

Large Language Models

Bartosz Pankratz

Wprowadzenie

- **Large Language Models (LLMs)** are a class of deep learning models capable of generating text and solving natural language processing tasks. Their characteristic feature is complexity – they usually have billions of trainable parameters. LLMs are trained in an **Unsupervised** or **semi-supervised** manner on large text datasets.

NLP – Brief History

- In 1950s, Shannon noticed that language could be modeled as probability.
- For most of the twentieth century, the dominant paradigm of **natural language processing (NLP)** was the creation of expert models consistent with linguistic theories (e.g., Chomsky's concept of generative grammar).
- The 1990s saw a resurgence in the statistical approach to analysis of language.

NLP – Brief History

- **N-gram** is a class of models where text (*sequence*) is modelled as a conditional probability $P(x_i|x_0, x_1, \dots, x_{i-1})$. Example:

$$P(\text{Dogs}|\text{I}, \text{Like})$$

NLP – Brief History

- Most common N-gram models:
 - **Unigram** – each probability is independent:

$$P(I, \text{Like}, \text{Dogs}) = P(I)P(\text{Like})P(\text{Dogs})$$

- **Bigram:**

$$P(\text{Ala}, \text{ma}, \text{kota}) = P(I|\langle s \rangle)P(\text{Like}|I)P(\text{Dogs}|\text{Like})$$

- **Trigram:**

$$P(\text{Ala}, \text{ma}, \text{kota}) = P(I|\langle s \rangle, \langle s \rangle)P(\text{Like}|\langle s \rangle, I)P(\text{Dogs}|I, \text{Like})$$

Neural networks

Since the early 2000s, N-gram models have been built using neural networks.

The Bengio et al. 2003 model was a significant breakthrough, vastly improving over existing frequentist approaches.

Moreover, such a model learned to embed words in a low-dimensional latent space. of a continuous real numbers. This significantly reduced the model's complexity, allowing it to avoid the **curse of dimensionality**.

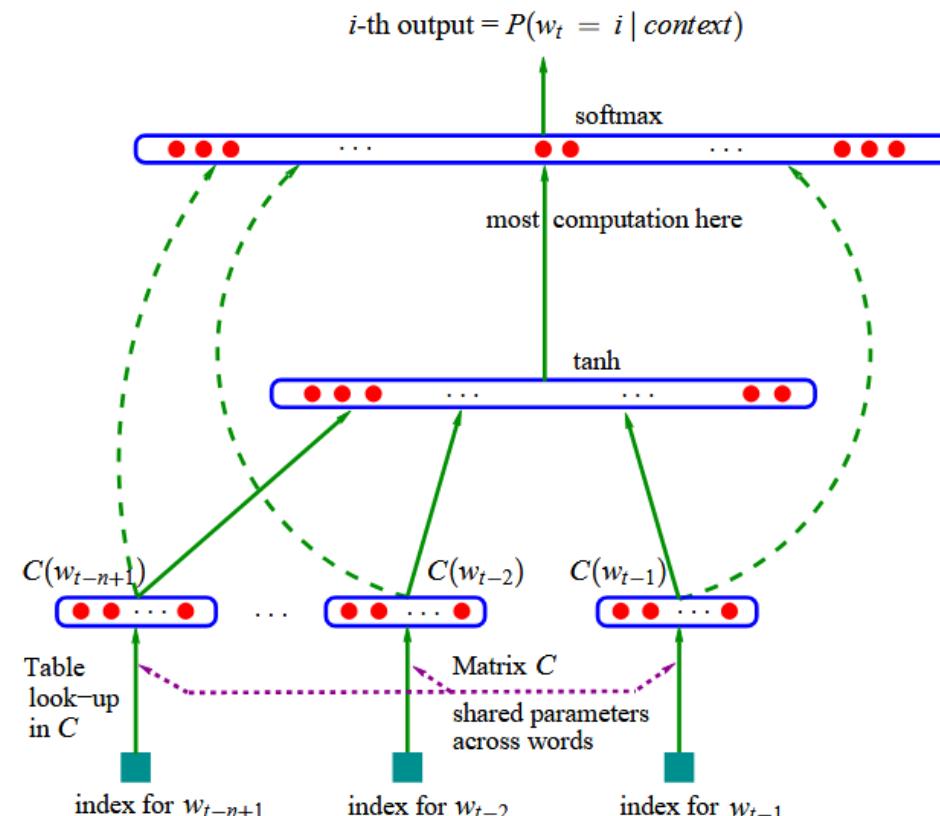


Figure 1: Neural architecture: $f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i -th word feature vector.

Source: Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.* 3, null (3/1/2003), 1137–1155.

Neural Networks

The embeddings of text were a breakthrough that allowed for the construction of models that solve classic NLP tasks (e.g. document classification, tone analysis) more efficiently.

A good example is a **word2vec** model:

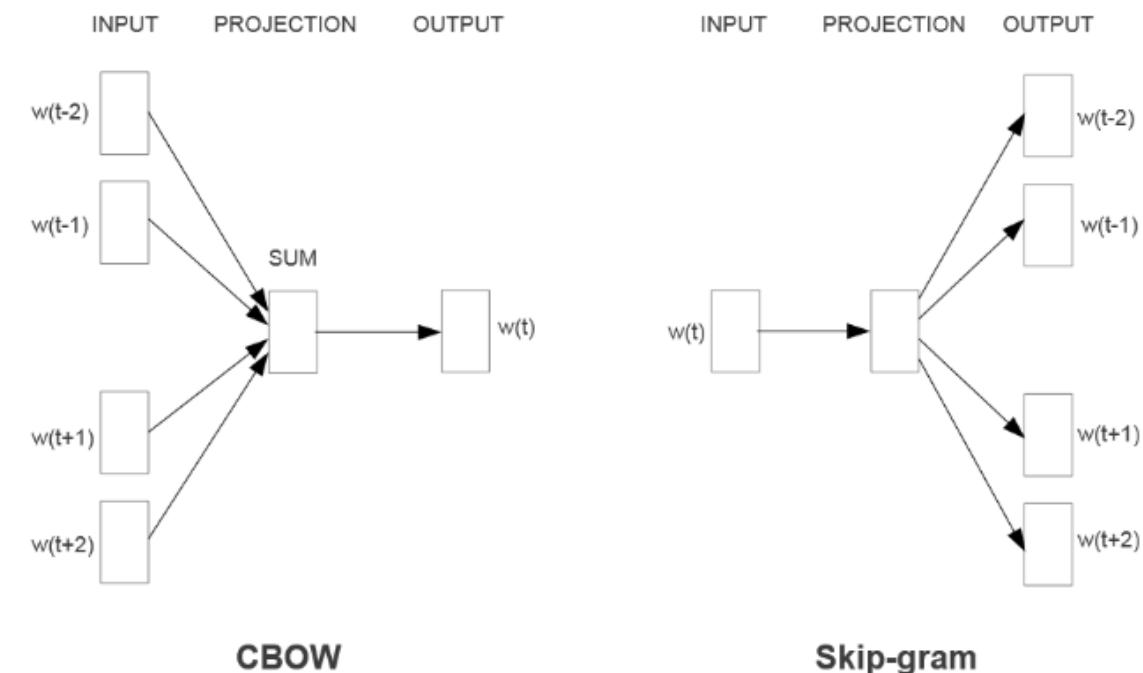


Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Source: Mikolov, Tomas & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR. 2013.

word2vec

In a **SkipGram**, model, each token x_i is surrounded by the **context window** of length $2\ell + 1$. centered around x_i . The goal of the model is to maximize the probability of observing the proceeding ℓ and succeeding ℓ token. For example, in the sentence:

Network science is an academic field which studies complex networks .

The model is trained to predict the words in italics given the word **field** as input.

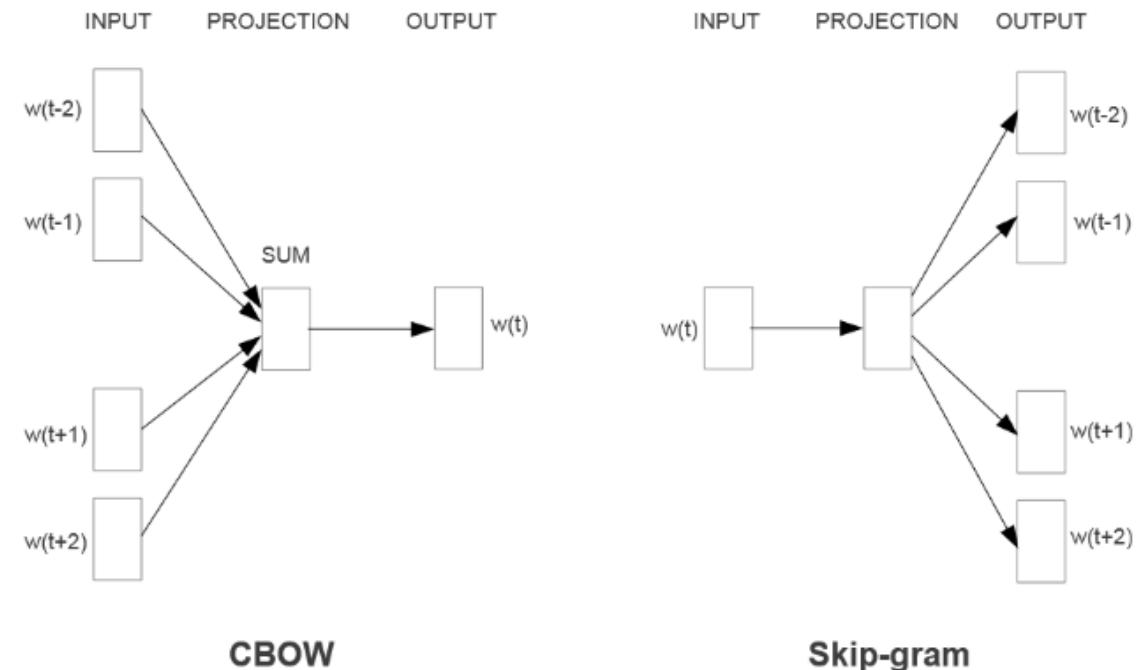


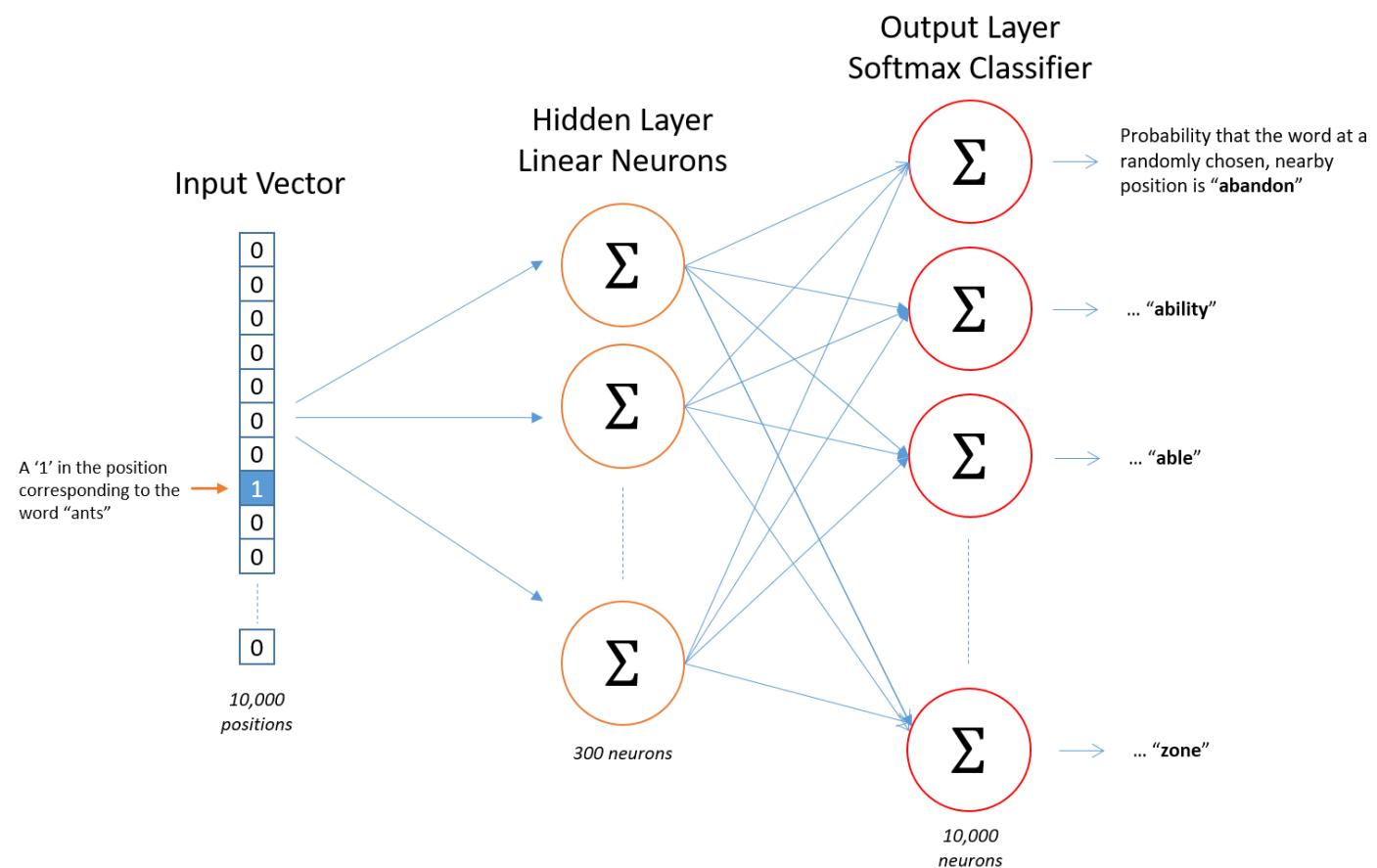
Figure 1: New model architectures. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

Source: Mikolov, Tomas & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR. 2013.

word2vec

Let x_i be a token from the text corpus \mathcal{D} , with N unique tokens (words). Word2vec is defined as a two-layer model:

- First layer, $E_s \in \mathbb{R}^{N \times d}$ is a linear layer which embeds the input word x_i as a latent vector z_i .
- Second layer, $E_t \in \mathbb{R}^{d \times N}$ tries to predict the probability that input is surrounded by kth word for $k \in [1, 2, \dots, N]$.



Source: <https://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>

word2vec

- The objective function is as follows:

$$\max_z \sum_{i=1}^N \log P(\{x_{i-\ell}, \dots, x_{i-1}, x_{i+1}, \dots, x_{i+\ell}\} | z_i)$$

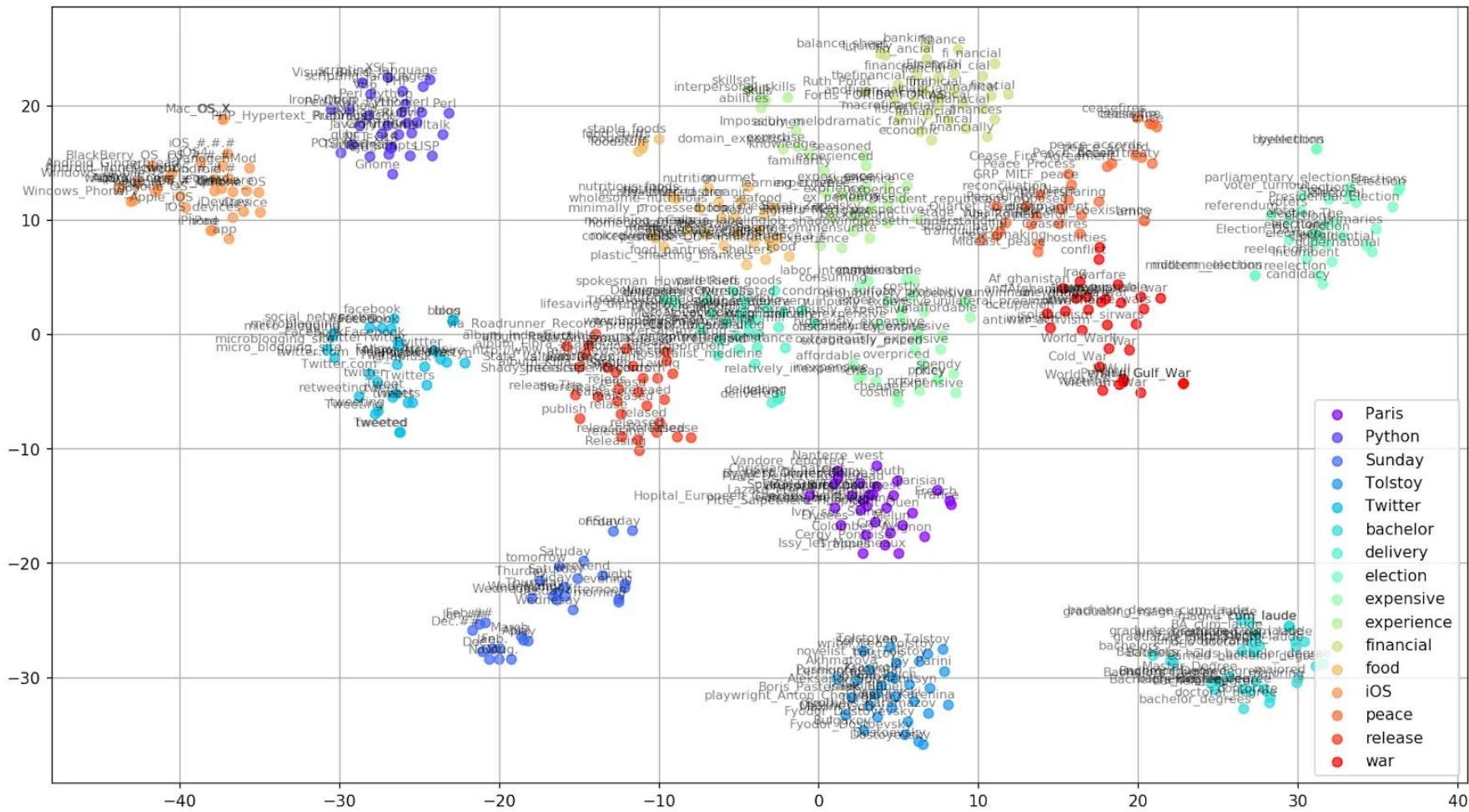
where $z_i = x_i^\top E_s$. After simplification:

$$\max_z \sum_{i=1}^N \log P \prod_{-\ell \leq j \geq \ell, j \neq i} P(z_{i+j} | z_i)$$

- where $P(z_{i+j} | z_i)$ is approximated by the **softmax**:

$$P(z_{i+j} | z_i) = \frac{\exp(e_{t,i+j}^\top e_{s,i})}{\sum_{k=1}^N \exp(e_{t,k}^\top e_{s,i})}$$

word2vec



Source: <https://medium.com/data-science/google-news-and-leo-tolstoy-visualizing-word2vec-word-embeddings-with-t-sne-11558d8bd4d>

NLP – Brief History

- There are two major drawbacks of such approach:
 - The length of the modeled probability chain must be constant and relatively short.
 - Algorithm learn the **meaning** of the word, not its **semantic role**.

