



EURO²

Mastering Transformers: From Building Blocks to Real-World Applications

Attention by Prof.Dr.Tuğba Taşkaya Temizel

12 Sept 2023

What is Attention?

- **Attention** is, to some extent, motivated by how we pay visual attention to different regions of an image or correlate words in one sentence.
 - It is a weight vector, which is dynamically calculated using deep networks.

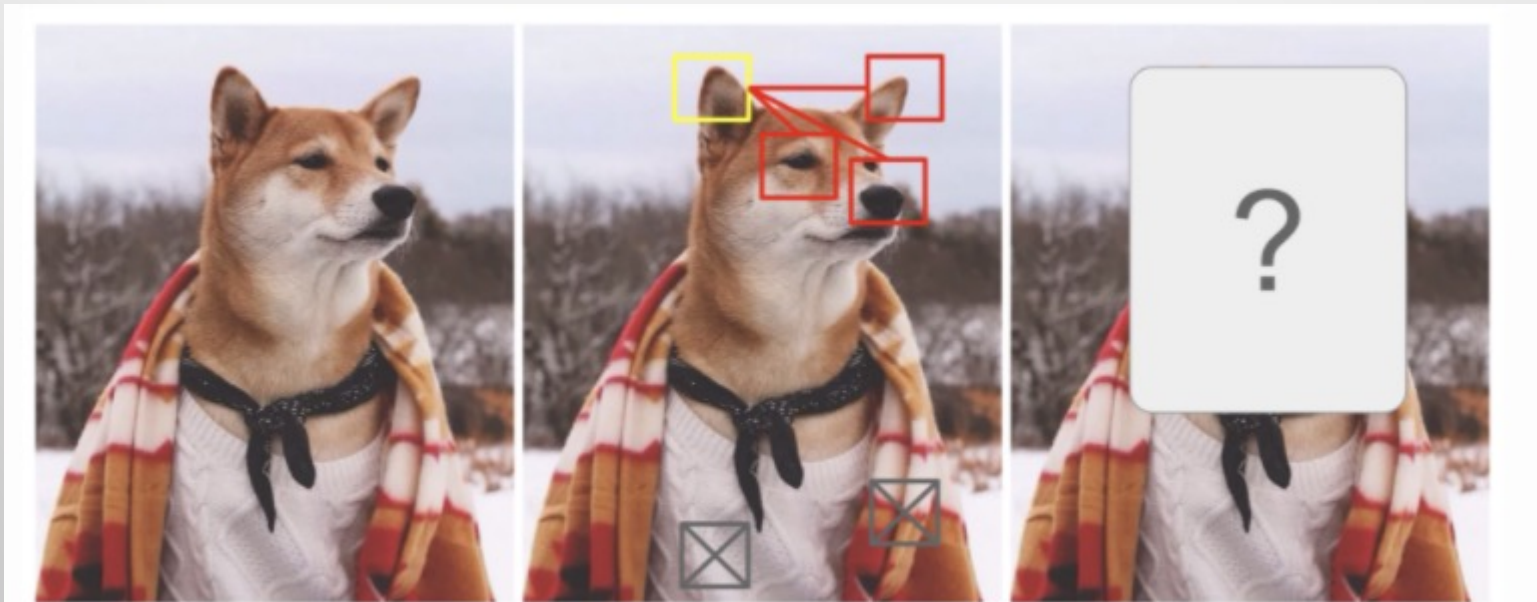


Fig. 1. A Shiba Inu in a men's outfit. The credit of the original photo goes to Instagram

@mensweardog.

What is Attention?

- Let's make an analogy with databases.

Word ID	Word
1	Dog
2	Cat
3	Rabbit

Key

Value

Query: Retrieve the Word ID=1

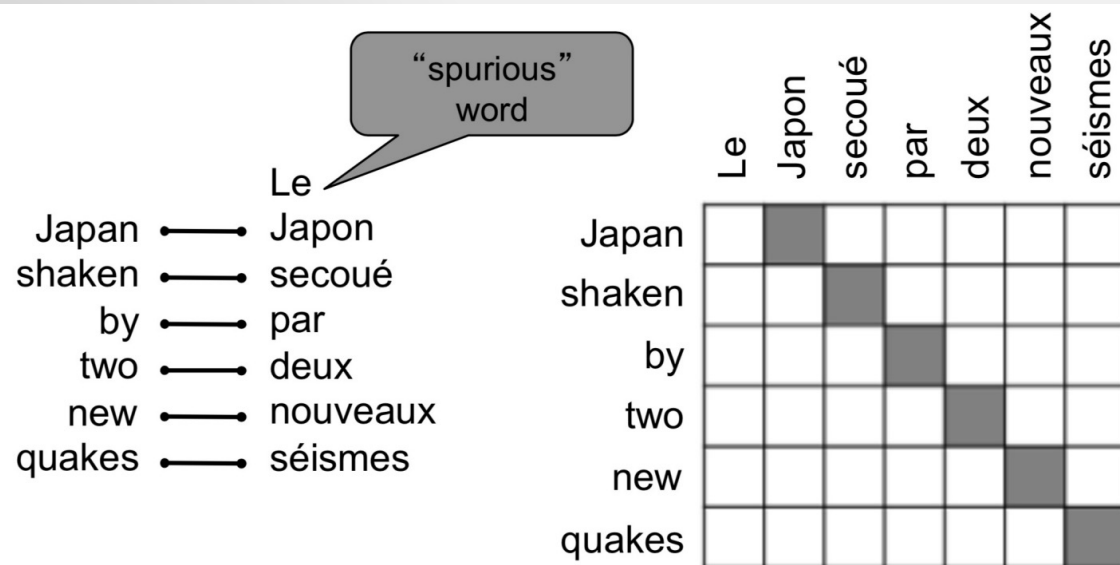
Query is aligned with the key in the database.
The matching key returns the value.

This is one-to-one alignment-> Query is matched with only one word id.

Background: Alignment

Example: Alignment for Machine Translation

- Let's think about machine translation task.
 - Machine translation** is the task of translating a sentence x in one language (source language) to another sentence y in another language (target language).
- Alignment is the **correspondence between particular words** in the translated sentence pair.
- Typological** differences between languages lead to complicated alignments!
- Note: Some words have **no counterpart**

