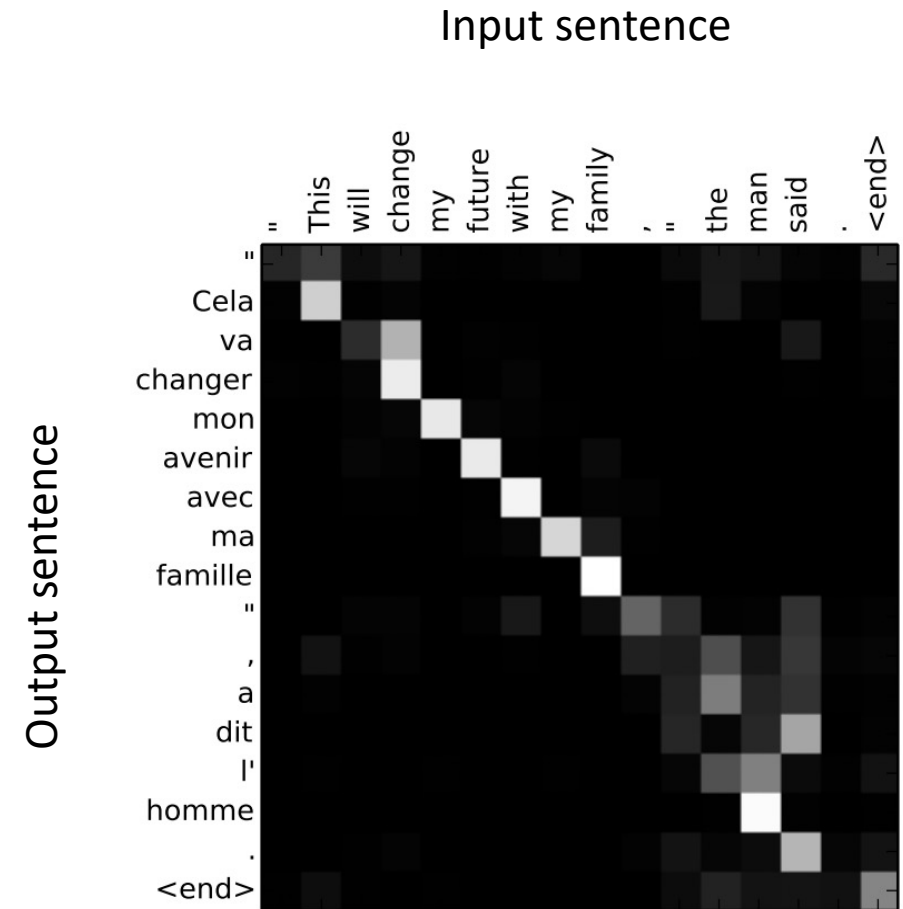
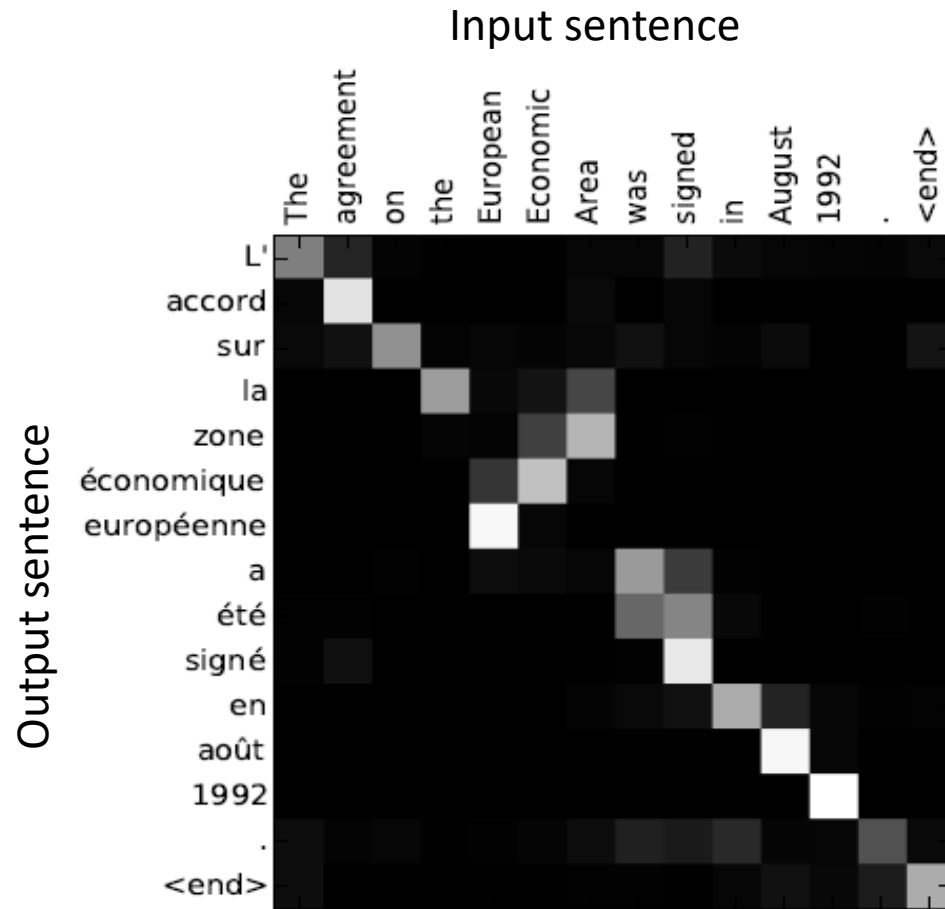
Performance no longer drops for longer sentences!