Recursive Neural Networks

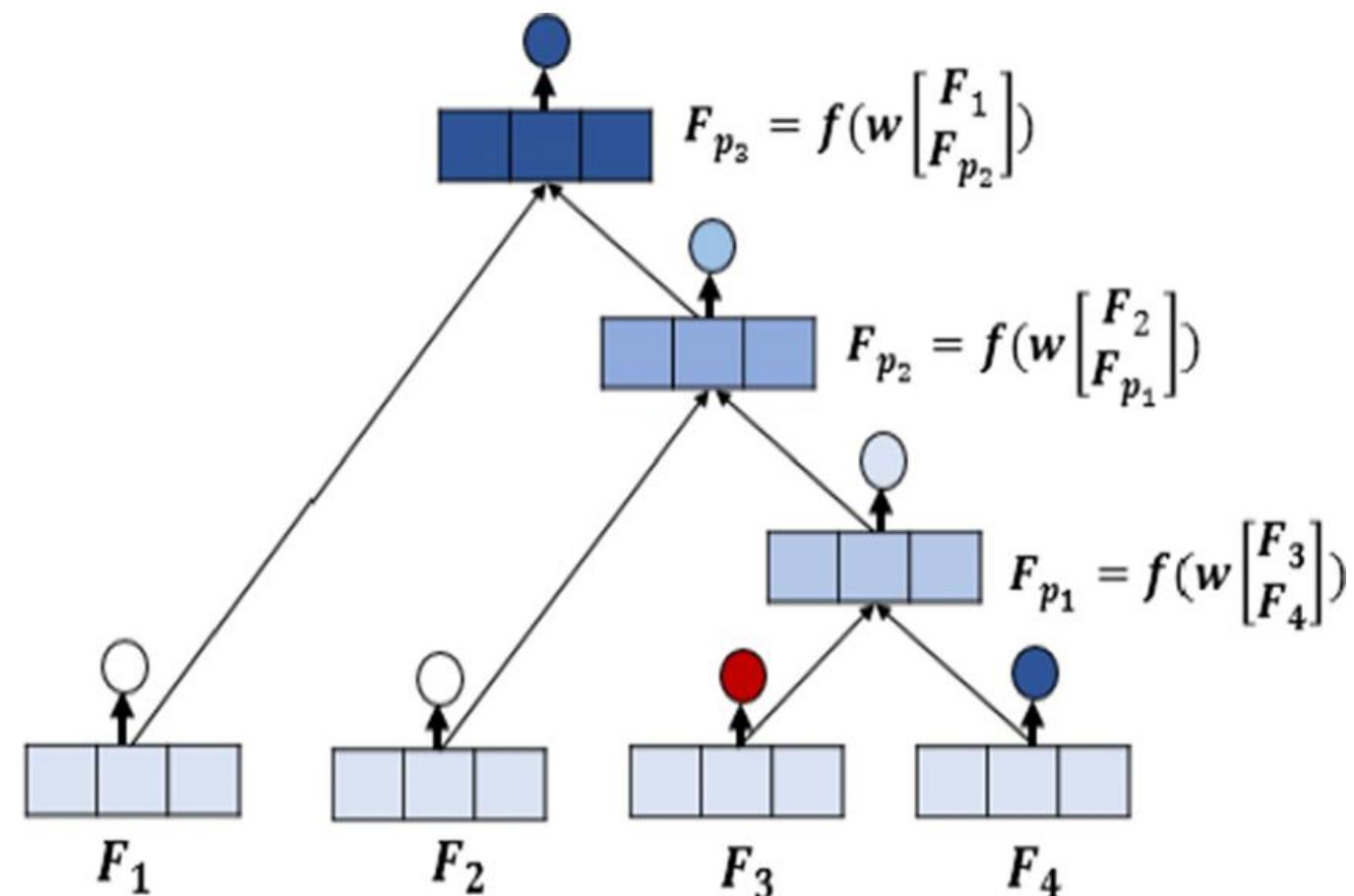
Recursive Neural Networks

(Hammer et al. 2004) were proposed to overcome these issues.

This model is built from two distinct elements:

- a word embedding,
- and a tree-like structure (**Parse tree**) representing data.

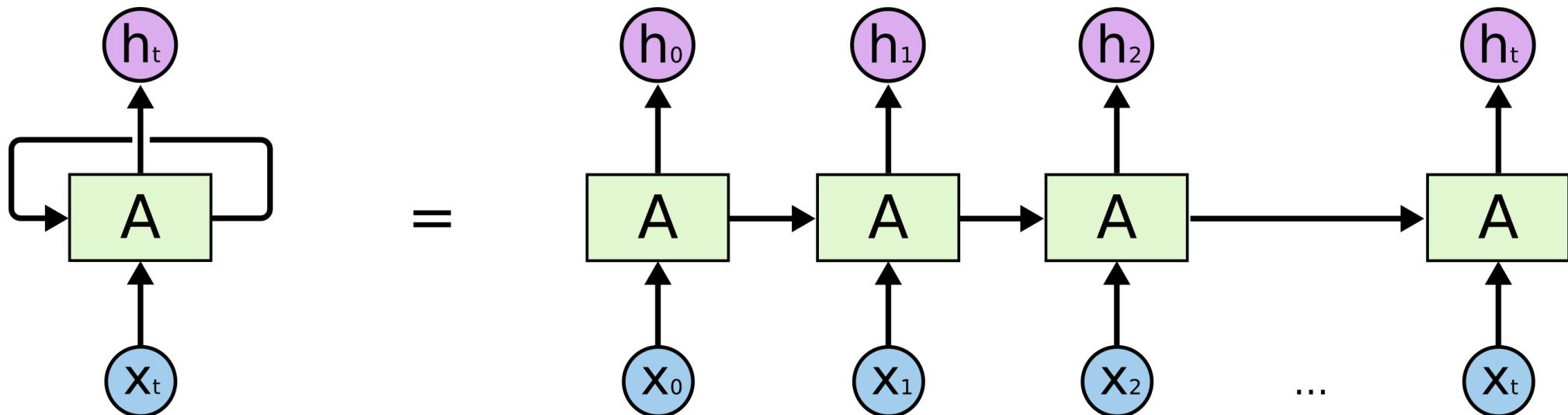
The model learns a weight vector that is used to aggregate successive vertices of a parse tree (going from leaf to root).



Source: Sadr, Hossein & Pedram, Mir Mohsen & Teshnehab, Mohammad. (2019). A Robust Sentiment Analysis Method Based on Sequential Combination of Convolutional and Recursive Neural Networks. *Neural Processing Letters*. 50. 10.1007/s11063-019-10049-1.

Recurrent Neural Networks

- **Recurrent Neural Networks (RNN)** are networks that allow information to be propagated not just from layer to layer, but across each layer:



Source: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Recurrent Neural Networks

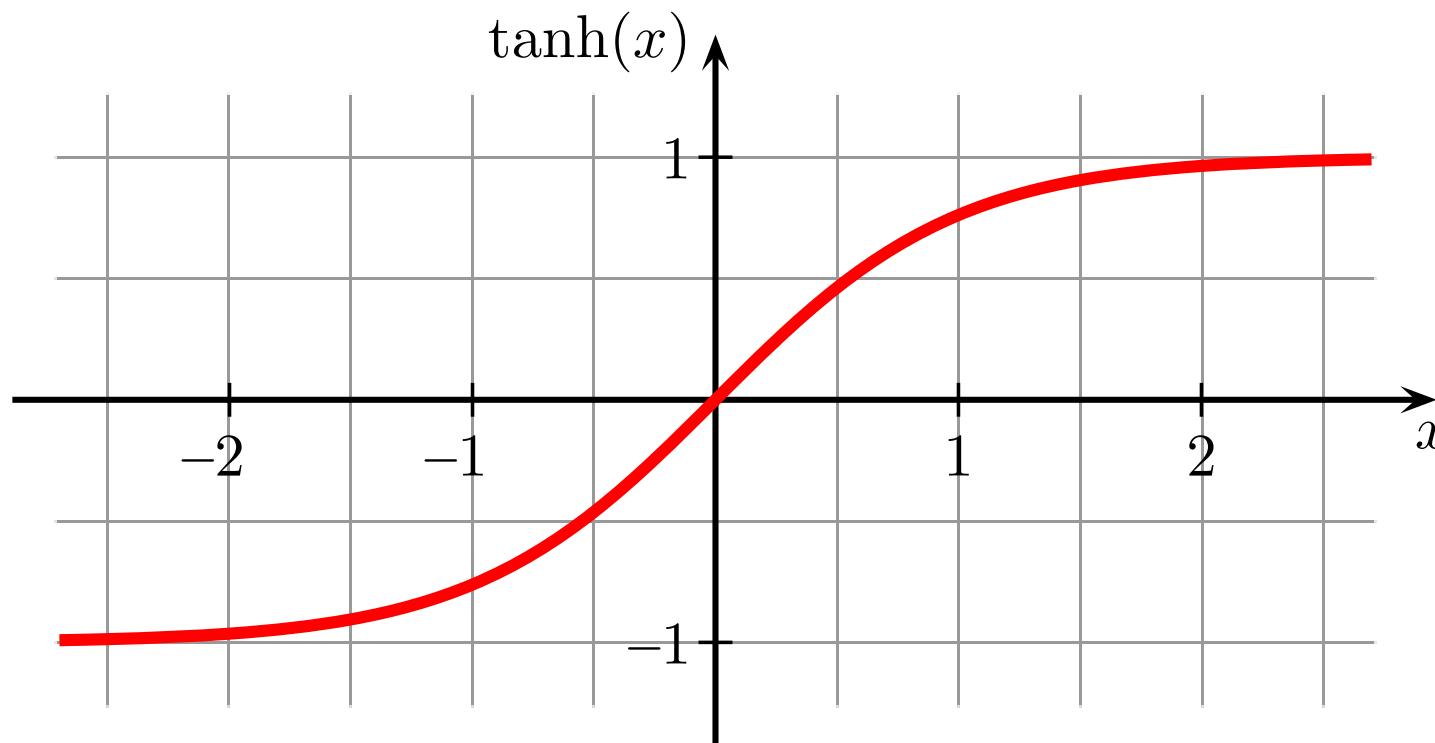
- A key element of such a model is its **hidden state** h_t , which is propagated between neurons on the same layer.
- **Elman Net** (1990) is a classic example of RNN model. h_t is computed as follows:

$$h_t = f_h(\theta_x x_t + \theta_h h_{t-1} + b)$$

for $t = 1, 2, \dots, d$, where f_h is an activation function (**hyperbolic tangent**), h_{t-1} is a hidden state of a neighboring neuron, x_t is an input from the previous layer, and θ_x, θ_h, b are trainable weights.

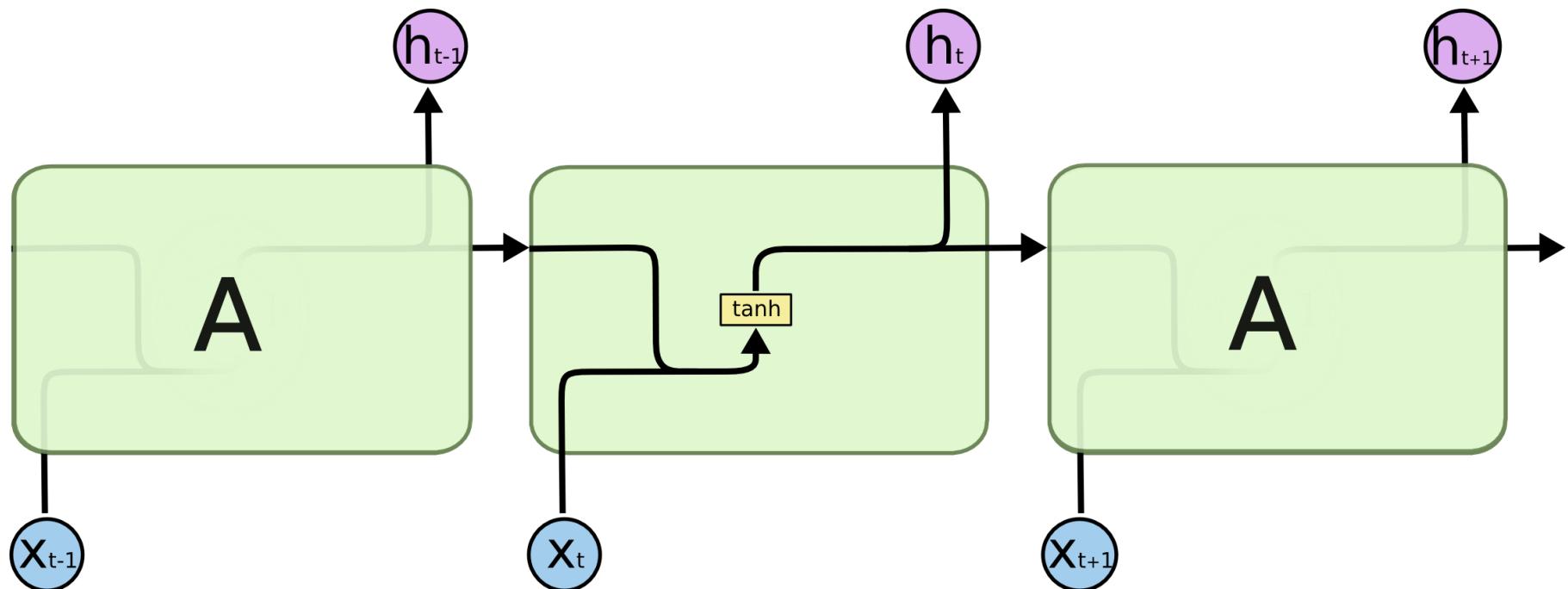
Recurrent Neural Networks

Hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



Recurrent Neural Networks

- A key element of such a model is its **hidden state** h_t , which is propagated between neurons on the same layer:

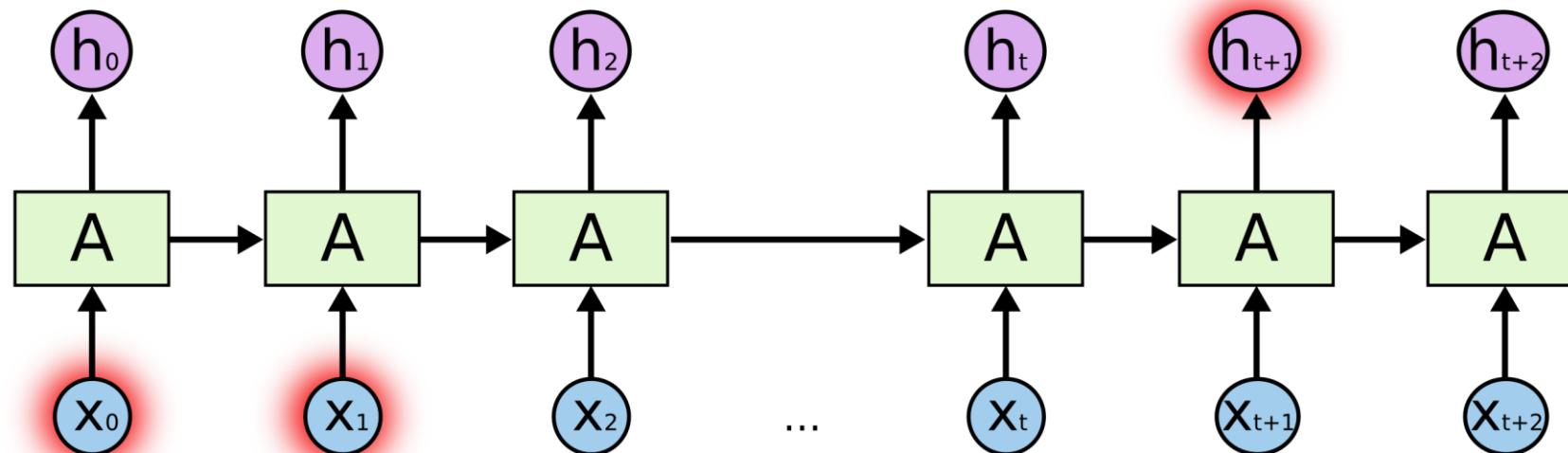


Source: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Sieci rekurencyjne

- However, basic RNNs are unable to model **long-term relationships** between words:

Janek ate a ripe and juicy apple for breakfast, then went back to work.