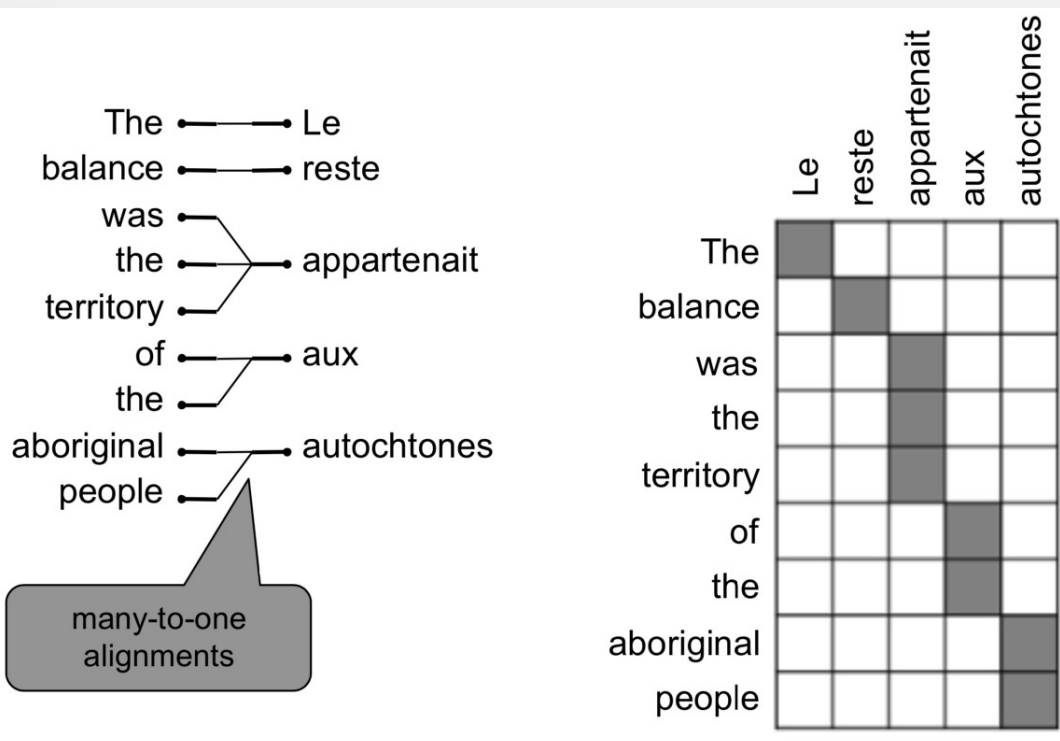


- Alignment can be **one-to-one**

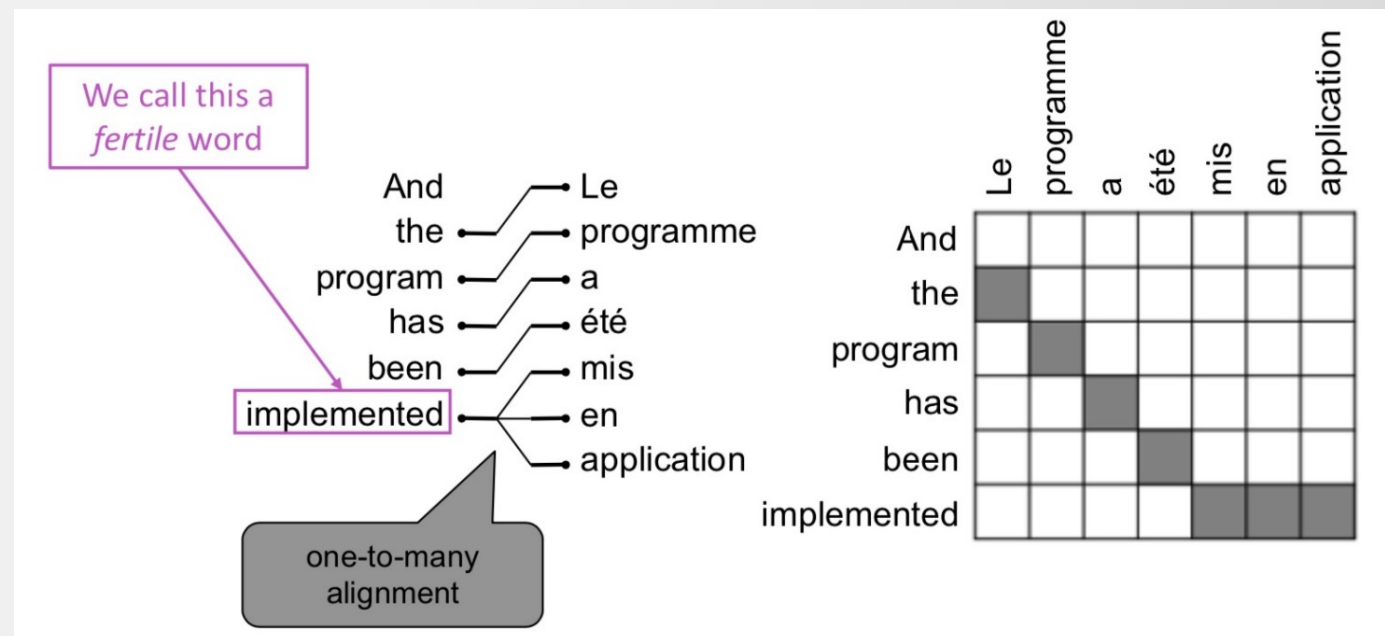
Background: Alignment

Example: Alignment for Machine Translation

- Alignment can be **many-to-one**



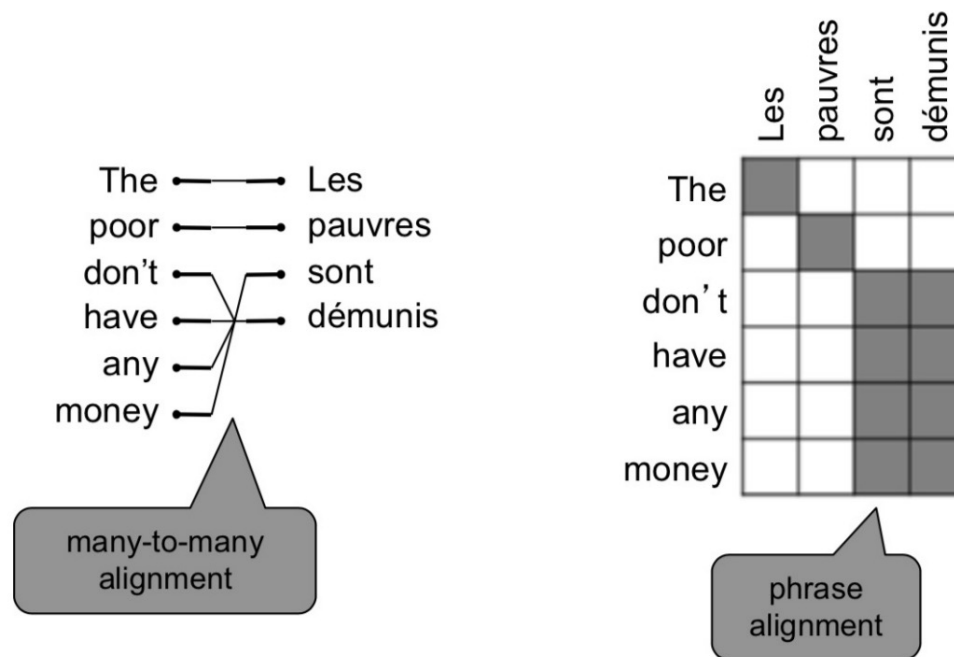
- Alignment can be **one-to-many**



Background: Alignment

Example: Alignment for Machine Translation

- Alignment can be **many-to-many (phrase-level)**



Examples from: "The Mathematics of Statistical Machine Translation: Parameter Estimation", Brown et al, 1993. <http://www.aclweb.org/anthology/I93-2003>

Background: Alignment

Example: Alignment for Machine Translation

- We learn as a combination of several factors, including:
 - Probability of particular words aligning (also depends on position in sent)
 - Probability of particular words having a particular fertility (number of corresponding words)
 - etc.
- Alignments a are latent variables: They aren't explicitly specified in the data!
 - Require the use of special learning algorithms (like Expectation-Maximization) for learning the parameters of distributions with latent variables
 - Earlier studies relied on statistical machine translation (SMT)

What is Attention?

- Attention can be implemented and calculated in numerous ways and architectures.
 - Attention is a general deep learning technique.
- Keys, values and queries vary based on the application task and model.
 - Given a set of vector **values**, and a vector **query**, **attention** is a technique to compute a weighted sum of the **values**, dependent on the **query**.
 - In other words, the output is the weighted sum of the **values**, where the weight of each value is given by the compatibility between **query** and **key**. The dot product provides a measure of compatibility.
- *Query attends to the values!*
- *Query determines which values to focus on!*
- *Keys, Values and Query are all vectors.*

What is Attention?