Visualizing Attention



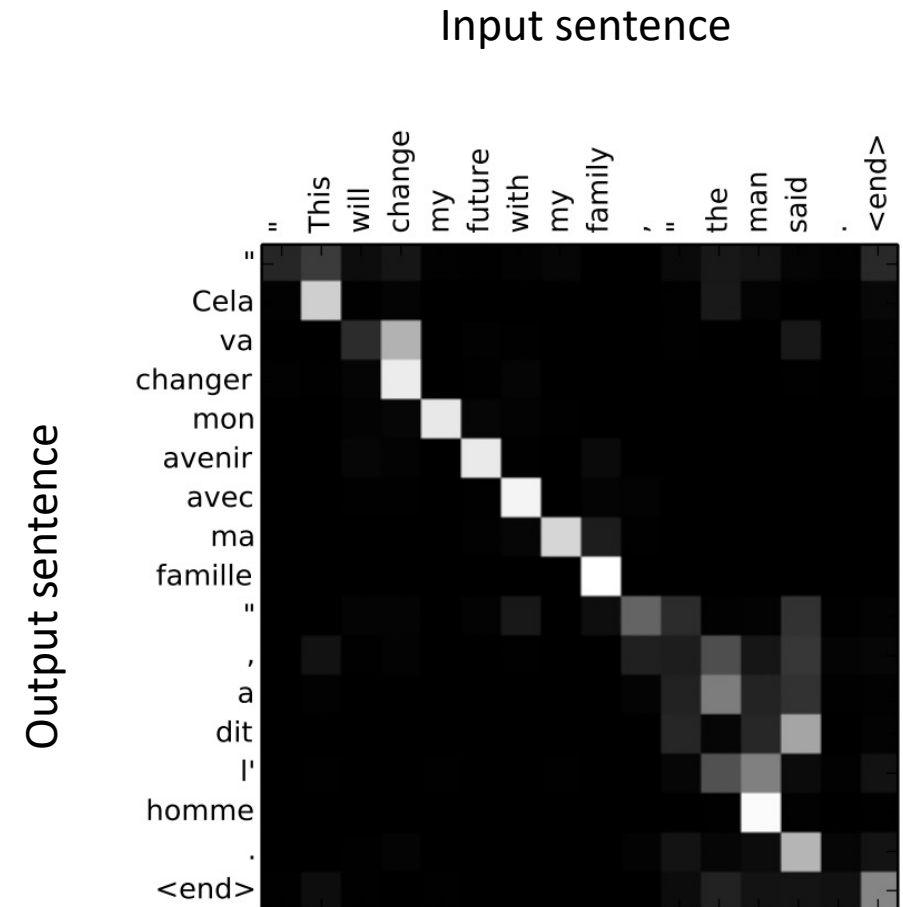
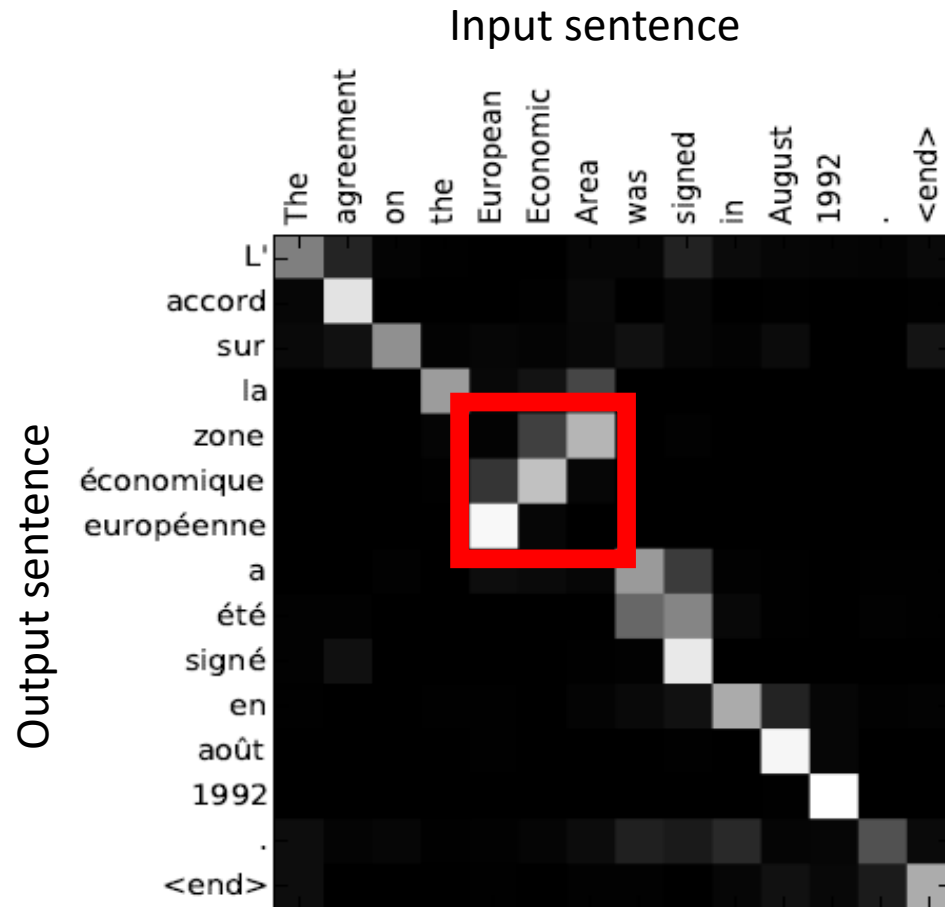
Values are 0 to 1, with whiter pixels indicating larger attention weights