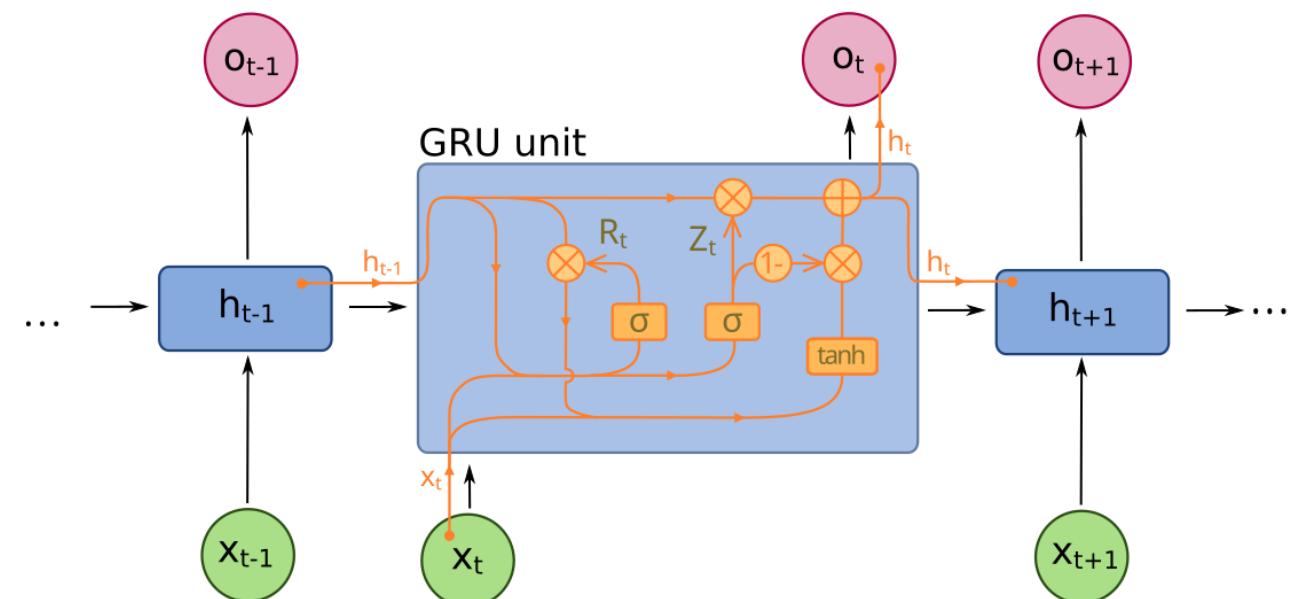
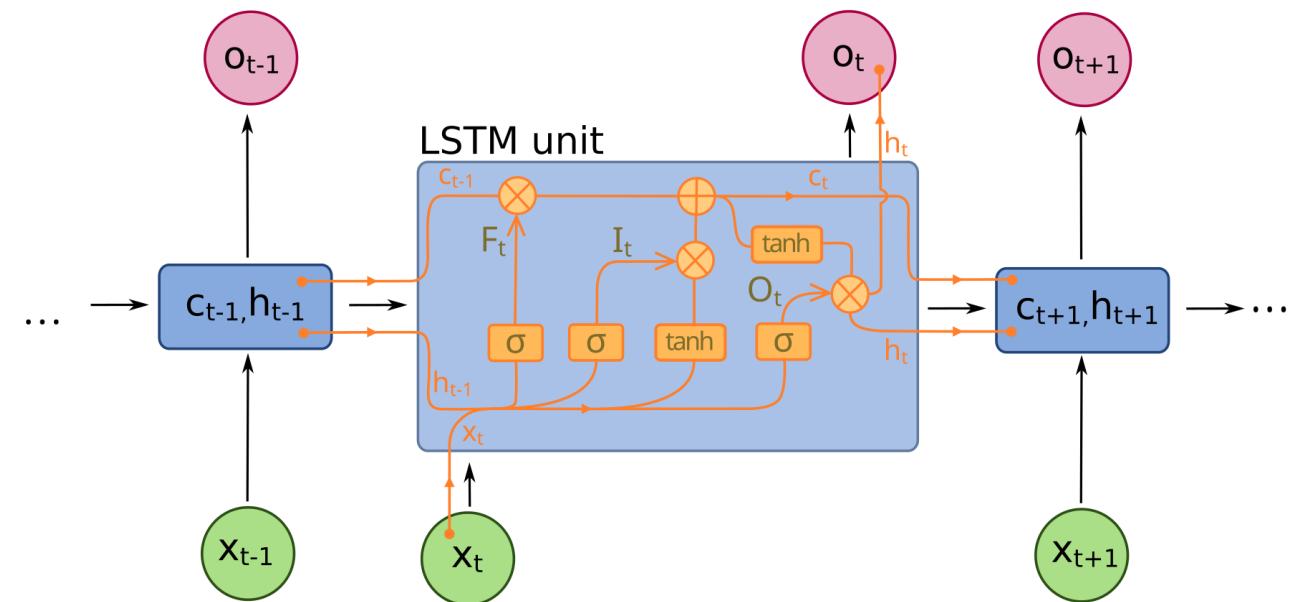
Source: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Recurrent Neural Networks

The addition of a sophisticated **filtering mechanism** helps with this issue. Modern RNNs can control which information should be stored in a hidden state h_t , and which should be forgotten.

Examples

- Long Short-Term Memory (LSTM) (Hochreiter i Schmidhuber 1997)
- Gated Recurrent Units (GRU) (Cho et al. 2014)

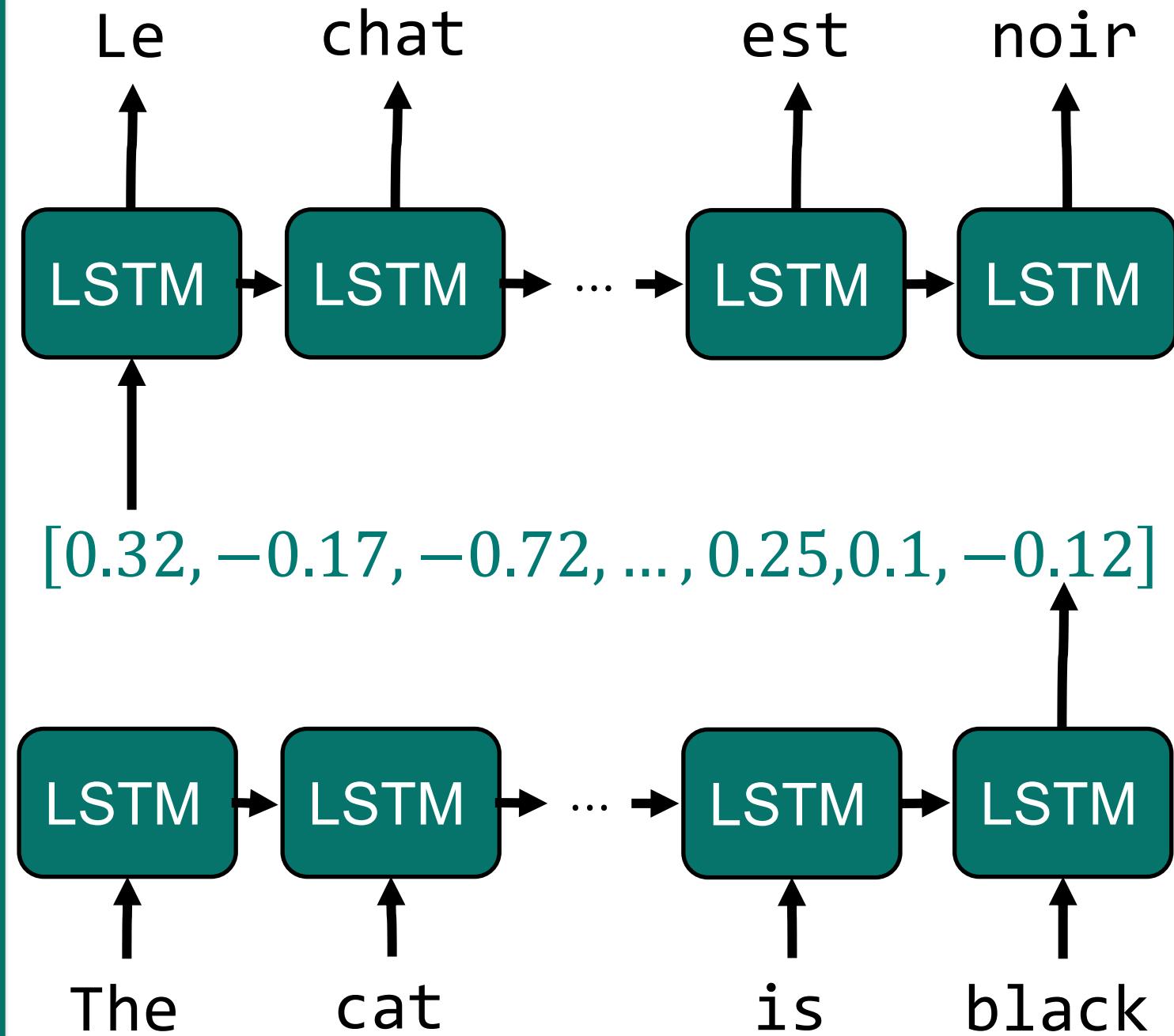


Source: https://en.wikipedia.org/wiki/Recurrent_neural_network

Recurrent Neural Networks

Memory models (LSTM or GRU) have broadened the neural network applications.

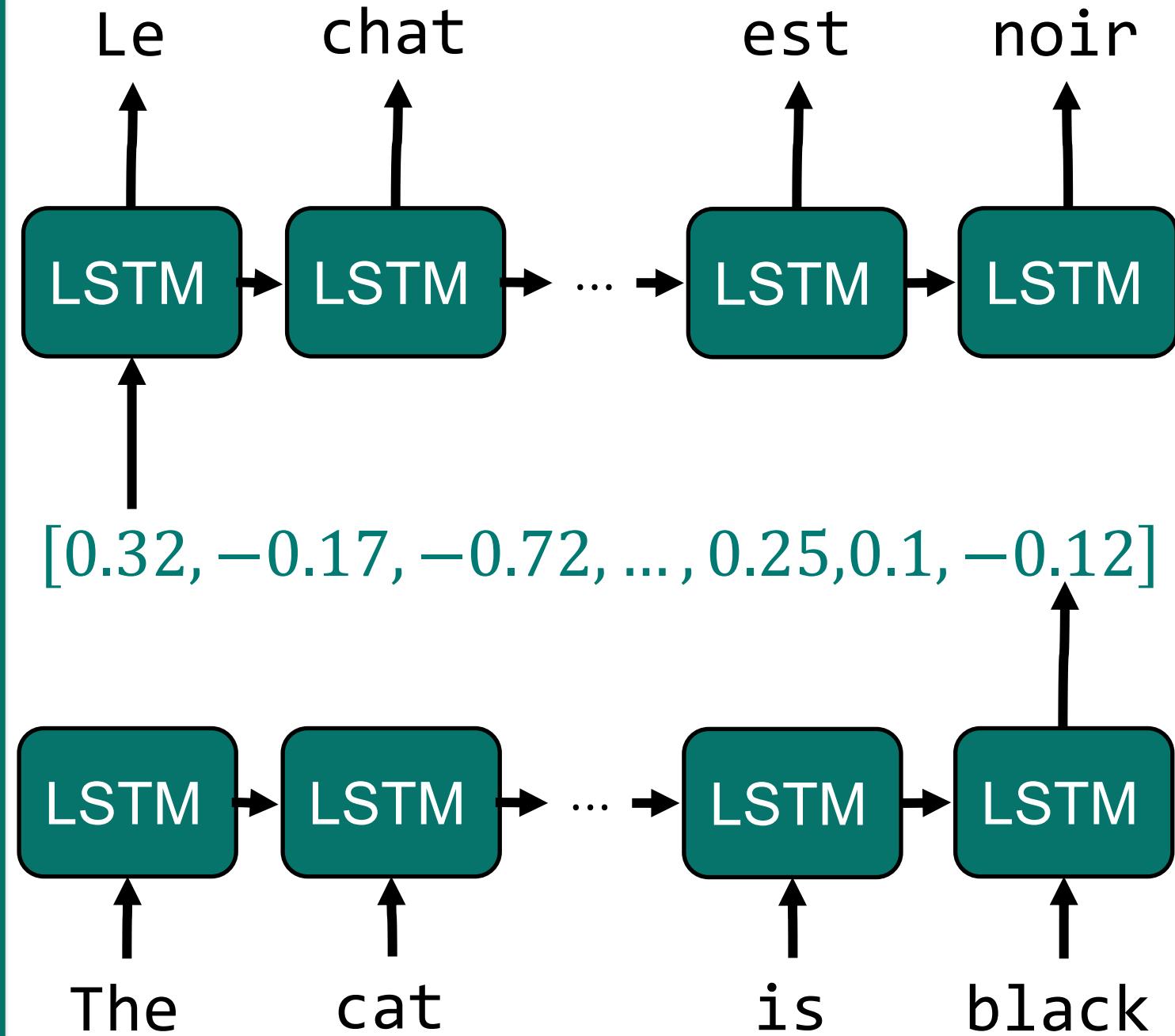
seq2seq (Sutksever et al. 2014), that map one sequence to another (e.g., to translate it, summarize it, or generate a caption under an image).



Recurrent Neural Networks

However, such a model is still just an **n-gram**.

Information flows in only one direction, even if relationships between words in a sentence can be more complicated.



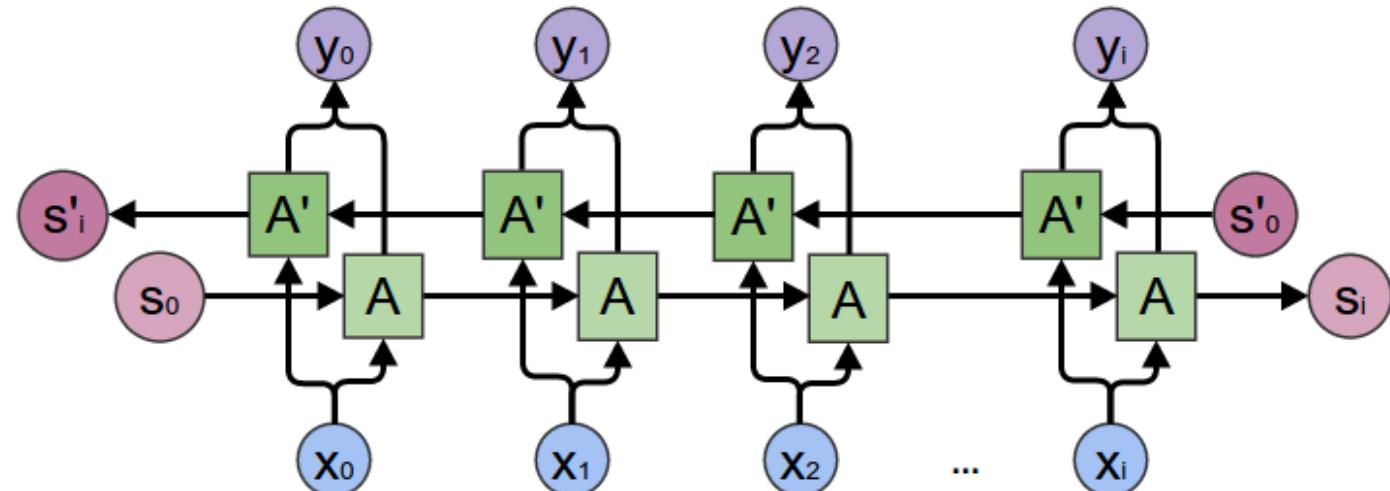
Recurrent Neural Networks

However, such a model is still just an **n-gram**.

Information flows in only one direction, even if relationships between words in a sentence can be more complicated.

How to improve such a model?

- By using the **bidirectional network**.



Source: <https://colah.github.io/posts/2015-09-NN-Types-FP/>

Recurrent Neural Networks

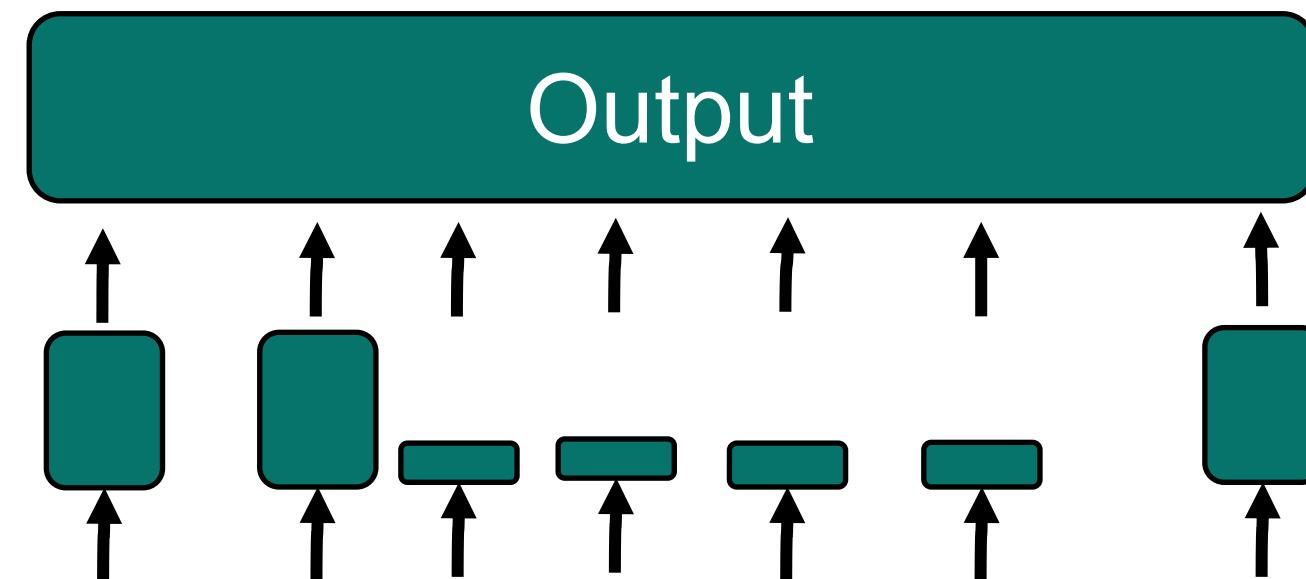
However, such a model is still just an **n-gram**.

Information flows in only one direction, even if relationships between words in a sentence can be more complicated.

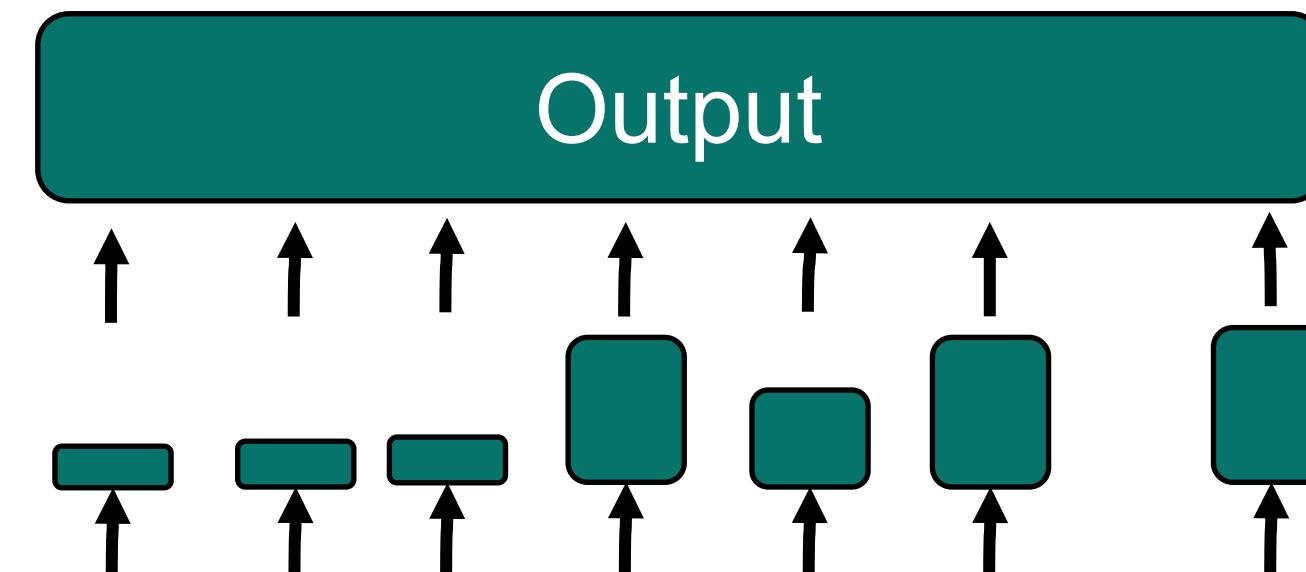
How to improve such a model?