- Let's denote $\mathcal{D} \stackrel{\text{def}}{=} \{(\mathbf{k}_1, \mathbf{v}_1), \dots, (\mathbf{k}_m, \mathbf{v}_m)\}$ including m tuples of keys and values. Denote q as a query. We can define the attention over D as:

$$\text{Attention}(\mathbf{q}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i,$$

- where $\alpha(q, k_i) \in \mathbb{R}$ ($i = 1, \dots, m$) are scalar weights. This operation is also called as **attention pooling**.
 - If one of the weights $\alpha(q, k_i)$ is 1 and others are zero, it corresponds to a traditional database query.
 - All weights are equal, i.e. $\alpha(q, k_i) = \frac{1}{m}$ for all i . This amounts to averaging across the entire database, also called **average pooling** in deep learning.
 - The weights $\alpha(q, k_i)$ form a convex combination, i.e. $\sum_i \alpha(q, k_i) = 1$ and $\alpha(q, k_i) \geq 0$ for all i . This is the common setting in deep learning.

Attention in Seq2Seq Models

Background

Encoder-Decoder Model

- Encoder-decoder networks, or sequence-to-sequence networks, are models capable of generating contextually appropriate, arbitrary length, output sequences.

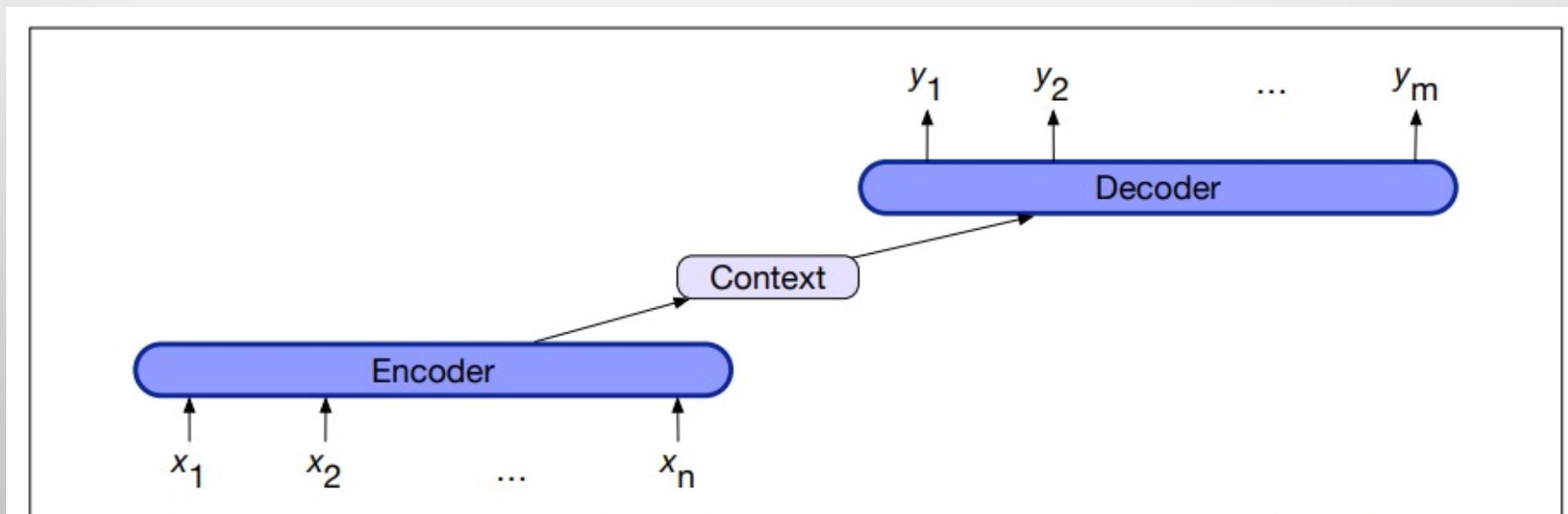
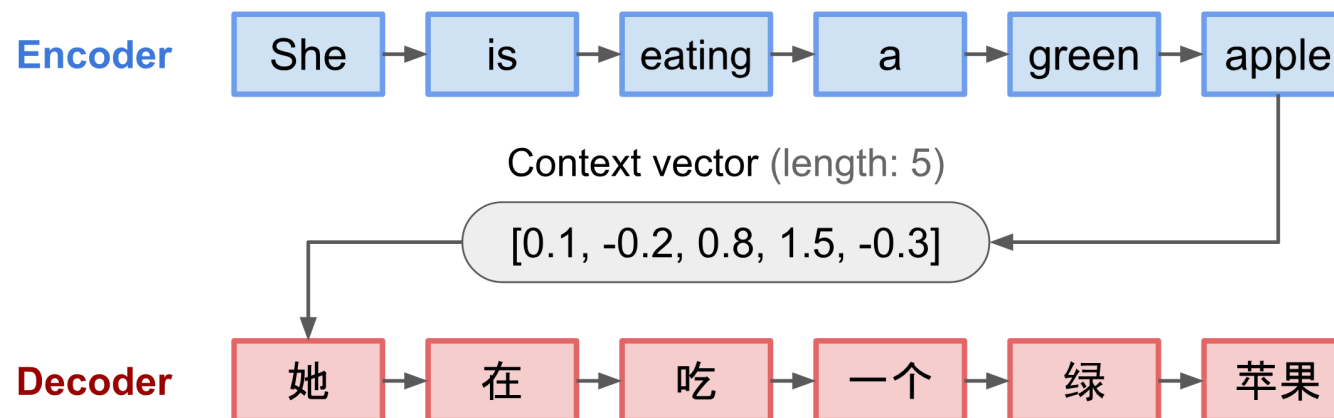


Figure 11.3 The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

Background

Encoder-Decoder Model

- Encoder-decoder networks consist of three components:
 1. An **encoder** that accepts an input sequence, x_1^n , and generates a corresponding sequence of contextualized representations, h_1^n .
 - LSTMs, GRUs, convolutional networks, and Transformers can all be employed as encoders.
 2. A **context** vector, c , which is a function of h_1^n , and conveys the essence of the input to the decoder.



Background

Encoder-Decoder Model

- Encoder-decoder networks consist of three components:
 3. A **decoder**, which accepts c as input and generates an arbitrary length sequence of hidden states h_1^m , from which a corresponding sequence of output states y_1^m , can be obtained.
 - Just as with encoders, decoders can be realized by any kind of sequence architecture.

What's wrong with Seq2Seq Model?

- The final hidden state is acting as a bottleneck: it must represent absolutely everything about the meaning of the source text, since the only thing the decoder knows about the source text is what's in this context vector.