Visualizing Attention



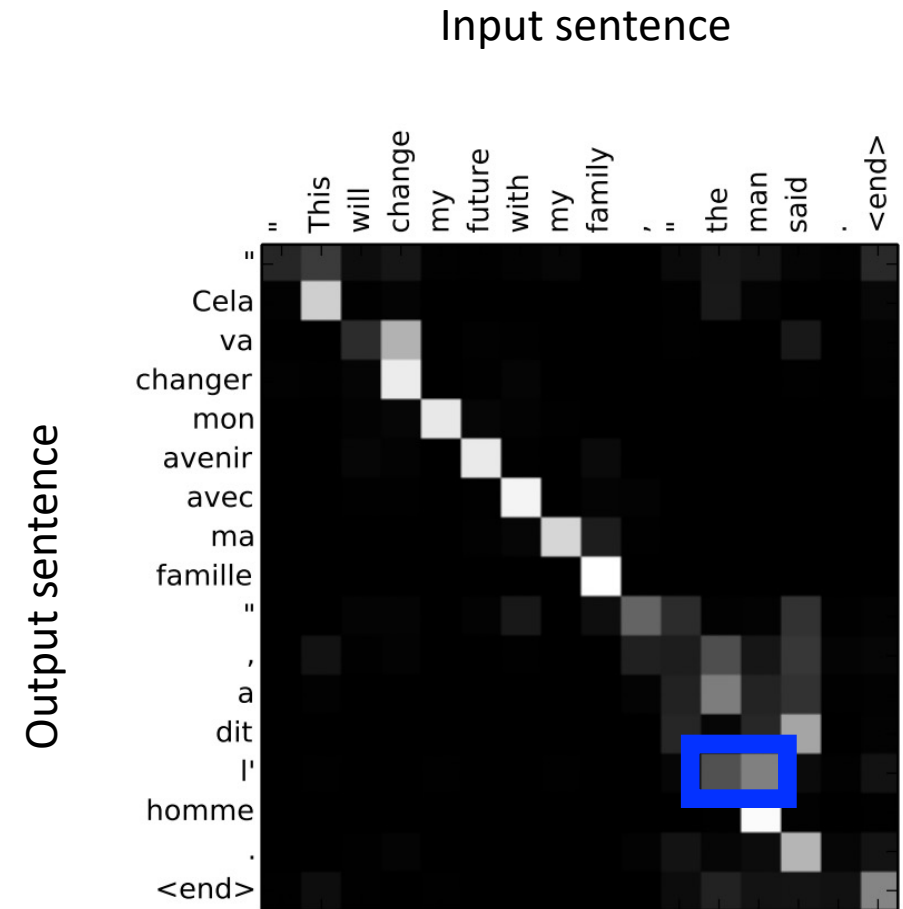
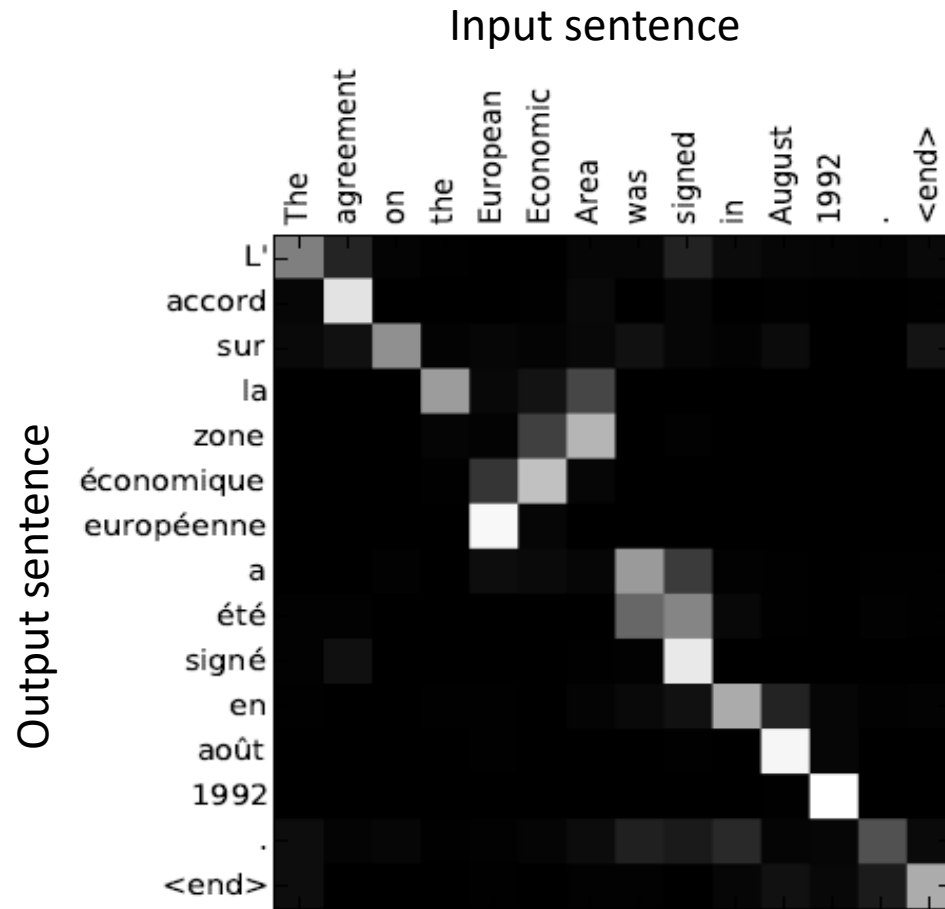
What insights can we glean from these examples?