- By using the **bidirectional network**.
- Or by introducing the **Attention mechanism**, that learns non-trivial interactions between words.

What did Janek do?



What is an apple like?



Janek ate a ripe and juicy apple.

Attention Mechanism

For a given sequence x of length n :

$$x = [x_1, x_2, \dots, x_n]$$

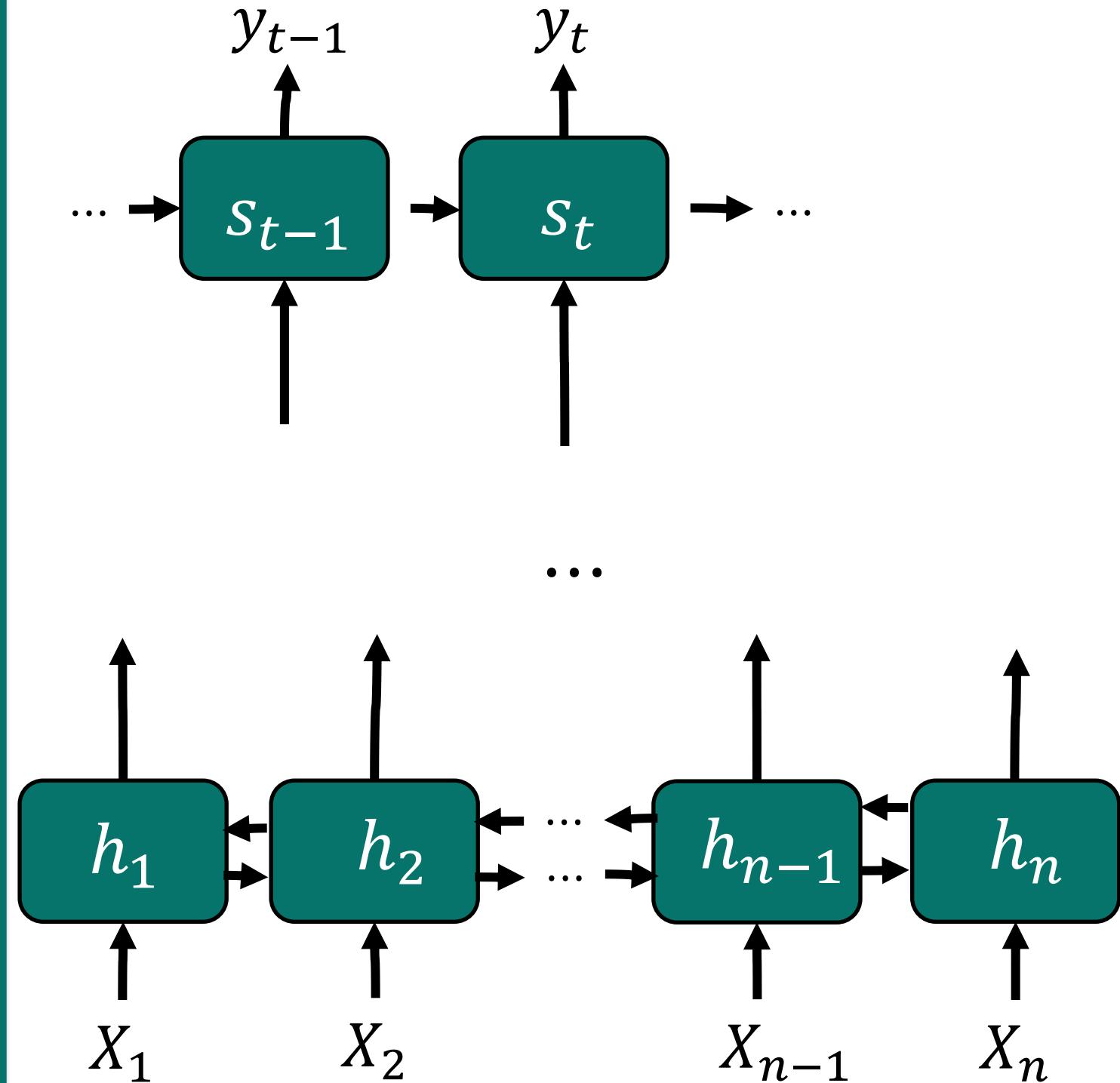
Our goal is to generate an output sequence y of length m :

$$y = [y_1, y_2, \dots, y_m]$$

The input model (*encoder*) is a bidirectional RNN, such that the hidden state h_i is the concatenation of the hidden state for both directions of the model:

$$h_i = [\vec{h}_i; \overleftarrow{h}_i]$$

Output (*decoder*) model is a one direction RNN with a hidden state s_t for $t = 1, 2, \dots, m$.



Attention Mechanism

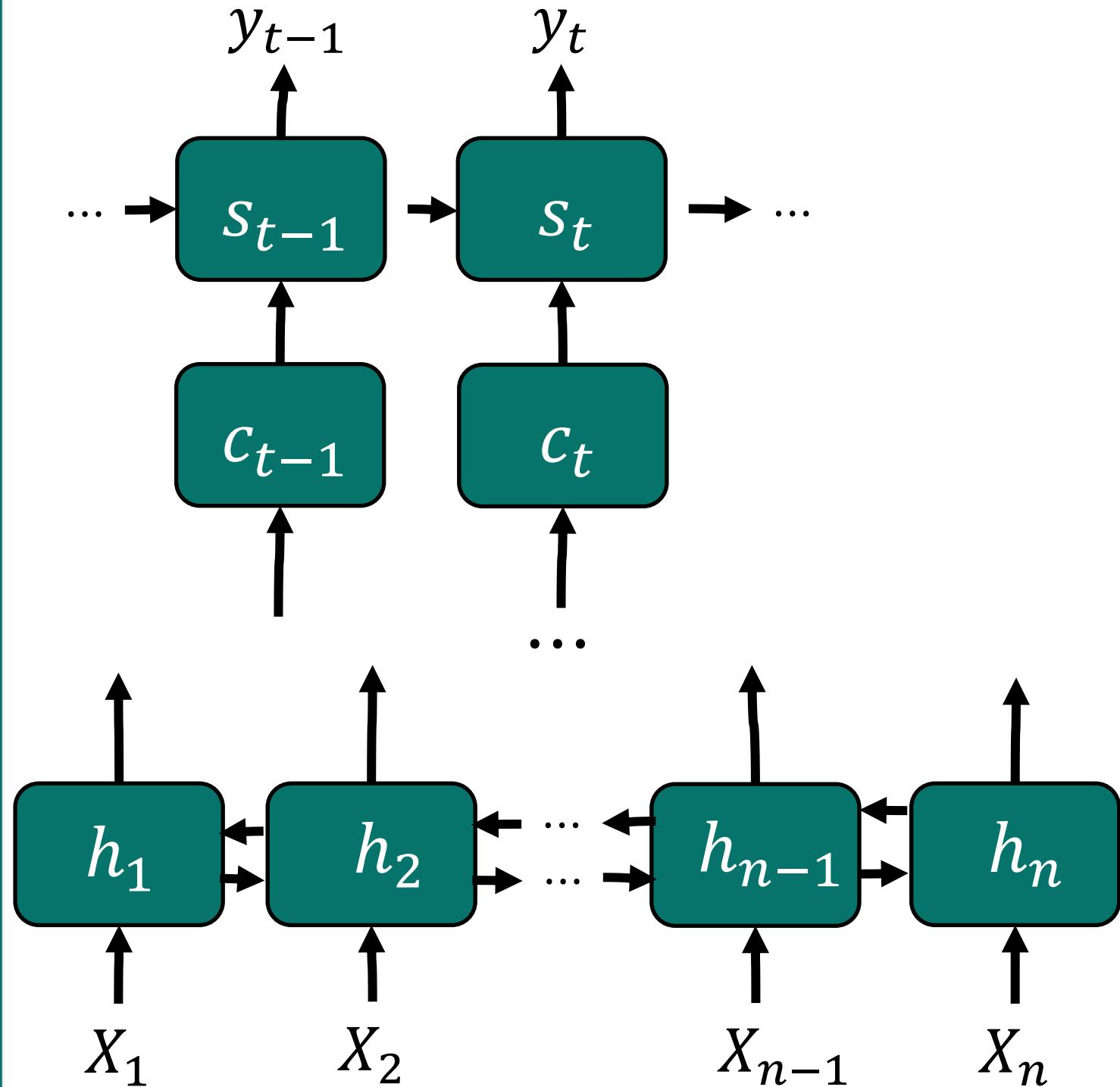
Prediction of next word y_t is modeled as a conditional probability:

$$P(y_t | \{y_1, y_2, \dots, y_{t-1}\}, c_t) = g(y_{t-1}, s_t, c_t)$$

where c_t is a **context vector**, representing the influence of the vector x on y_t .

In particular, the hidden state s_t is modelled as a function:

$$f(y_{t-1}, s_{t-1}, c_t)$$

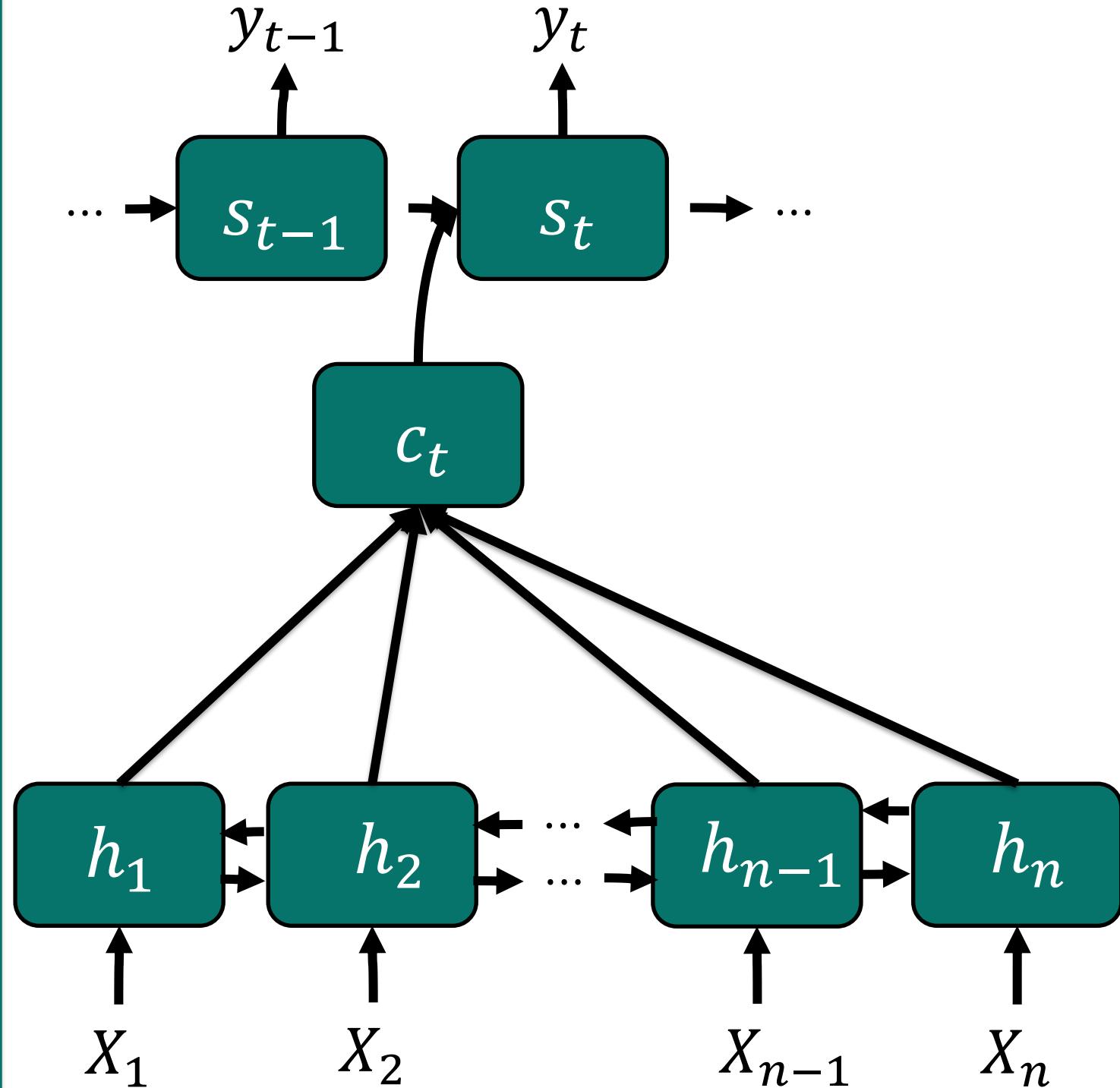


Attention Mechanism

Context c_t depends on the hidden states $h = [h_1, h_2, \dots, h_n]$ and on **impact (weight)** of each h_i on the s_t . Let us denote weight as α_{it} and let $0 \leq \alpha_{it} \leq 1$.

Then, c_t is a linear combination of α_{it} and h_i :

$$c_t = \sum_{i=1}^n \alpha_{it} h_i$$

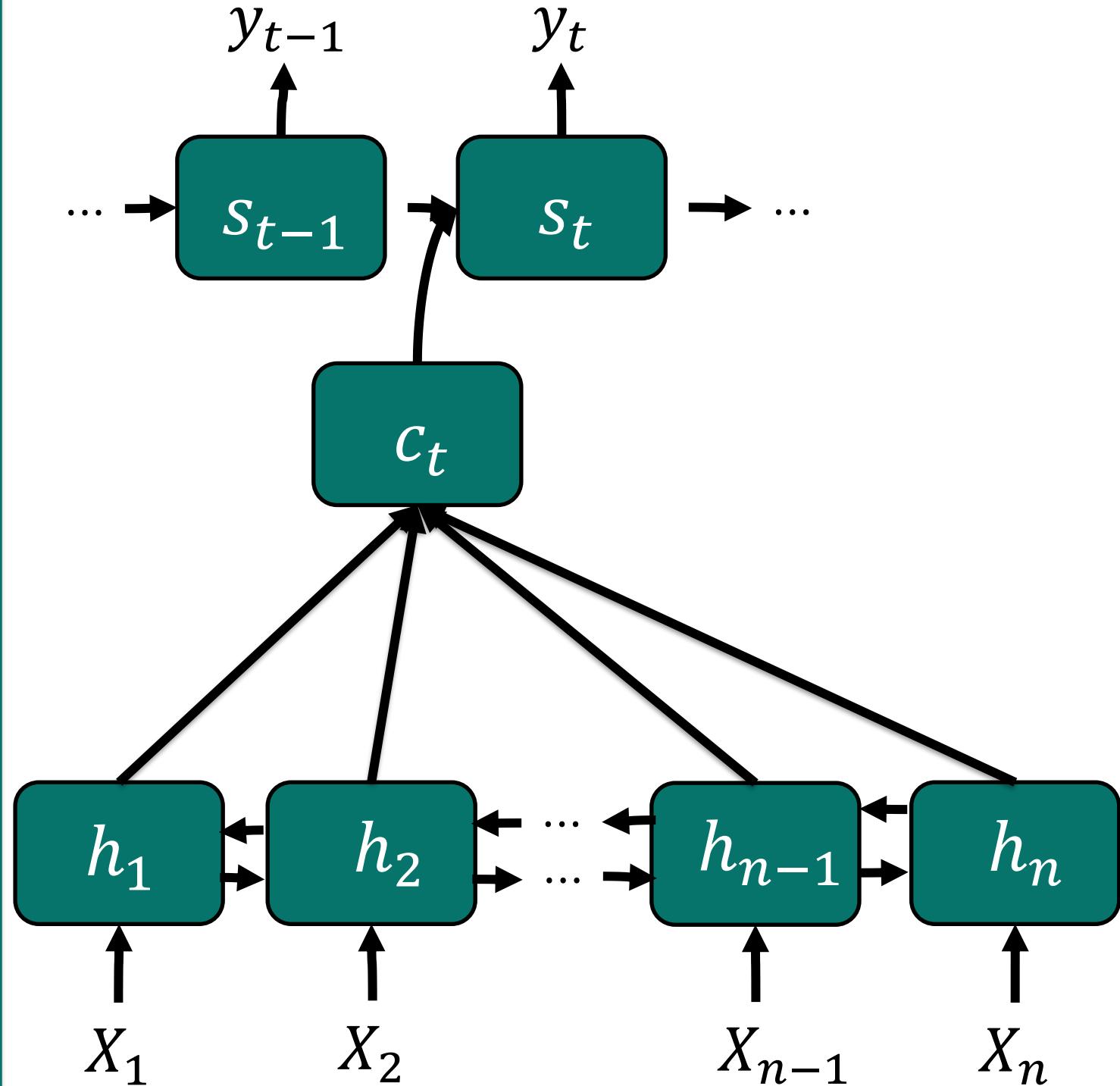


Attention Mechanism

Weight α_{it} is computed by:

$$\alpha_{it} = \frac{\exp(e_{it})}{\sum_{k=1}^n \exp(e_{kt})}$$

where e_{it} is a **score function** in a form: $e_{it} = d(h_i, s_{t-1})$.



Attention Mechanism

Weight α_{it} is computed by:

$$\alpha_{it} = \frac{\exp(e_{it})}{\sum_{k=1}^n \exp(e_{kt})}$$

where e_{it} is a **score function** in a form: $e_{it} = d(h_i, s_{t-1})$.