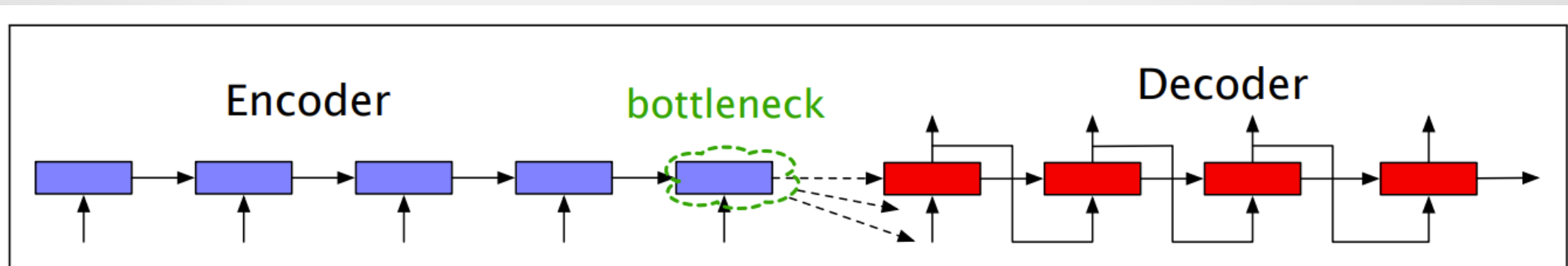


Figure 11.8 Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

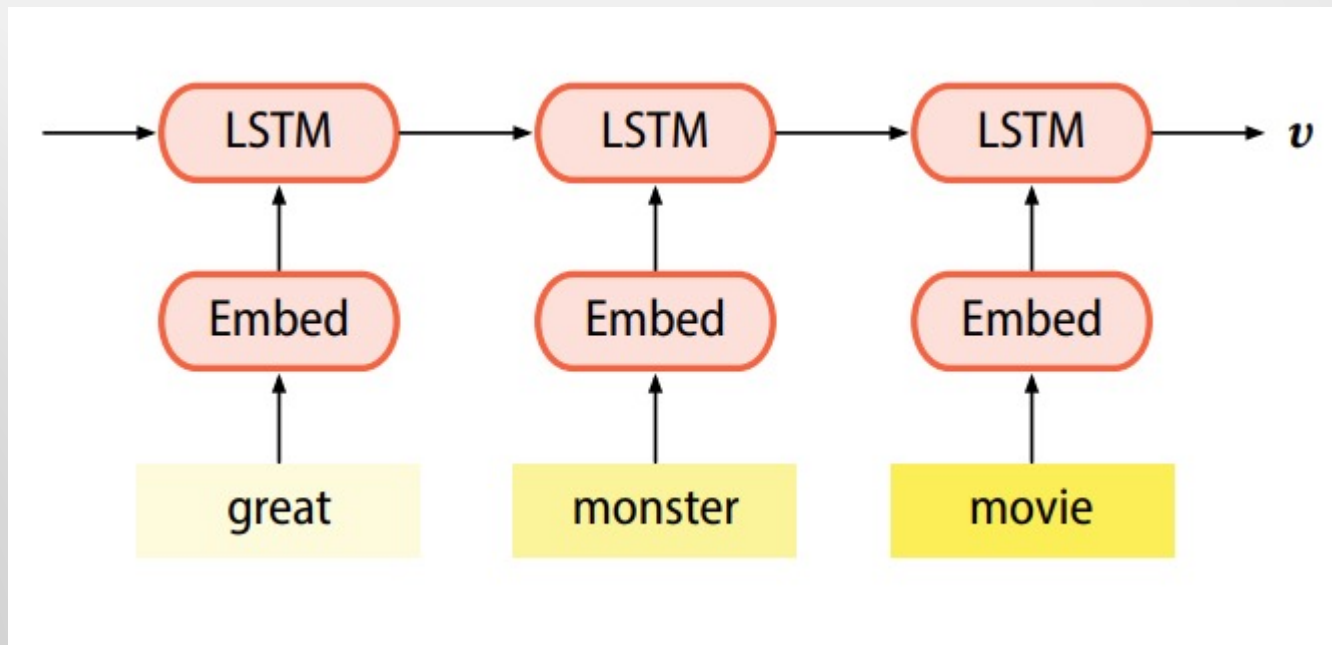
Encoder: Builds up sentence meaning

What's wrong with Seq2Seq Model?

- As the input sequence gets compressed and blended into a single dense vector, sufficiently large memory capacity is required to store past information.
 - As a result, the network generalizes poorly to long sequences while wasting memory on shorter ones.

What's wrong with Seq2Seq Model?

- **Recency bias:** Information at the beginning of the sentence, especially for long sentences, may not be equally well represented in the context vector.
- The last hidden state is prone to bias towards the recent past.



Attention in Seq2Seq Model

- The **attention mechanism** is a solution to the bottleneck problem, a way of allowing the decoder to get information from all the hidden states of the encoder, not just the last hidden state.
- Here, the context vector c is a single vector that is a function of the hidden states of the encoder, that is, $c = f(h_1^n)$.
 - Because the number of hidden states varies with the size of the input, we can't use the entire tensor of encoder hidden state vectors directly as the context for the decoder.
 - The idea of attention is instead to create the single fixed-length vector c by taking a weighted sum of all the encoder hidden states h_1^n .

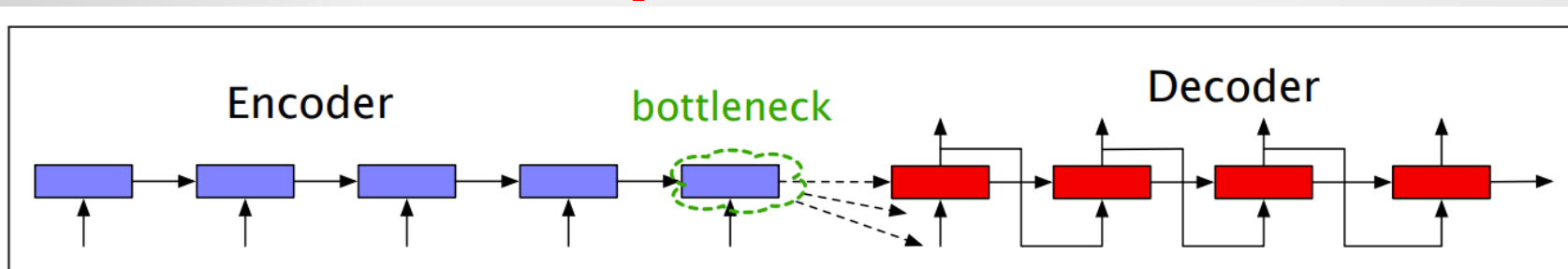


Figure 11.8 Requiring the context c to be only the encoder's final hidden state forces all the information from the entire source sentence to pass through this representational bottleneck.

Attention in Seq2Seq Model

- Attention replaces the static context vector with one that is dynamically derived from the encoder hidden states at each point during decoding.
- This context vector, c_i , is generated anew with each decoding step i and takes all the encoder hidden states into account in its derivation.

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

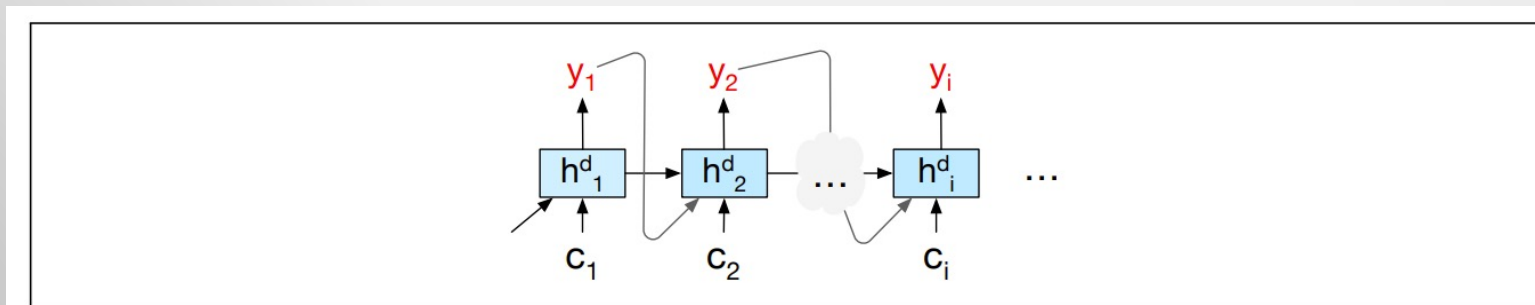


Figure 11.9 The attention mechanism allows each hidden state of the decoder to see a different, dynamic, context, which is a function of all the encoder hidden states.

Attention in Seq2Seq Model

- Let's calculate how relevant each encoder state is to the decoder state captured in h_{i-1}^d
- We capture **relevance** by computing— at each state i during decoding—a $\text{score}(h_{i-1}^d, h_j^e)$ for each encoder state j .