Visualizing Attention



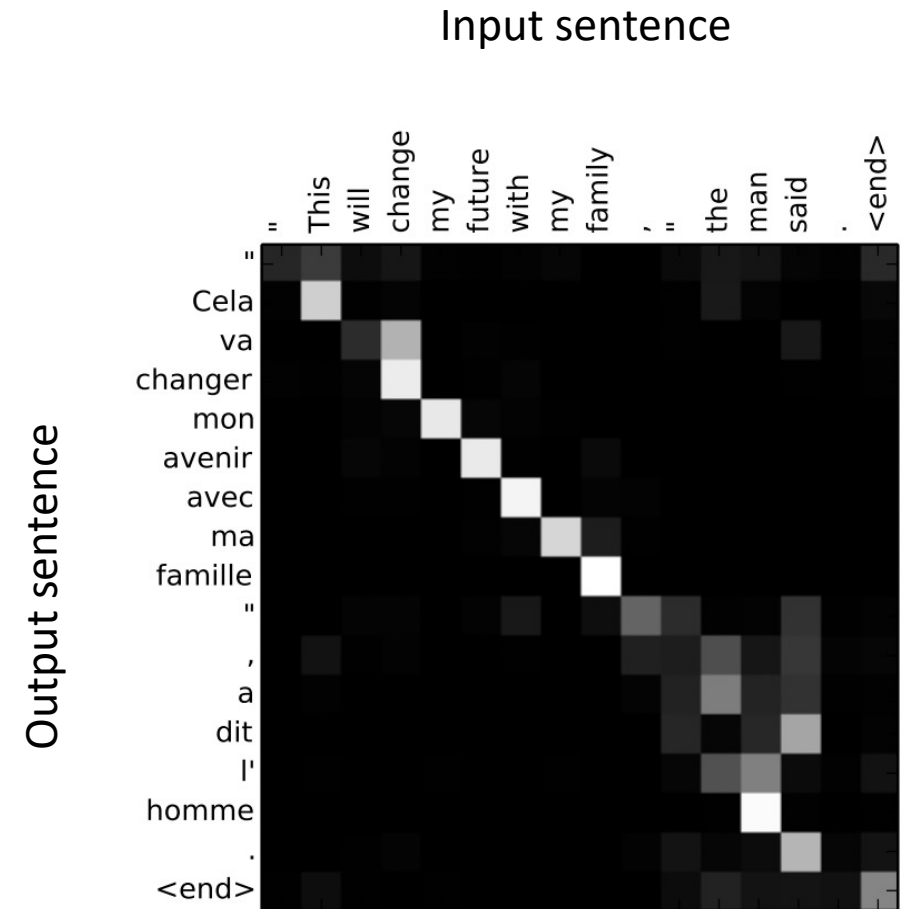
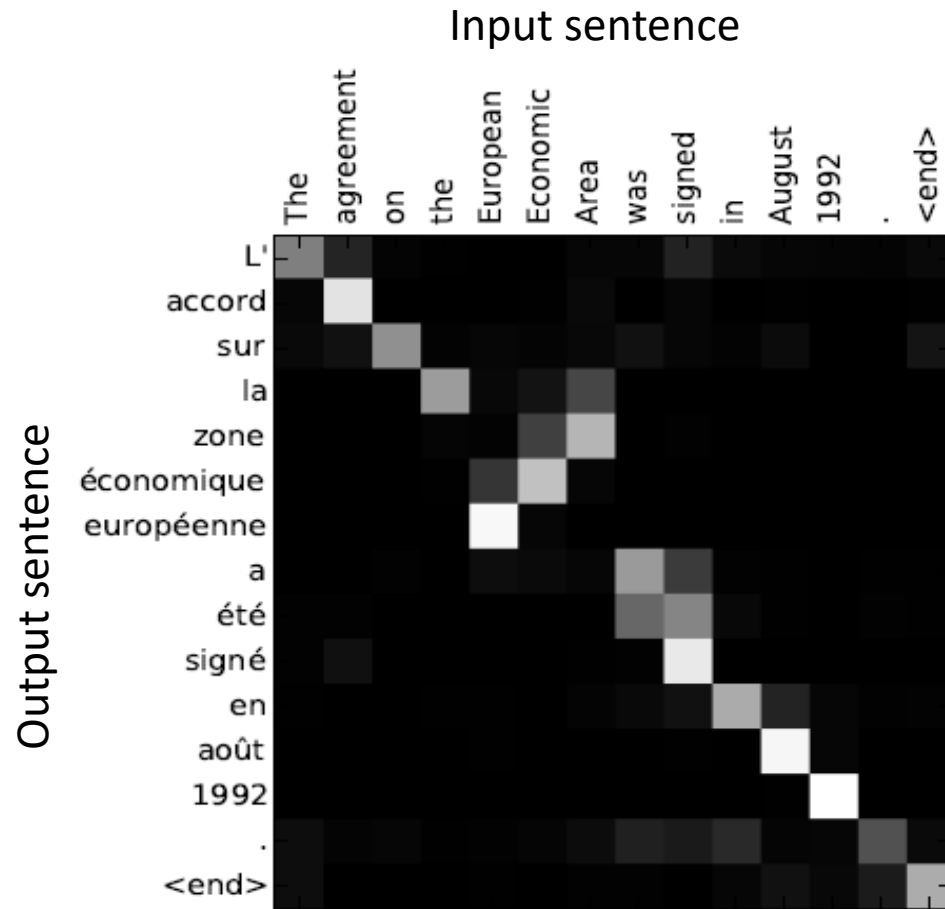
While a linear alignment between input and output sentences is common, there are exceptions (e.g., order of adjectives and nouns can differ)

Visualizing Attention



Output words are often informed by more than one input word;
e.g., “man” indicates translation of “the” to l’ instead of le, la, or les

Visualizing Attention



It naturally handles different input and output lengths
(e.g., 1 extra output word for both examples)

Today's Topics

- Motivation: machine neural translation for long sentences
- Encoder
- Decoder: attention
- Performance evaluation



The End