There are multiple ways of calculating score function (Luong et al. 2015):

- **additive:**

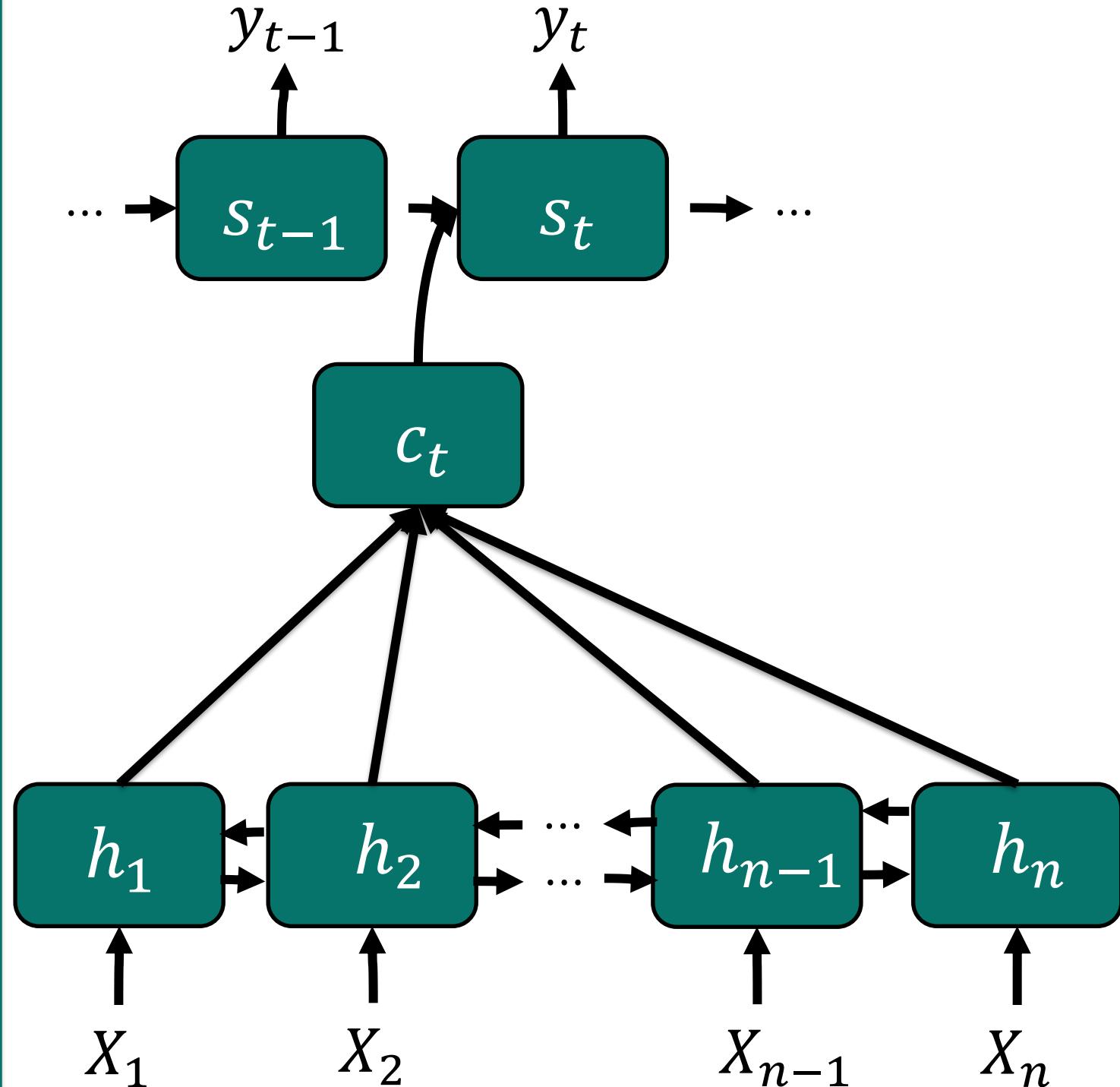
$$d(h_i, s_{t-1}) = \theta_e^\top \text{tahn}(\theta_{hs}[h_i; s_{t-1}])$$

- **general form:**

$$d(h_i, s_{t-1}) = s_{t-1}^\top \theta_{hs} h_i$$

- **dot product:**

$$d(h_i, s_{t-1}) = s_{t-1}^\top h_i$$



Attention Mechanism

Weight α_{it} is computed by:

$$\alpha_{it} = \frac{\exp(e_{it})}{\sum_{k=1}^n \exp(e_{kt})}$$

where e_{it} is a **score function** in a form: $e_{it} = d(h_i, s_{t-1})$.

There are multiple ways of calculating score function (Luong et al. 2015):

- **additive:**

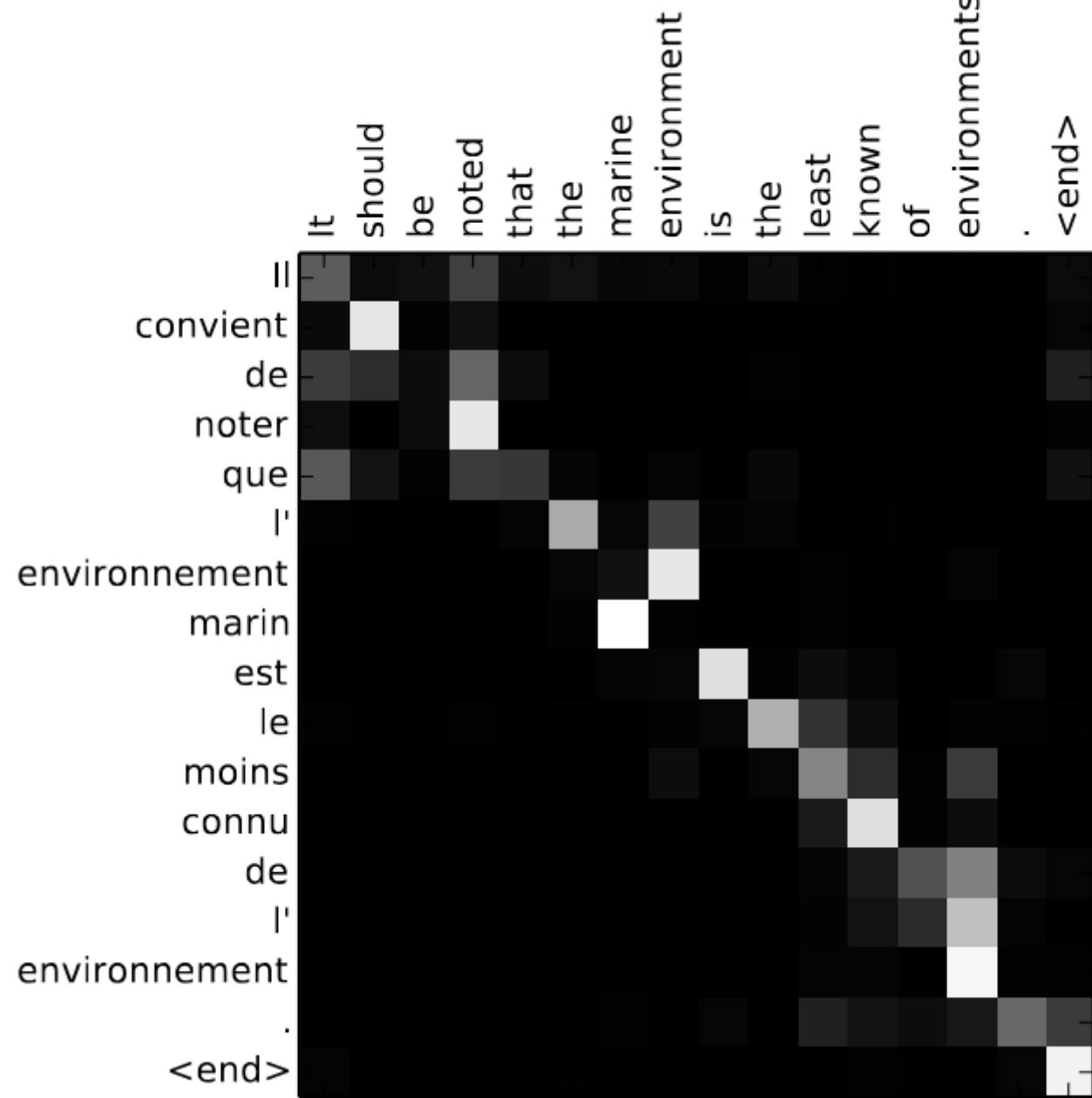
$$d(h_i, s_{t-1}) = \theta_e^\top \text{tahn}(\theta_{hs}[h_i; s_{t-1}])$$

- **general form:**

$$d(h_i, s_{t-1}) = s_{t-1}^\top \theta_{hs} h_i$$

- **dot product:**

$$d(h_i, s_{t-1}) = s_{t-1}^\top h_i$$

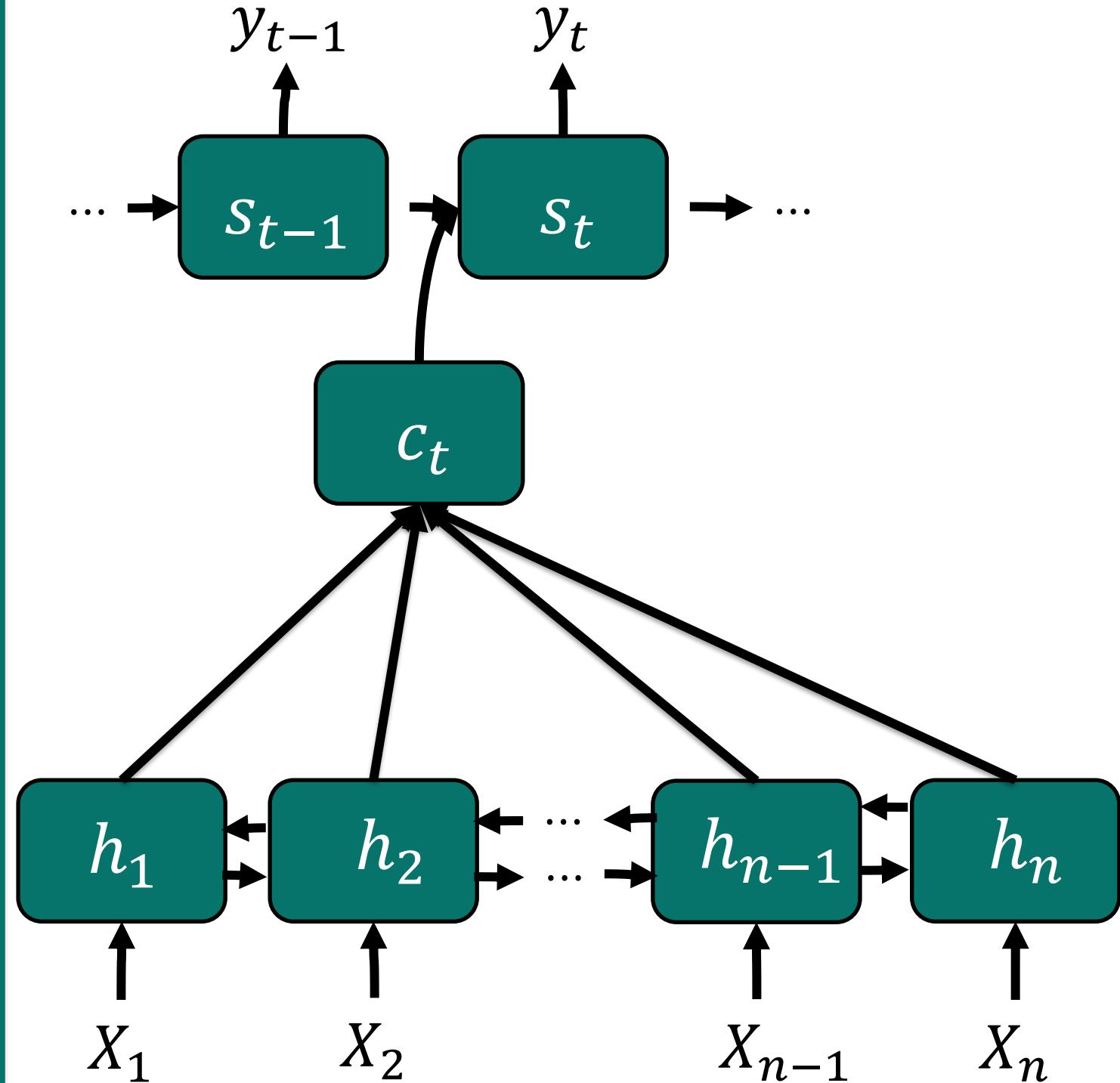


Źródło: Bahdanau, Dzmitry & Cho, Kyunghyun & Bengio, Y.. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. ArXiv. 1409.

Attention Mechanism

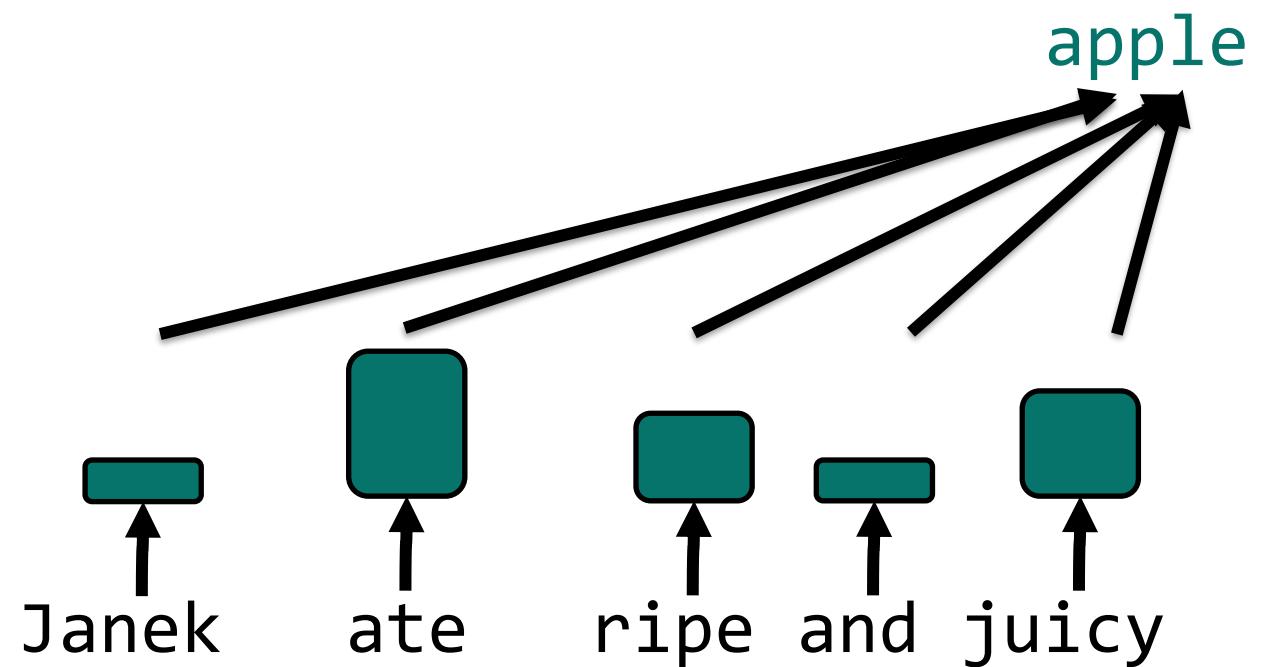
Moreover, attention can be computed:

- **Globally** – We consider all observations in the vector \mathbf{h}
- **Locally** – Attention is computed for the window $[p_t - D, p_t + D]$, where p_t is the index of hidden states vector \mathbf{h} , such that h_{p_t} influence s_t the most.



Attention Mechanism

In the **self-attention mechanism** (Cheng et al. 2016) score is computed only for a single sequence.



Transformers

- Recurrent Neural Networks with an attention mechanism have two major drawbacks:
 - They are **slow**.
 - A combination of bidirectional RNN and attention is **redundant**.
- In *Attention is All You Need* (Vaswani, et al., 2017) authors proposed a new model that uses the attention mechanism without the Recurrent Neural Network backbone.

Transformers

Before we discuss the transformer architecture, let us focus on how data is **tokenized**.

The easiest way is to encode each word (token) in the collected text corpus $x \in \mathcal{D}$ as a binary vector in the corresponding array.

Such coding has two drawbacks:

- It is inefficient; the size of the text corpus is usually very large.
- It is impossible to recognize and generate text that is not part of the corpus (*out-of-vocabulary*, OOV)

$$\begin{bmatrix} I \\ Like \\ Dogs \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

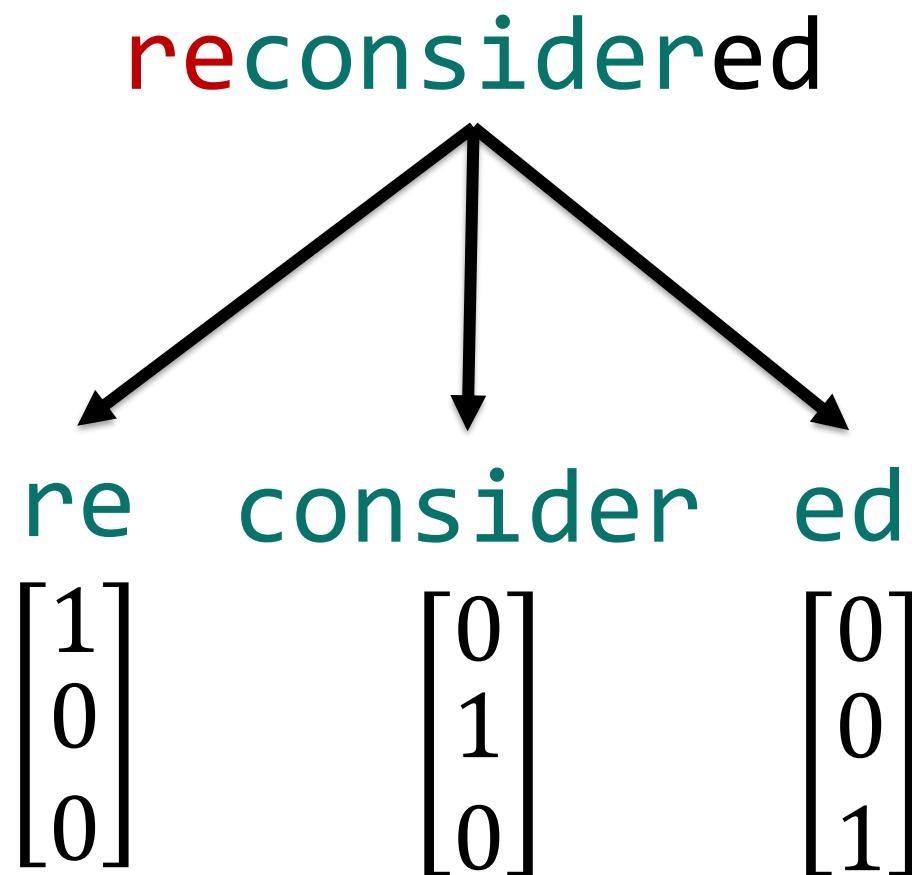
Transformers

Before we discuss the transformer architecture, let us focus on how data is **tokenized**.

Therefore, a more effective way of encoding is to encode morphemes (subwords) instead of full words.