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

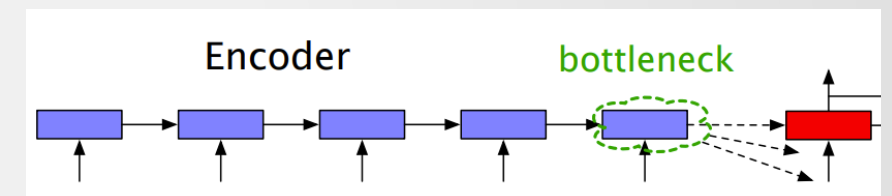
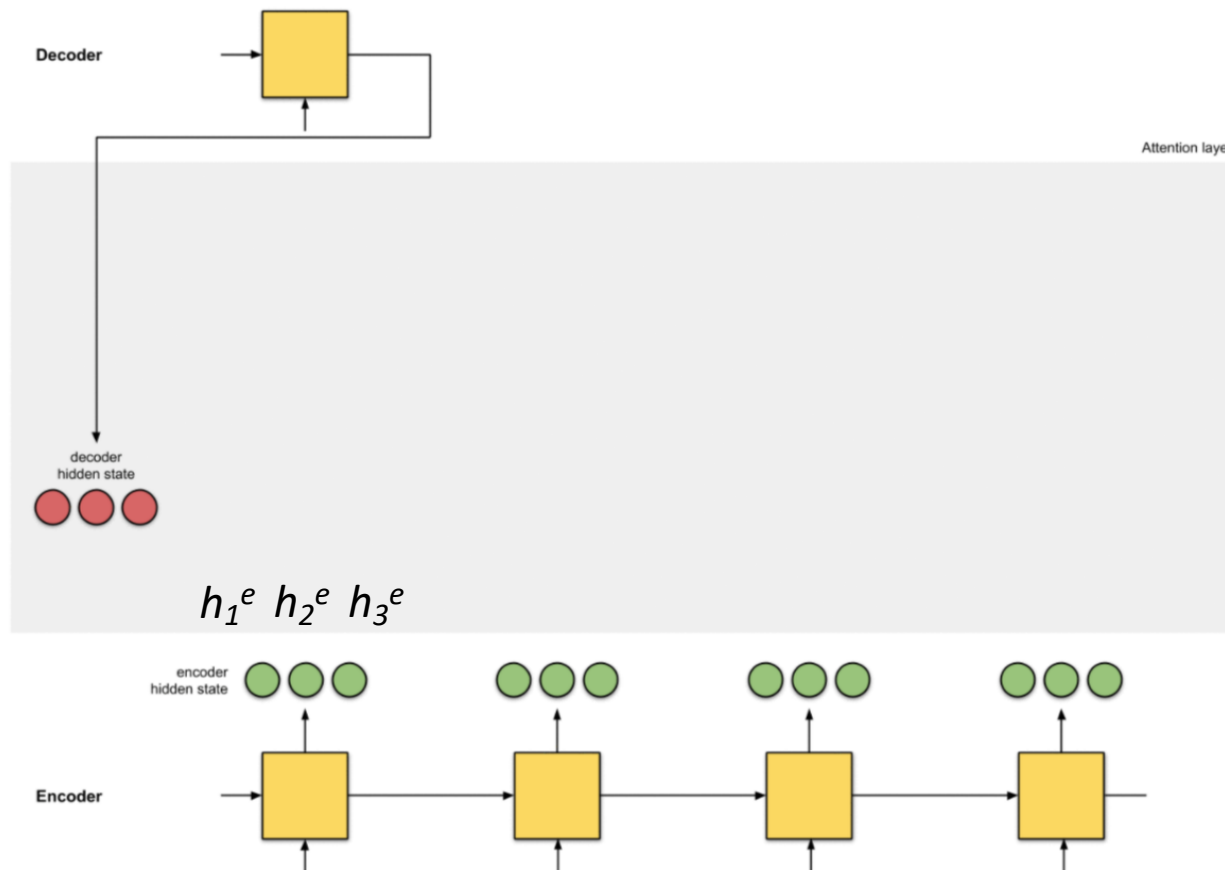
Attention in Seq2Seq Model

A Toy Example

- The vector of these scores across all the encoder hidden states gives us the relevance of each encoder state to the current step of the decoder.

In this example, we have 4 encoder hidden states and the current decoder hidden state ($d=4$).

Note that the last consolidated encoder hidden state is fed as input to the first time step of the decoder.



Encoder last state.

decoder_hidden = [10, 5, 10] → Query

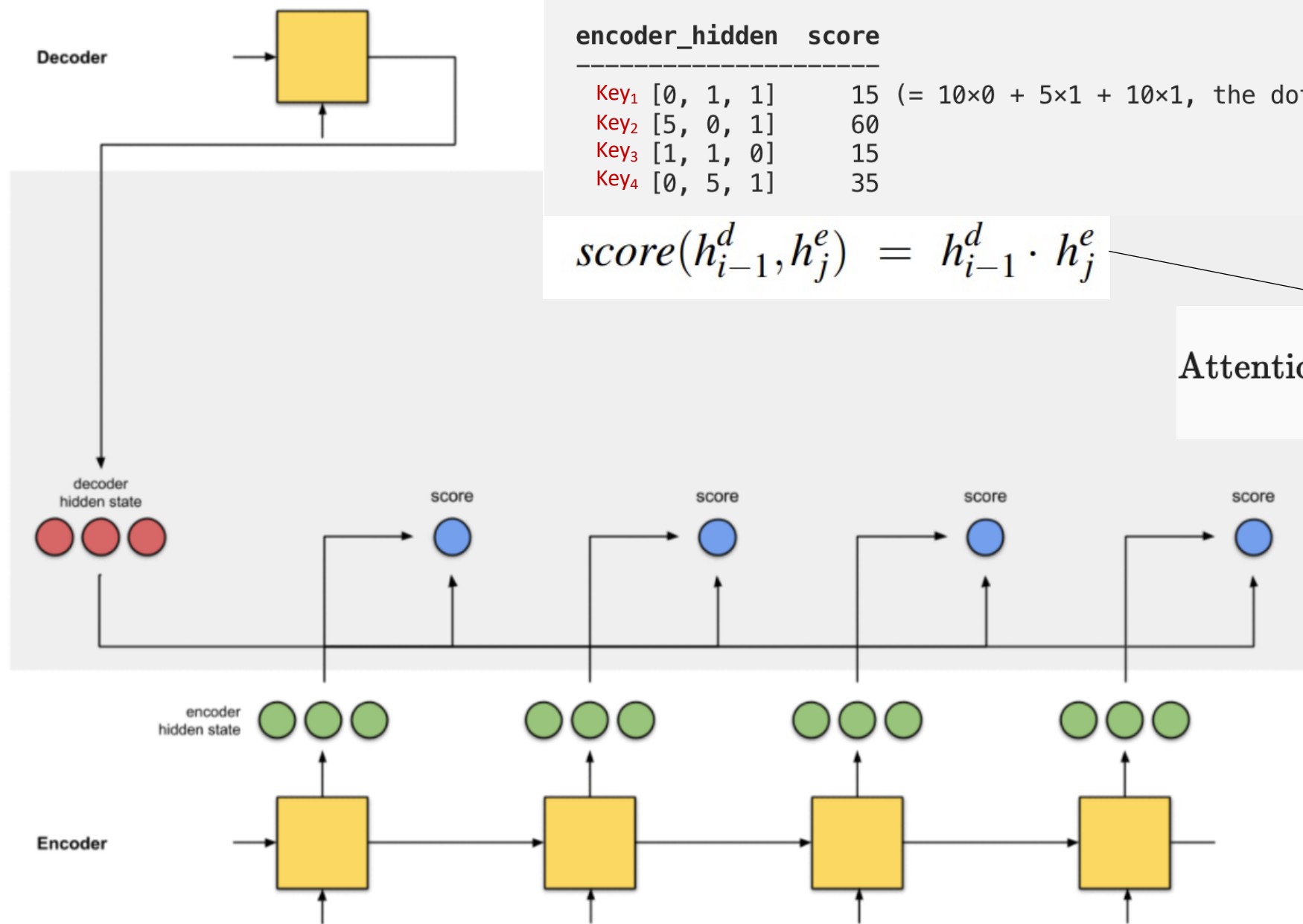
encoder_hidden score

Key ₁	[0, 1, 1]	15	(= 10×0 + 5×1 + 10×1, the dot product)
Key ₂	[5, 0, 1]	60	
Key ₃	[1, 1, 0]	15	
Key ₄	[0, 5, 1]	35	

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

We work on $\alpha(q, k_i)$

$$\text{Attention}(\mathbf{q}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i,$$



decoder_hidden = [10, 5, 10] \rightarrow Query

encoder_hidden score

Key ₁	[0, 1, 1]	15	(= 10×0 + 5×1 + 10×1, the dot product)
Key ₂	[5, 0, 1]	60	
Key ₃	[1, 1, 0]	15	
Key ₄	[0, 5, 1]	35	

$$\text{score}(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

The higher the score means the more relevant the hidden states of encoder and decoder hidden states are.

We work on $\alpha(q, k_i)$

$$\text{Attention}(\mathbf{q}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i,$$

Calculate the dot product between the query and the key:
The higher the similarity, the higher the dot products between vectors is.

Motivation: To identify the corresponding words in the queries that are similar to the keys.

Attention in Seq2Seq Model

- We'll normalize them with a softmax to create a vector of weights, α_{ij} , that tells us the proportional relevance of each encoder hidden state j to the prior hidden decoder state, h_{i-1}^d

$$\begin{aligned}\alpha_{ij} &= \text{softmax}(\text{score}(h_{i-1}^d, h_j^e) \quad \forall j \in e) \\ &= \frac{\exp(\text{score}(h_{i-1}^d, h_j^e))}{\sum_k \exp(\text{score}(h_{i-1}^d, h_k^e))}\end{aligned}$$

Attention in Seq2Seq Model

- Finally, given the distribution in α , we can compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states.

Recall:

$$\text{Attention}(\mathbf{q}, \mathcal{D}) \stackrel{\text{def}}{=} \sum_{i=1}^m \alpha(\mathbf{q}, \mathbf{k}_i) \mathbf{v}_i,$$

$$c_i = \sum_j \alpha_{ij} h_j^e$$

Corresponds to the **Value**

encoder_hidden	score	score^
[0, 1, 1]	15	0
[5, 0, 1]	60	1
[1, 1, 0]	15	0
[0, 5, 1]	35	0

attention layer

$$\alpha_{ij} = \text{softmax}(\text{score}(h_{i-1}^d, h_j^e) \quad \forall j \in e)$$

$$= \frac{\exp(\text{score}(h_{i-1}^d, h_j^e))}{\sum_k \exp(\text{score}(h_{i-1}^d, h_k^e))}$$

The final scores after softmax here were rounded and simplified to help you follow the calculations better.