Algorithms:

- Byte-Pair Encoding.
- WordPiece Encoding
- SentencePiece Encoding



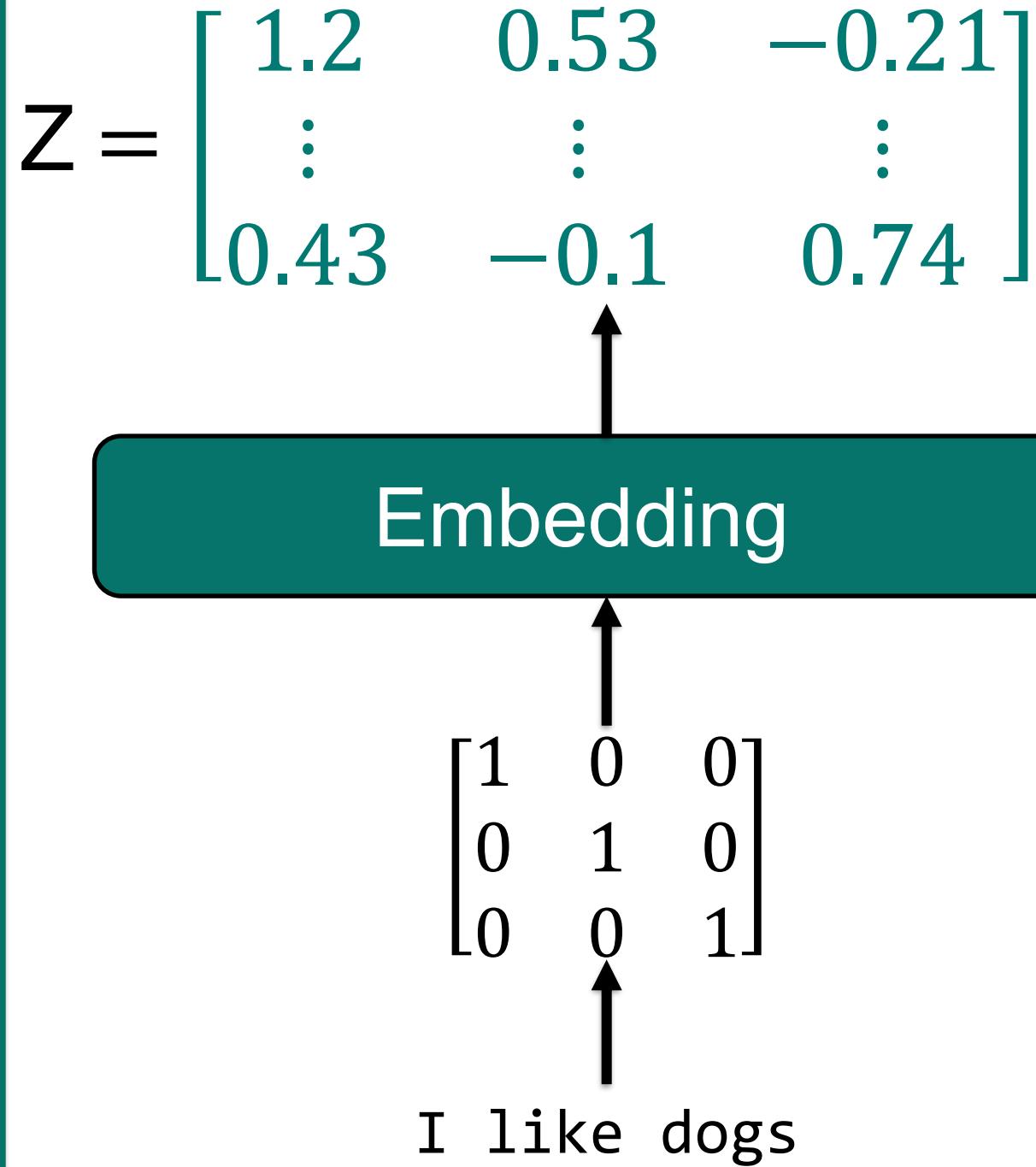
Transformers

After tokenization, data is embedded into the real numbers space.

In transformers, array of tokens $X \in [0,1]^{n \times |\mathcal{D}|}$ is multiplied by the embedding matrix $E \in \mathbb{R}^{|\mathcal{D}| \times d}$:

$$Z = XE$$

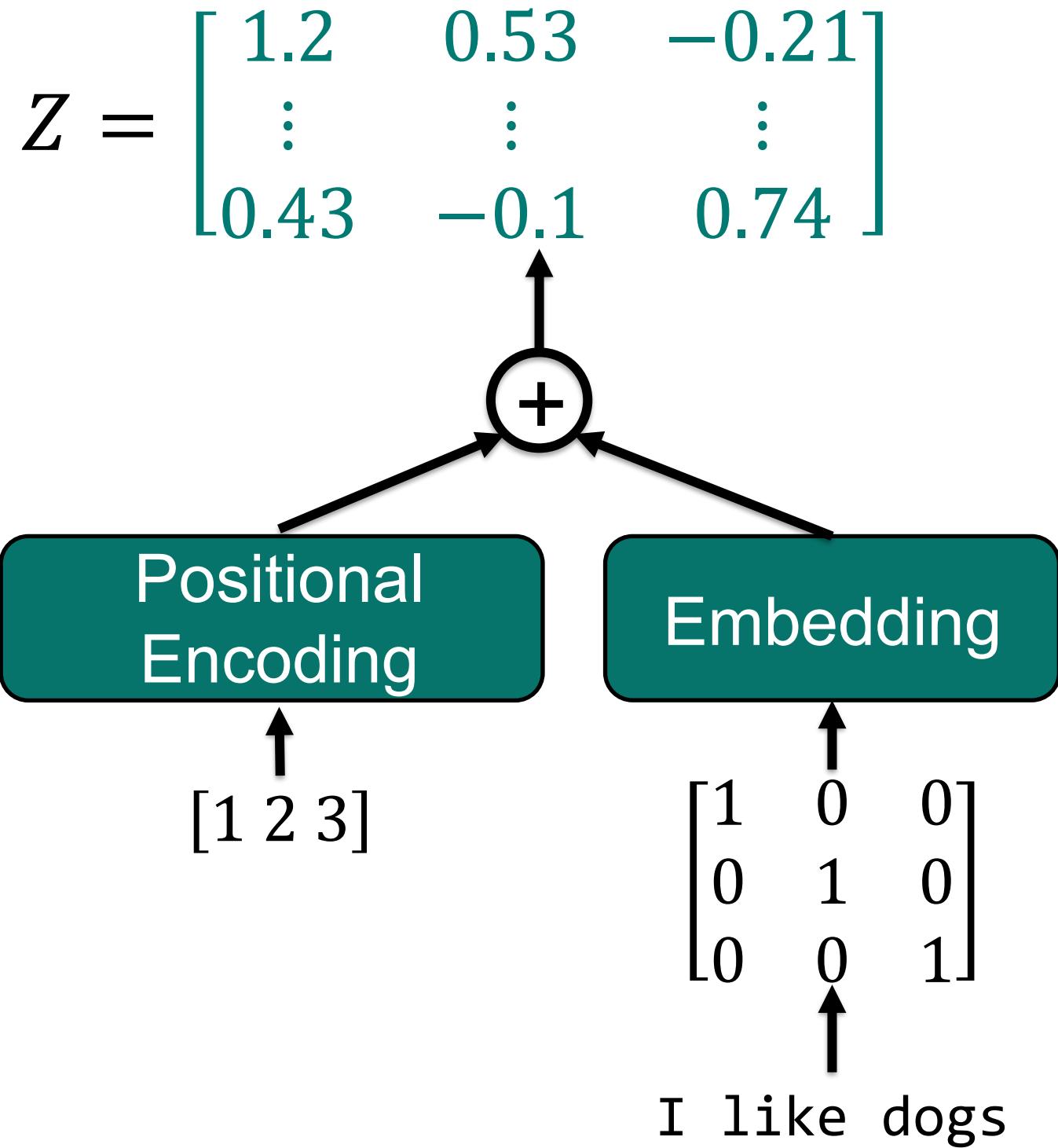
Embedding E is optimized during the training process.



Positional Encoding

Transformers are **static** – they do not store the information about the order of the input sequence x .

To solve this problem, we need to add information about the position of a given token with **positional encoding**.



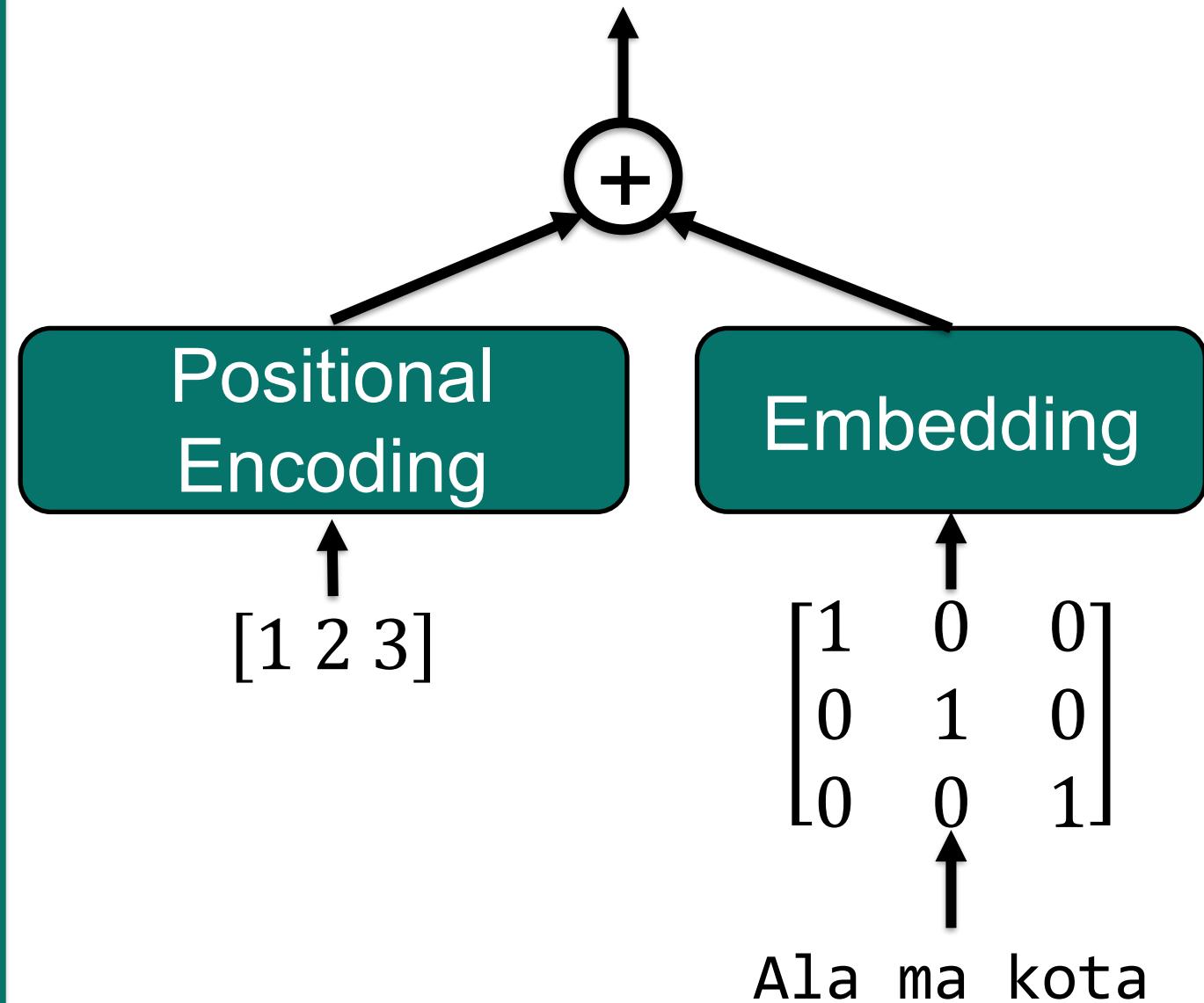
Positional Encoding

Good positional encoding must meet 2 conditions:

- Its value should be relatively small (e.g., between 0 and 1).
- Encoding should be independent of the length of the input sequence.

Vaswani, et al., 2017 proposed positional encoding method based on trigonometric functions: **sine** and **cosine**.

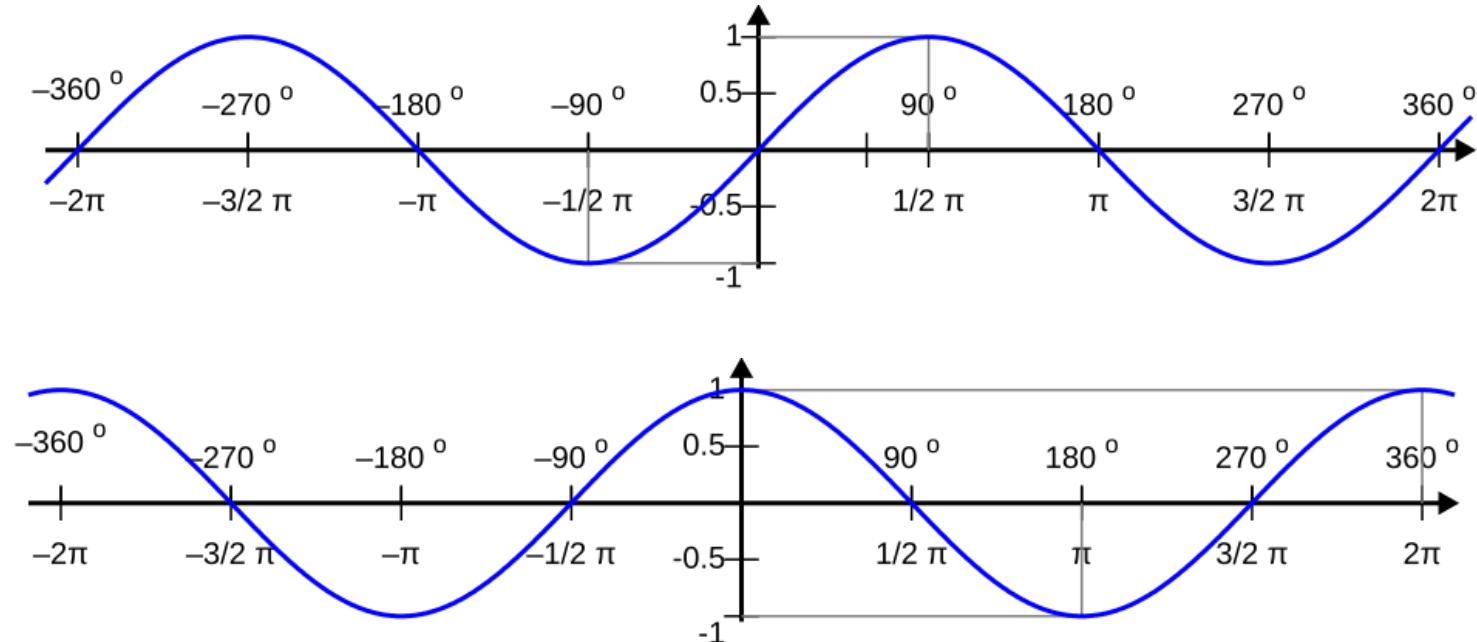
$$Z = \begin{bmatrix} 1.2 & 0.53 & -0.21 \\ \vdots & \vdots & \vdots \\ 0.43 & -0.1 & 0.74 \end{bmatrix}$$



Positional Encoding

Vaswani, et al., 2017 proposed positional encoding method based on trigonometric functions: **sine** and **cosine**.

To ensure that each index is uniquely encoded, functions with different periods are used.



Source: https://pl.wikipedia.org/wiki/Funkcje_trigonometryczne

Positional Encoding

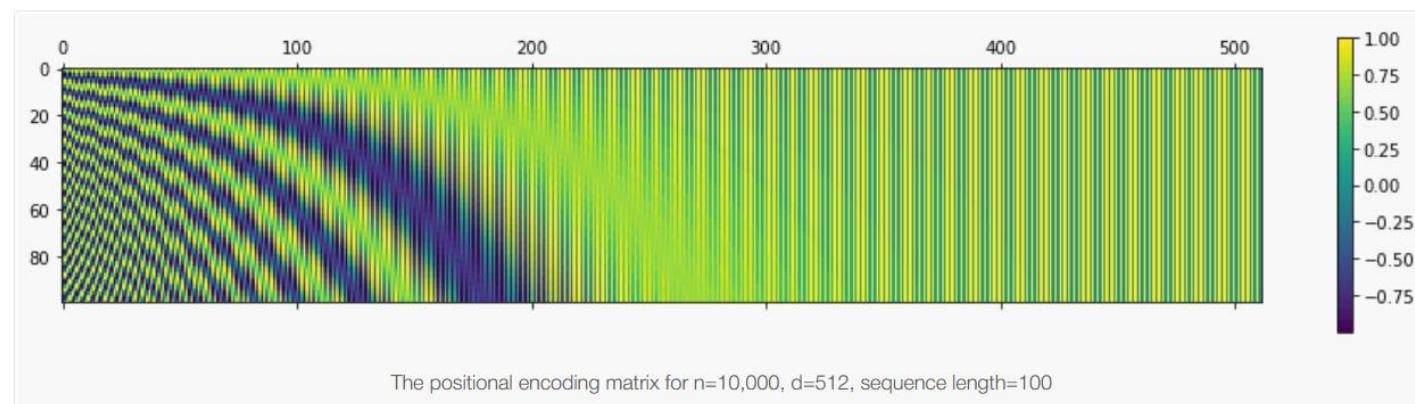
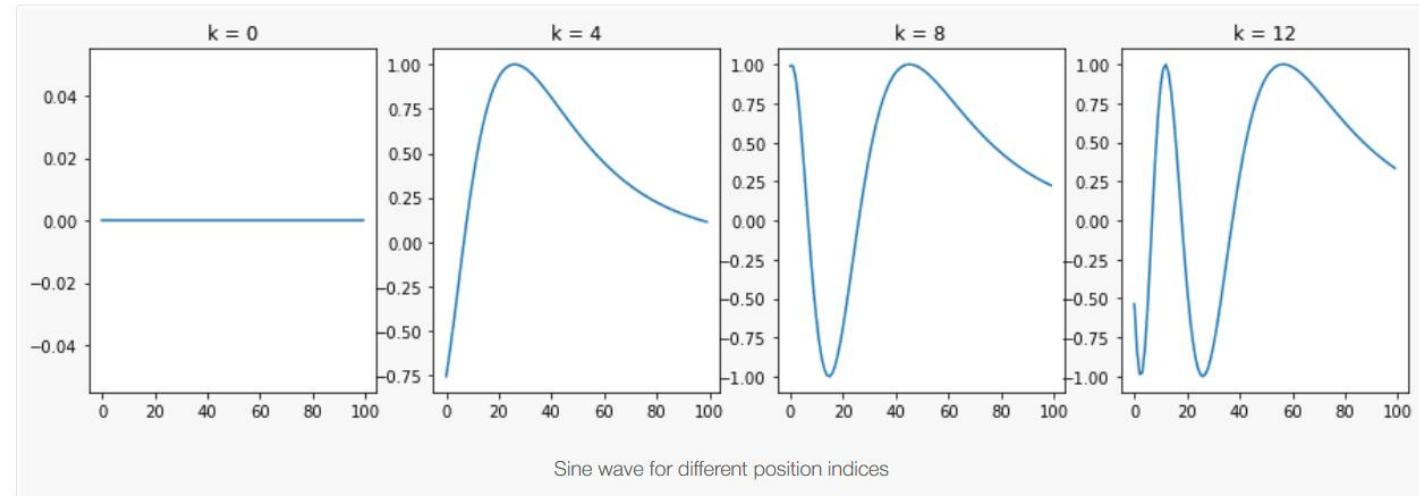
To ensure that each index is uniquely encoded, functions with different periods are used.