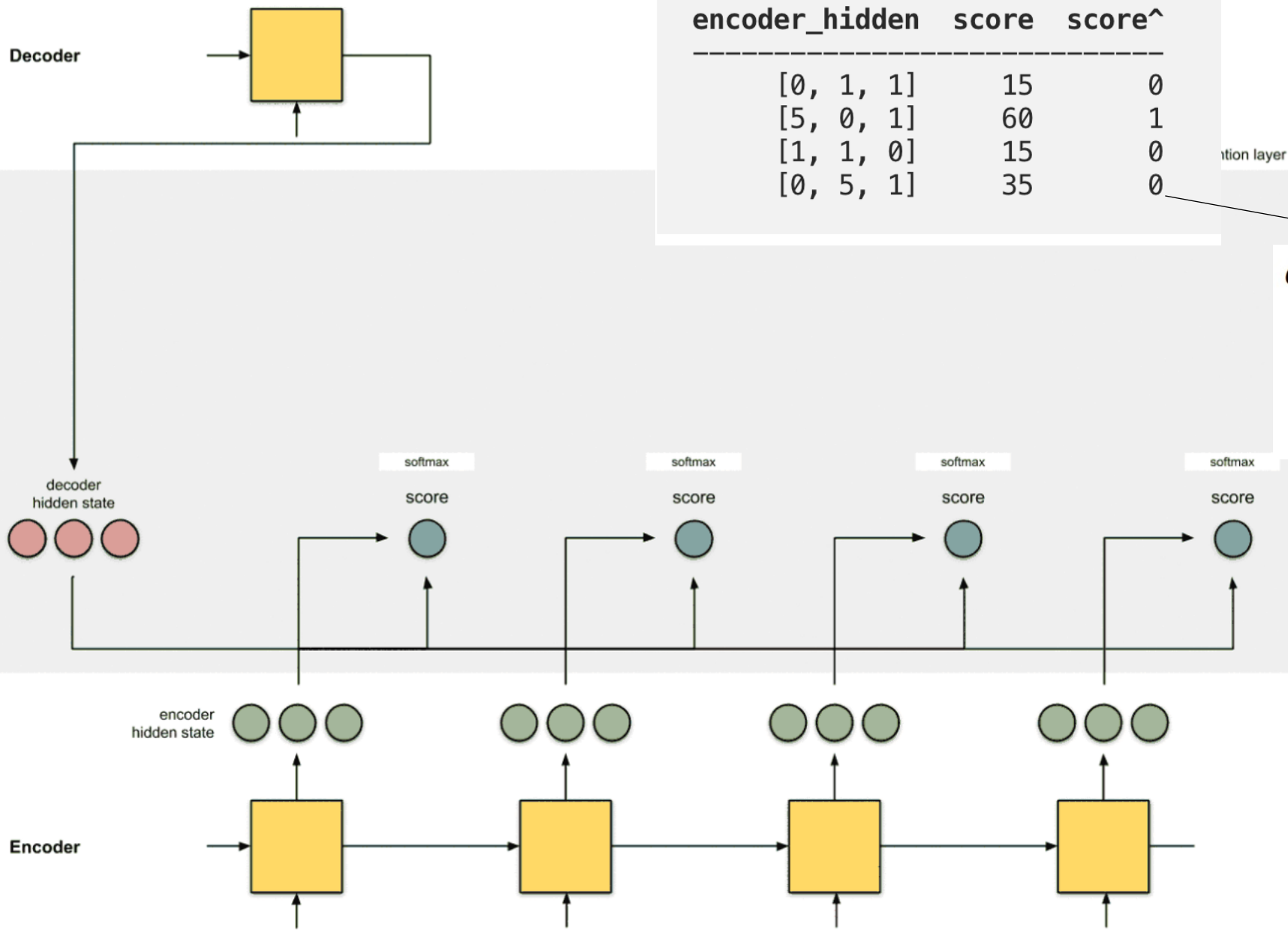
Total= 15+60+15+35=125

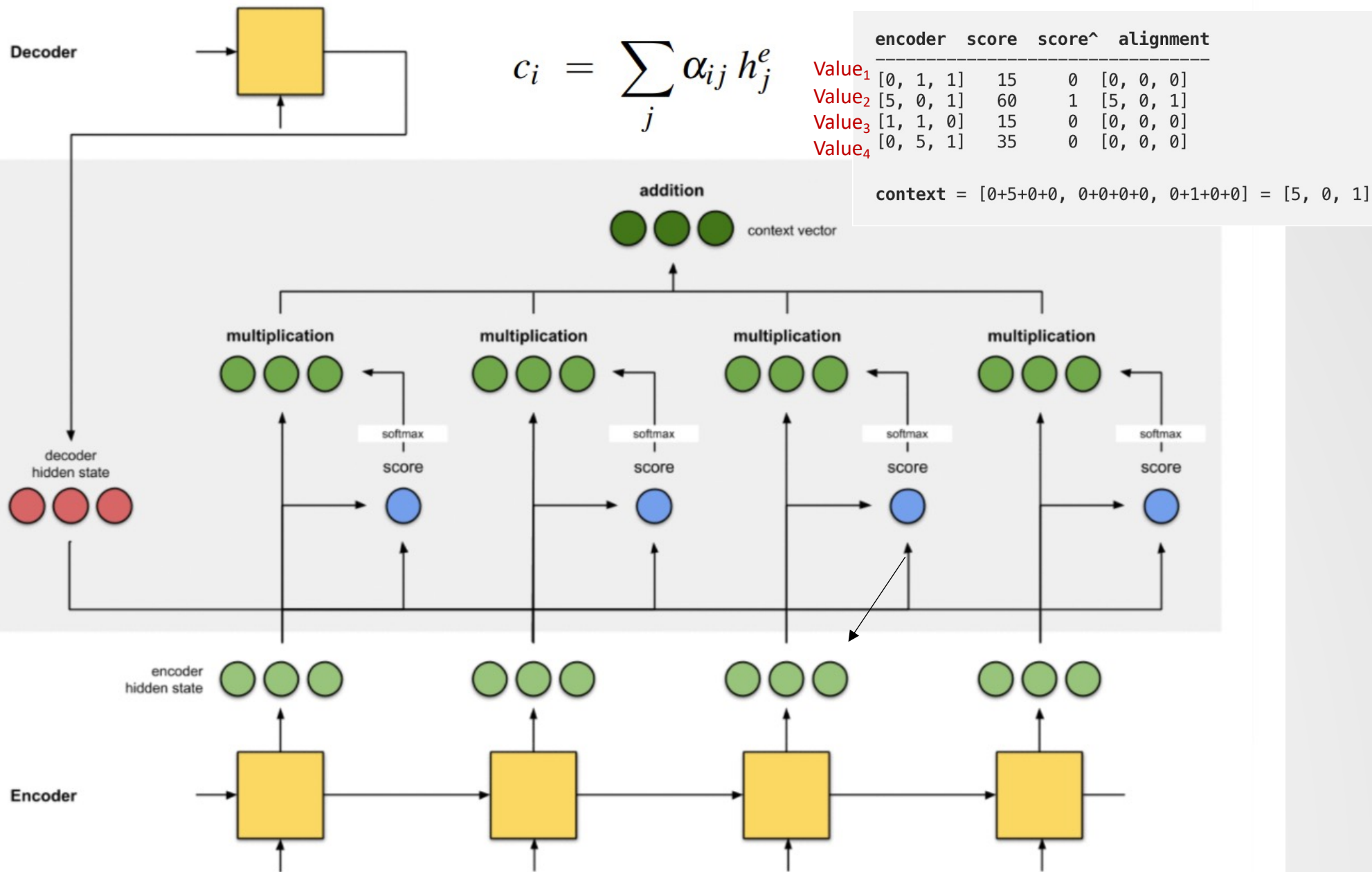
$$\alpha_{11} = \frac{15}{125} = 0.12 \sim 0$$

$$\alpha_{12} = \frac{60}{125} = 0.48 \sim 1$$

$$\alpha_{13} = \frac{15}{125} = 0.12 \sim 0$$

$$\alpha_{14} = \frac{35}{125} = 0.28 \sim 0$$





$$c_i = \sum_j \alpha_{ij} h_j^e$$

Value₁
Value₂
Value₃
Value₄

encoder	score	score [^]	alignment
[0, 1, 1]	15	0	[0, 0, 0]
[5, 0, 1]	60	1	[5, 0, 1]
[1, 1, 0]	15	0	[0, 0, 0]
[0, 5, 1]	35	0	[0, 0, 0]

context = [0+5+0+0, 0+0+0+0, 0+1+0+0] = [5, 0, 1]

$$c_i = \sum_j \alpha_{ij} h_j^e$$

context

Attention in Seq2Seq Model

- Instead of simple dot product attention, we get a more powerful function that computes the relevance of each encoder hidden state to the decoder hidden state by parameterizing the score with its own set of weights, W_a .
- The matrix of alignment scores is a nice byproduct to explicitly show the correlation between source and target words.

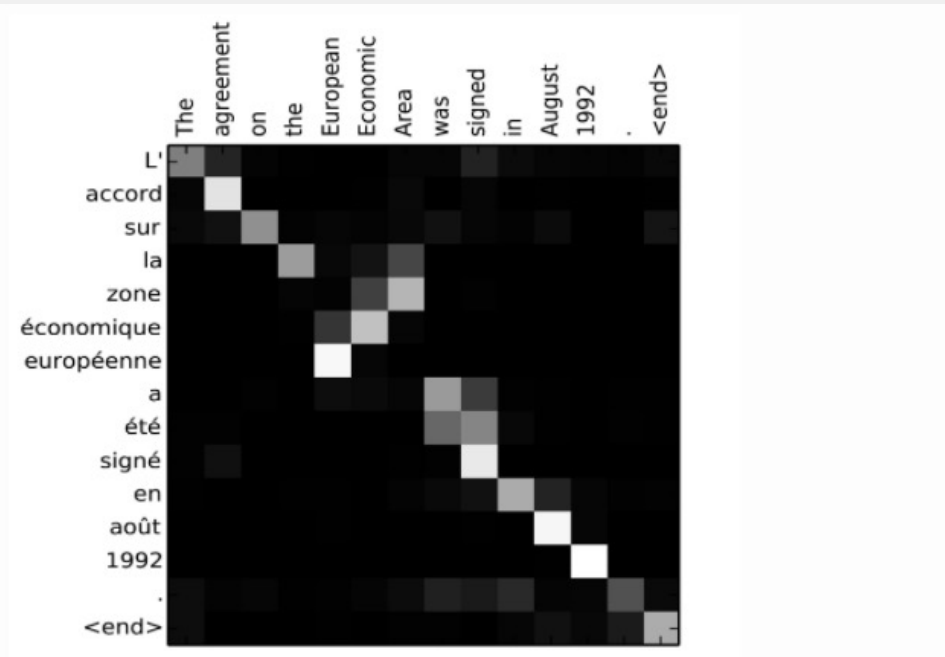
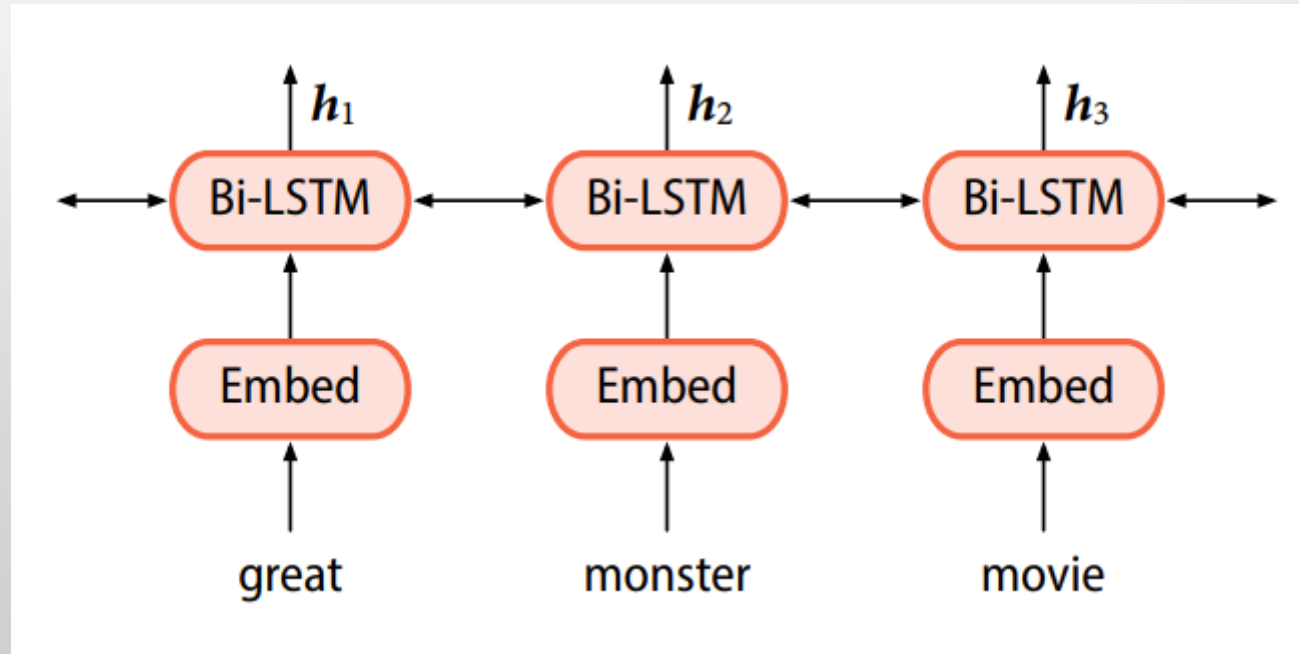