Let us define the positional encoding as a matrix $P_E = [p(i, j)]$, such that for each $i = 0, 1, 2, \dots, n$ and $j = 0, 1, 2, \dots, d/2$:

$$p(i, 2j) = \sin\left(\frac{i}{c^{2j/d}}\right)$$

$$p(i, 2j + 1) = \cos\left(\frac{i}{c^{2j/d}}\right)$$

where $c \in \mathbb{R}$ is constant, e.g. 10 000.



Source: <https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/>

Positional Encoding

$$\begin{bmatrix} I \\ Like \\ Dogs \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} \sin \frac{0}{2*0} & \cos \frac{0}{2*0} & \sin \frac{0}{2*1} & \cos \frac{0}{2*1} \\ \sin \frac{1}{2*0} & \cos \frac{1}{2*0} & \sin \frac{1}{2*1} & \cos \frac{1}{2*1} \\ \sin \frac{2}{2*0} & \cos \frac{2}{2*0} & \sin \frac{2}{2*1} & \cos \frac{2}{2*1} \\ & & \sin \frac{10}{2*0} & \cos \frac{10}{2*0} \\ & & \sin \frac{10}{2*1} & \cos \frac{10}{2*1} \end{bmatrix}$$

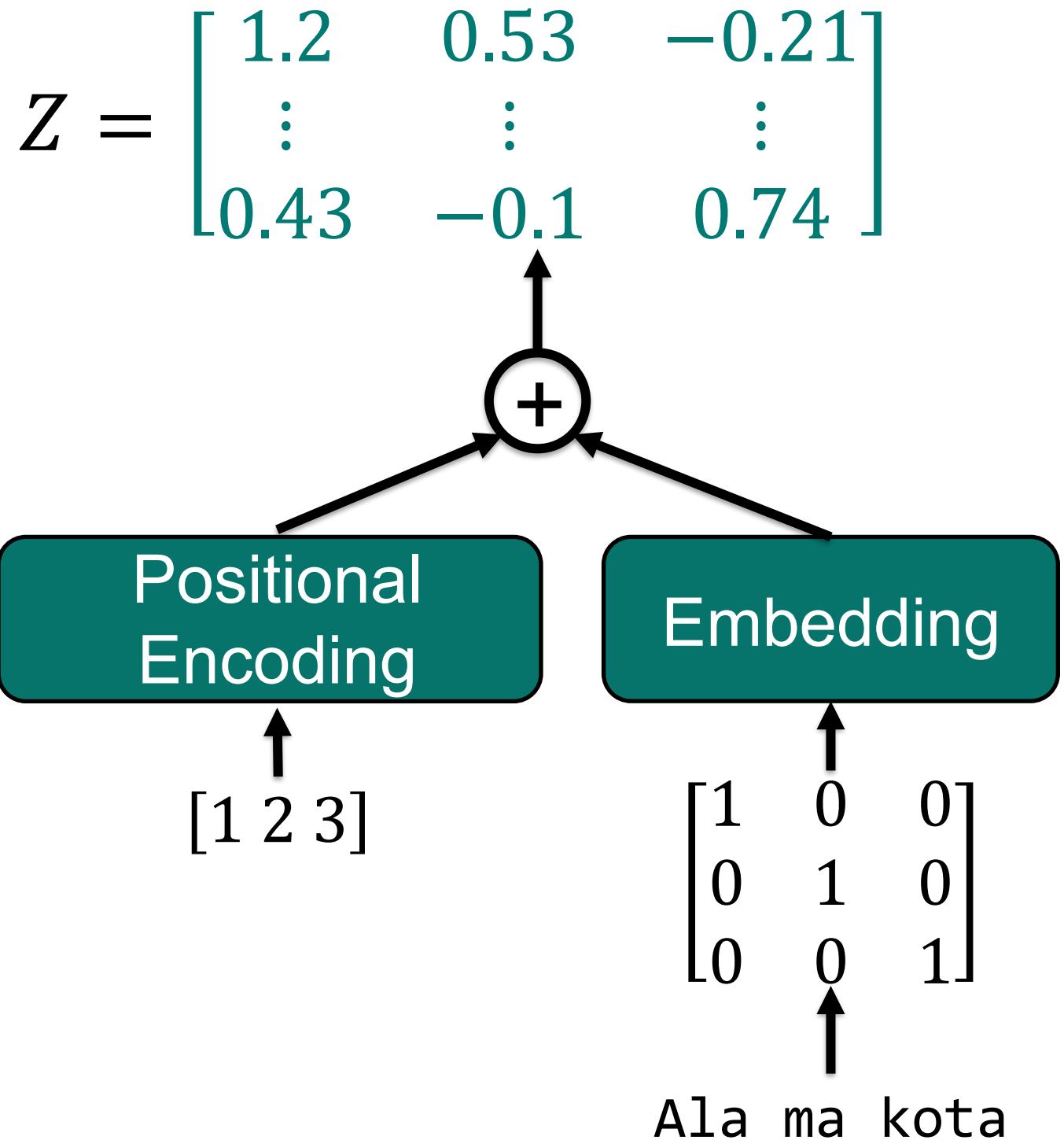
Positional Encoding

Encoded positions of words in sequences are simply added to the embedded representation of tokens:

$$\mathbf{Z} = \mathbf{X}\mathbf{E} + \mathbf{P}_E$$

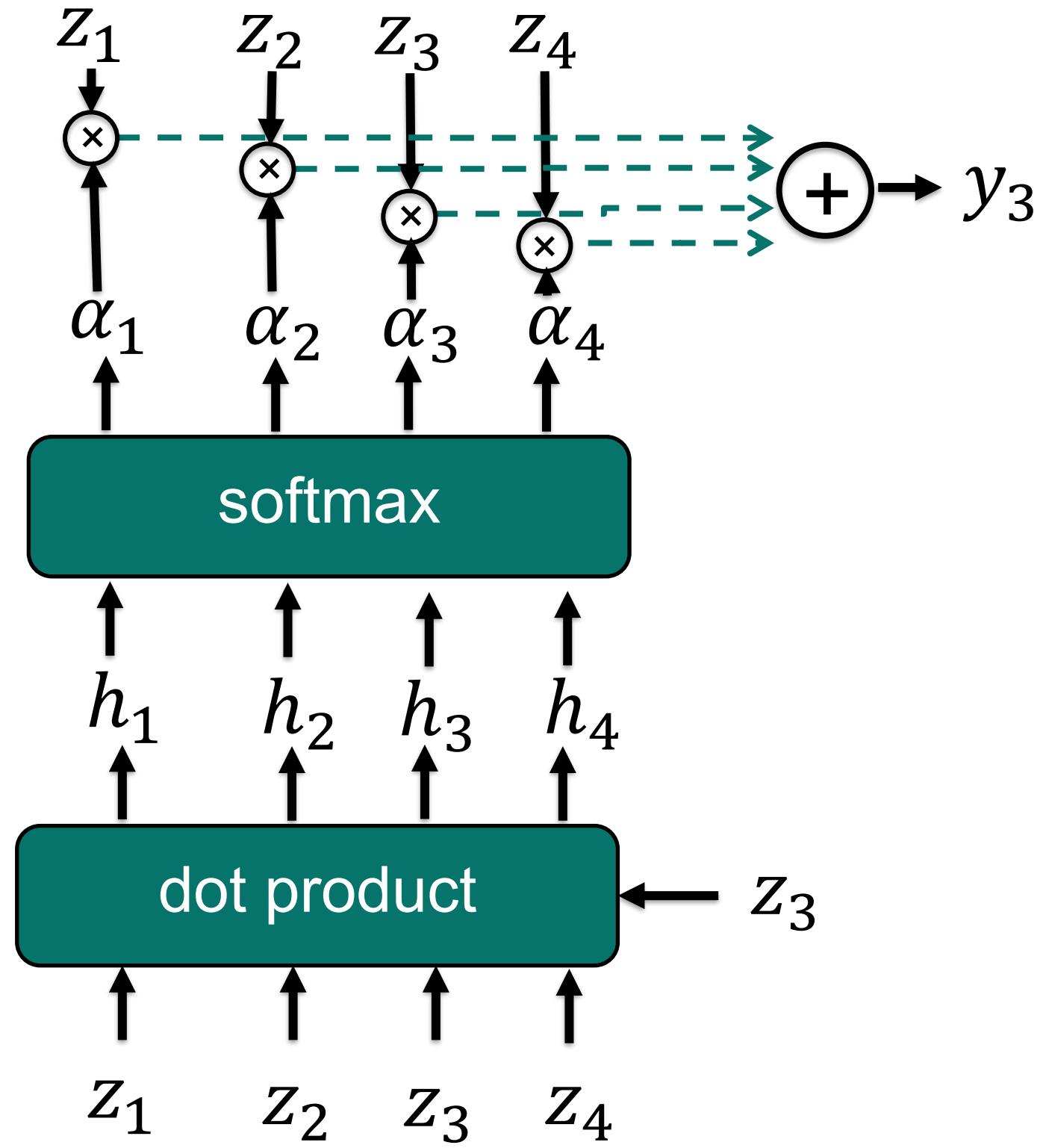
Alternative algorithms:

- Relative Positional Embedding
- Alibi
- Rotary Position Embedding (RoPE)



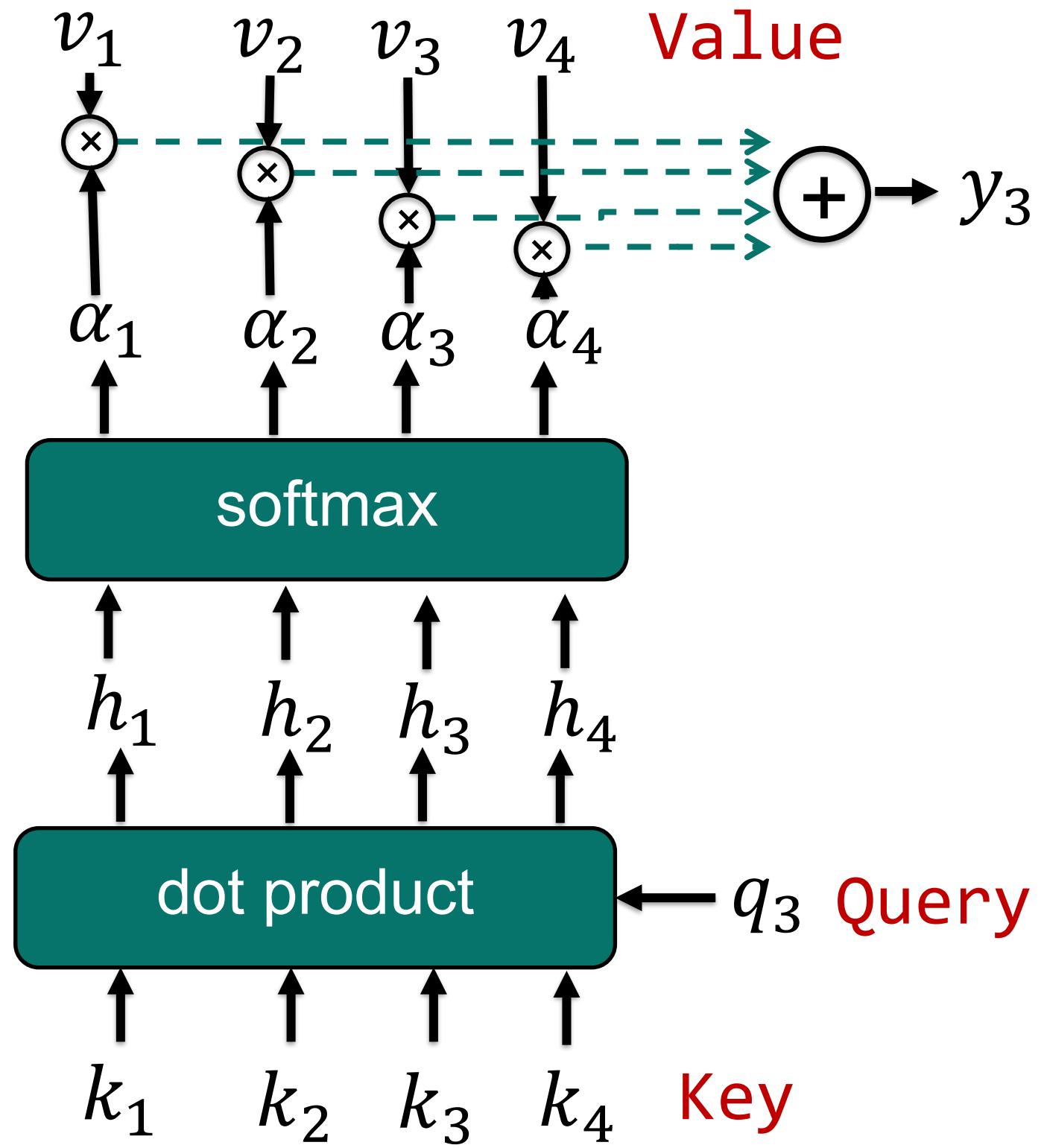
Transformers

Static attention mechanism:



Transformers

By analogy with databases:



Transformers

Vectors **key**, **query** i **value** are obtained by multiplying vector z_i by the given matrices:

$$q_i = z_i \theta_q$$

$$k_i = z_i \theta_k$$

$$v_i = z_i \theta_v$$

where $\theta_q, \theta_k, \theta_v$ are trainable weights.

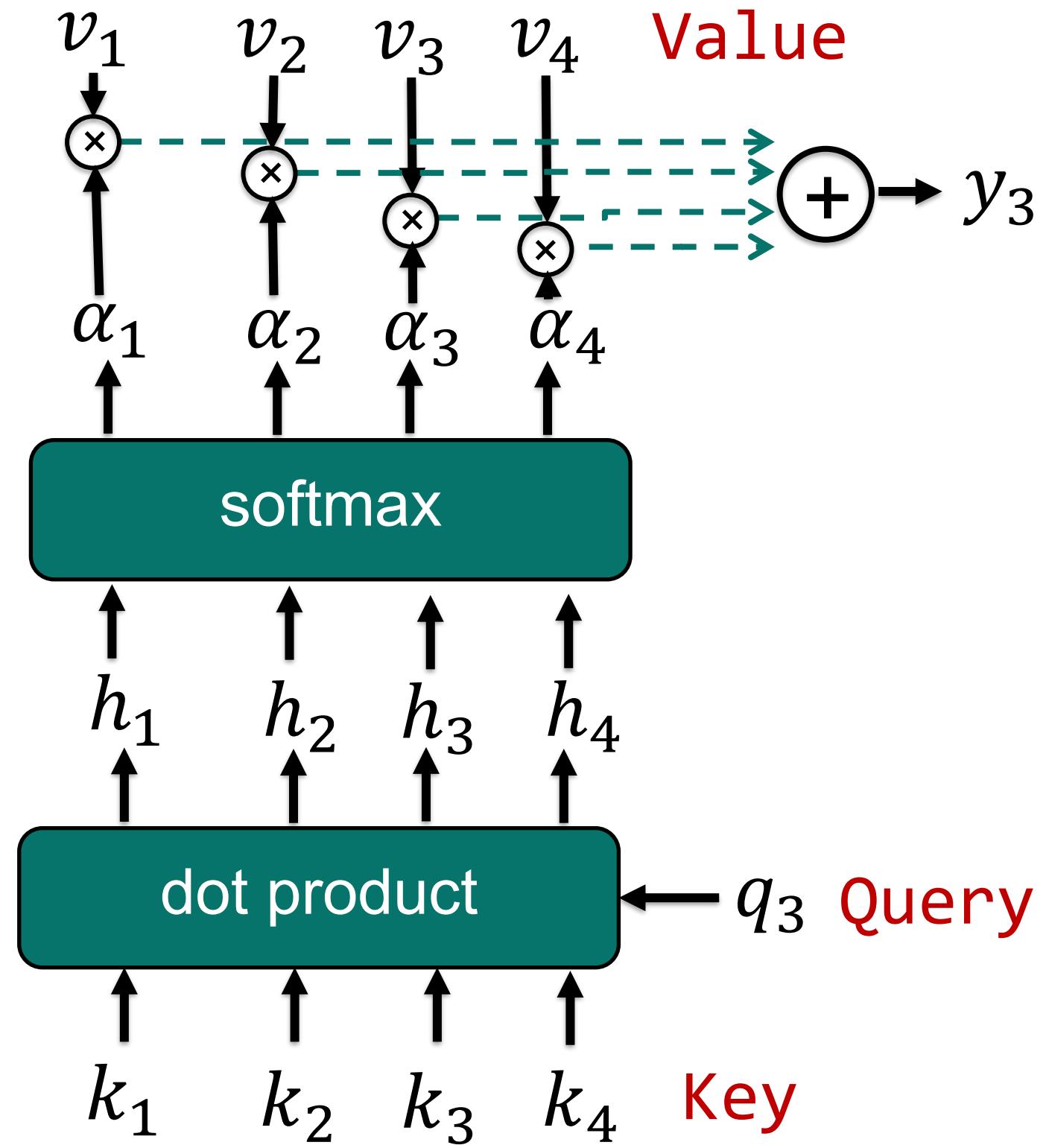
In a matrix form:

$$Q = Z\theta_q$$

$$K = Z\theta_k$$

$$V = Z\theta_v$$

where $Q \in \mathbb{R}^{m \times d_k}$, $K \in \mathbb{R}^{n \times d_k}$, $V \in \mathbb{R}^{n \times d_v}$.