Fig. 5. Alignment matrix of “L'accord sur l'Espace économique européen a été signé en août 1992” (French) and its English translation “The agreement on the European Economic Area was signed in August 1992”. (Image source: Fig 3 in Bahdanau et al., 2015)

Attention in Classification

Attention in Classification

- In the classification architecture, **keys** and **values** are the same; they correspond to the hidden states h_i .



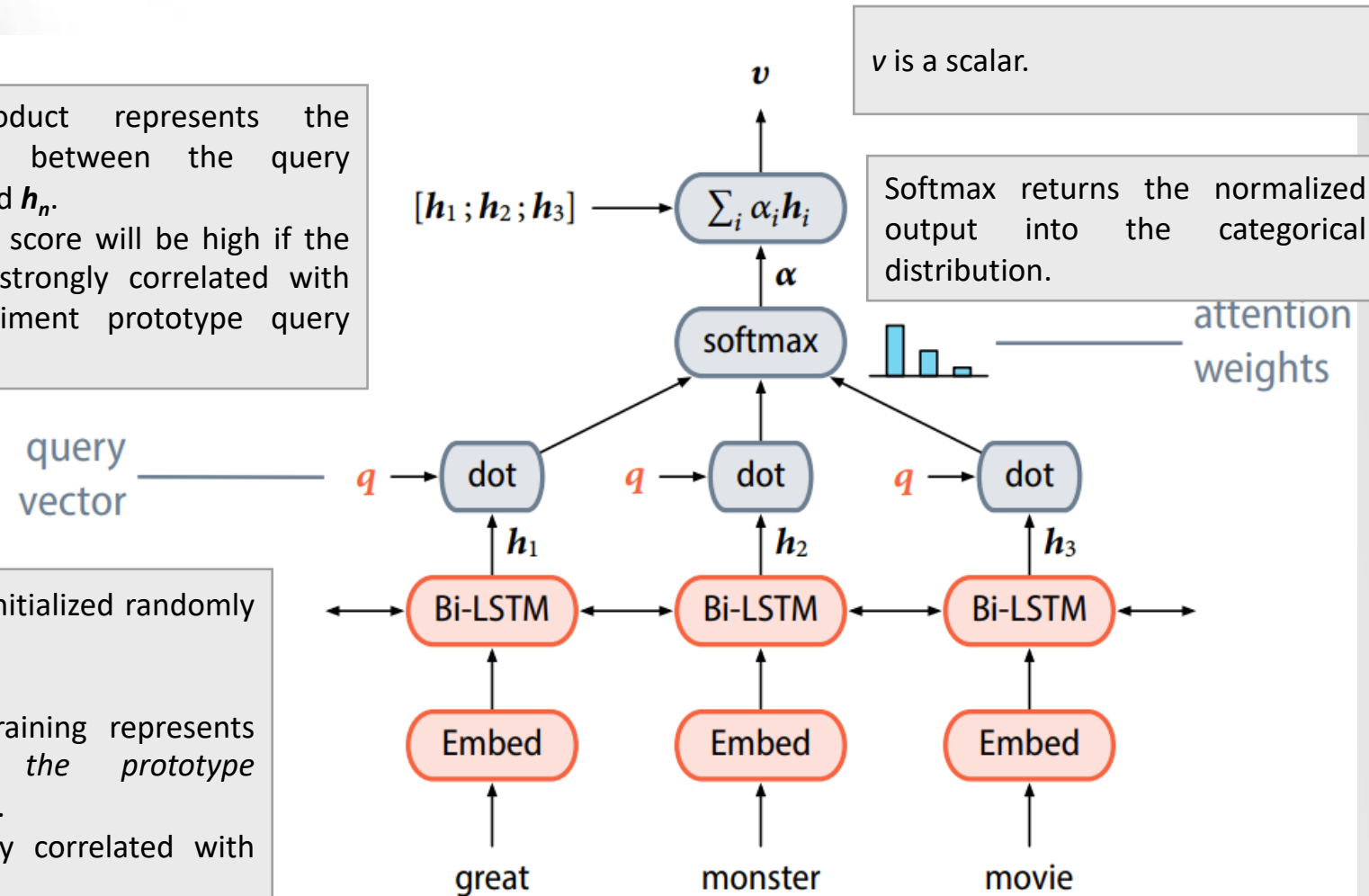
Assume that in this example we want to predict the sentiment of a given sentence.

Attention in Classification

Dot product represents the similarity between the query vector and h_n .
Attention score will be high if the word is strongly correlated with the sentiment prototype query vector.

Query vector q is initialized randomly at the beginning.

Eventually after training represents something like *the prototype sentiment indicator*.
e.g. words strongly correlated with sentiment.



Attention Variants

Attention Variants

- There are several ways you can compute attention scores $e \in \mathbb{R}^N$ from the values $v_1, \dots, v_N \in \mathbb{R}^{d_1}$ and the query $q \in \mathbb{R}^{d_2}$
- Basic dot-product attention: $e_i = q^T v_i \in \mathbb{R}$
 - Note that this assumes $d_1 = d_2$.
 - This is the version we saw earlier.
- Multiplicative attention: $e_i = q^T W v_i \in \mathbb{R}$
 - Where $W \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix
- Additive attention: $e_i = y^T \tanh(W_1 v_i + W_2 q) \in \mathbb{R}$
 - Where $W_1 \in \mathbb{R}^{d_3 \times d_1}$, $W_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $y \in \mathbb{R}^{d_3}$ is a weight vector.
 - d_3 (the attention dimensionality) is a hyperparameter.

Attention Types

- Attention-based models are classified into two broad categories, *global* and *local*.
 - These classes differ in terms of whether the “attention” is placed on all source positions or on only a few source positions.
 - While these models differ in how the context vector c_t is derived, they share the same subsequent steps.
- Attention can be classified into two based on how hidden states are incorporated in the final state.
 - *Soft attention* is when we calculate the context vector as a weighted sum of the encoder hidden states.
 - *Hard attention* is when, instead of weighted average of all hidden states, we use attention scores to select a single hidden state.

Attention

Summary

- Attention significantly improves neural machine translation performance
 - It is very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem.
 - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem.
 - Provides shortcut to faraway states.
- Attention provides some interpretability
 - By inspecting attention distribution, we can see what the decoder was focusing on.
 - The network just learned alignment by itself