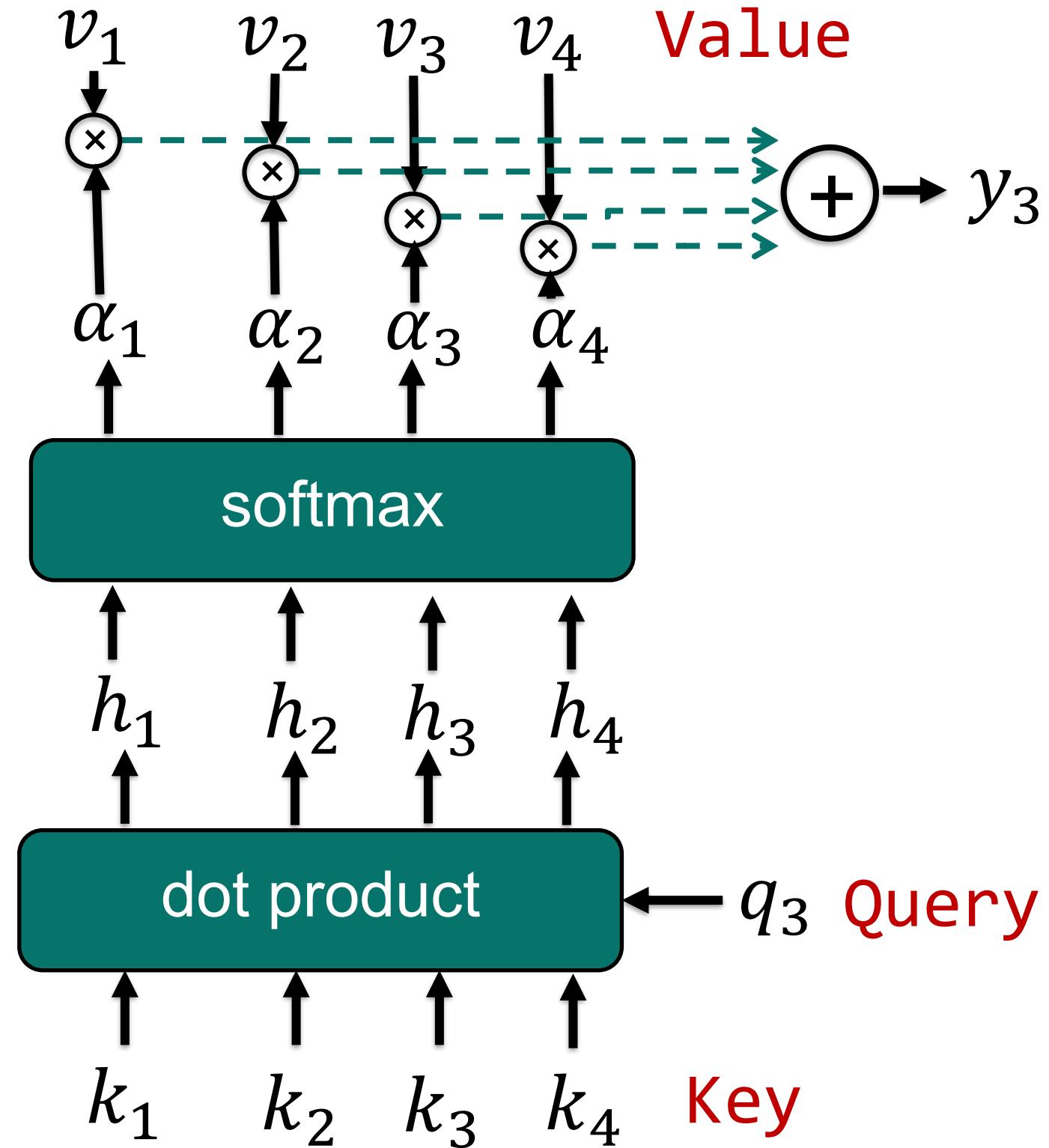
Transformers

In the case of **self-attention mechanism** is the same for all three matrices:

$$\mathbf{z}_q = \mathbf{z}_k = \mathbf{z}_v = \mathbf{z}$$

In the case of the **attention mechanism**, **Query** matrix is different than **Key** and **Value**:

$$\mathbf{z}_q \neq \mathbf{z}_k = \mathbf{z}_v$$



Transformers

- Attention is computed as follows:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}$$

where $\frac{1}{\sqrt{d_k}}$ is a scaling factor, which aims to reduce the chance of a vanishing gradient.

- To ensure that the autoregressive model does not “look ahead” mask is added to the attention:

$$\text{MaskedAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}$$

where \mathbf{M} is equal to:

$$\mathbf{M} = \begin{bmatrix} 0 & -\infty & -\infty & \cdots & -\infty \\ 0 & 0 & -\infty & \cdots & -\infty \\ 0 & 0 & 0 & \cdots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Multi-head Attention

Multi-head Attention is the most popular form of the attention mechanism.

It is a form of an *ensemble learning* – multiple attention heads are trained simultaneously.
Formally:

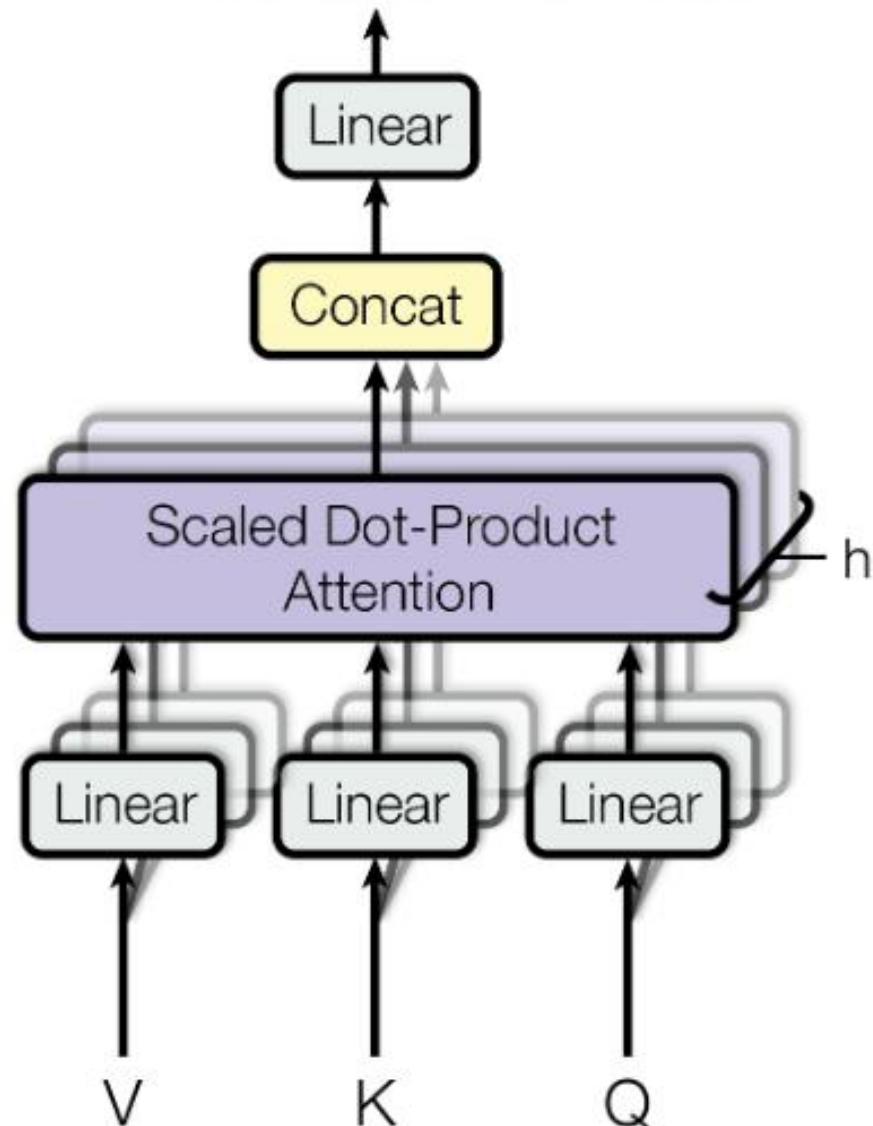
$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) =$$

$$[\text{head}_1; \dots; \text{head}_\ell] \theta_o$$

where $\theta_o \in \mathbb{R}^{\ell d_v \times n}$, is a trainable weight matrix, and head_i represents i th attention head:

$$\text{head}_i = \text{Attention}(\mathbf{Q}\theta_q^{(i)}, \mathbf{K}\theta_k^{(i)}\mathbf{V}\theta_q^{(i)})$$

Multi-Head Attention

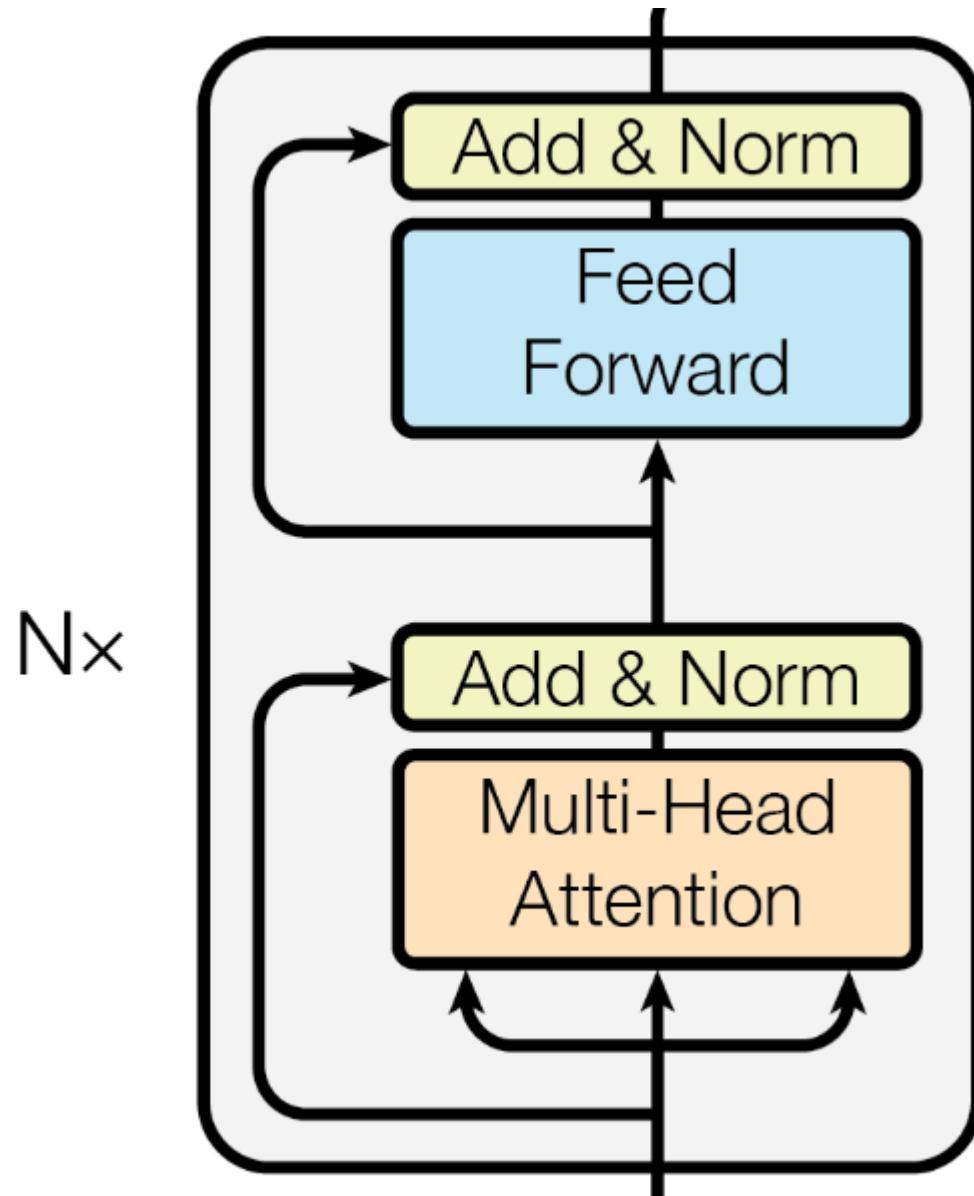


Source: Ashish Vaswani et al.. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

Transformers

In addition to the attention mechanism, the transformer consists of two additional sublayers:

- **Normalization :**
 $\text{LayerNorm}(y + \text{Sublayer}(y))$
- **Multi Layer Perceptron**

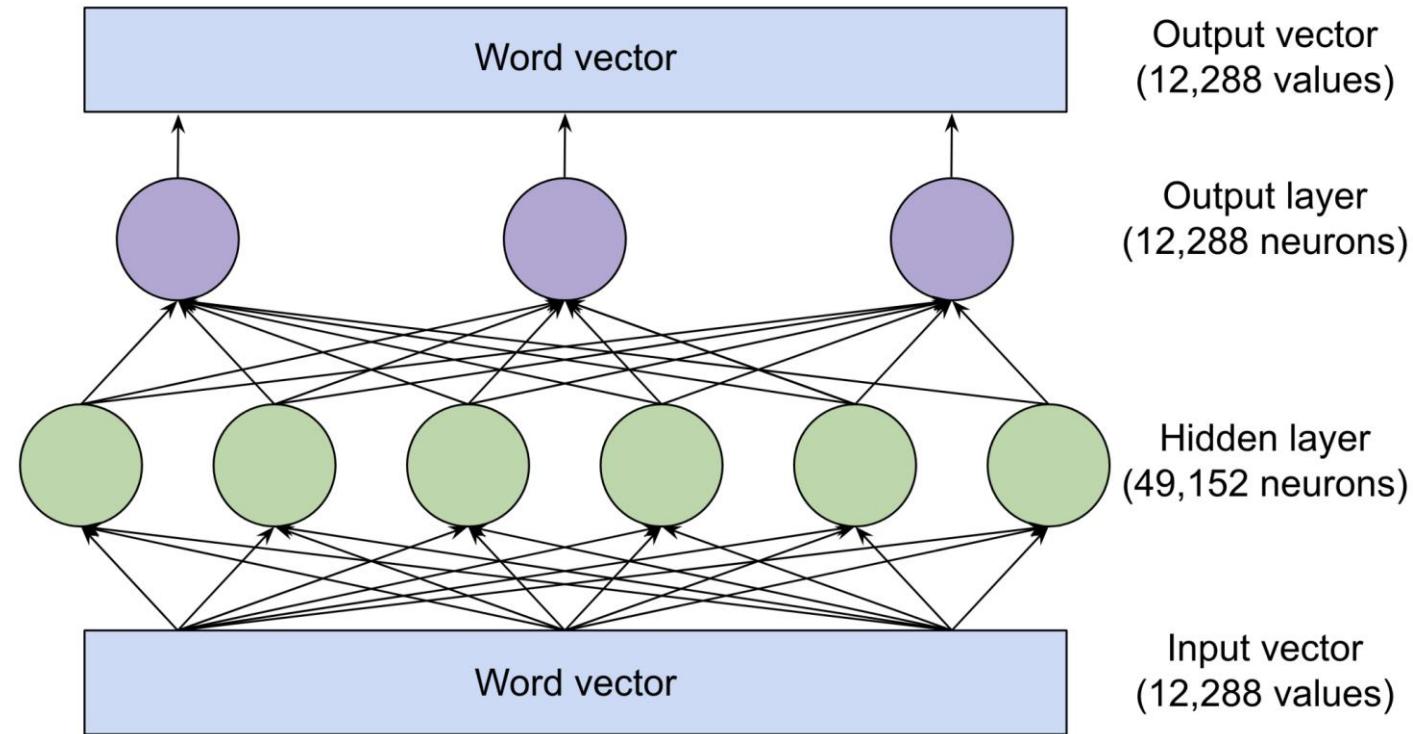


Source: Ashish Vaswani et al.. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

Transformers

In addition to the attention mechanism, the transformer consists of two additional sublayers:

- **Normalization :**
LayerNorm($y + \text{Sublayer}(y)$)
- **Multi Layer Perceptron**
 $\text{FFN}(y) = f_h(y\theta_h + b_h)\theta_f + b_f$
where $\theta_h, b_h, \theta_f, b_f$ are trainable weights, and f_h is an activation function, usually **rectifier**.

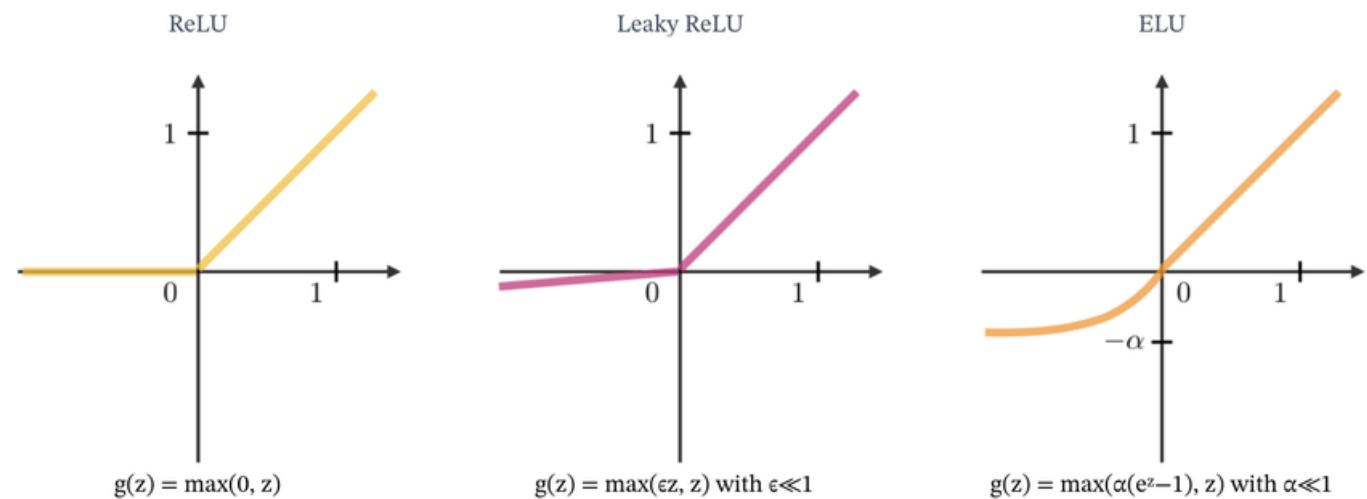


Source: <https://arstechnica.com/science/2023/07/a-jargon-free-explanation-of-how-ai-large-language-models-work/>

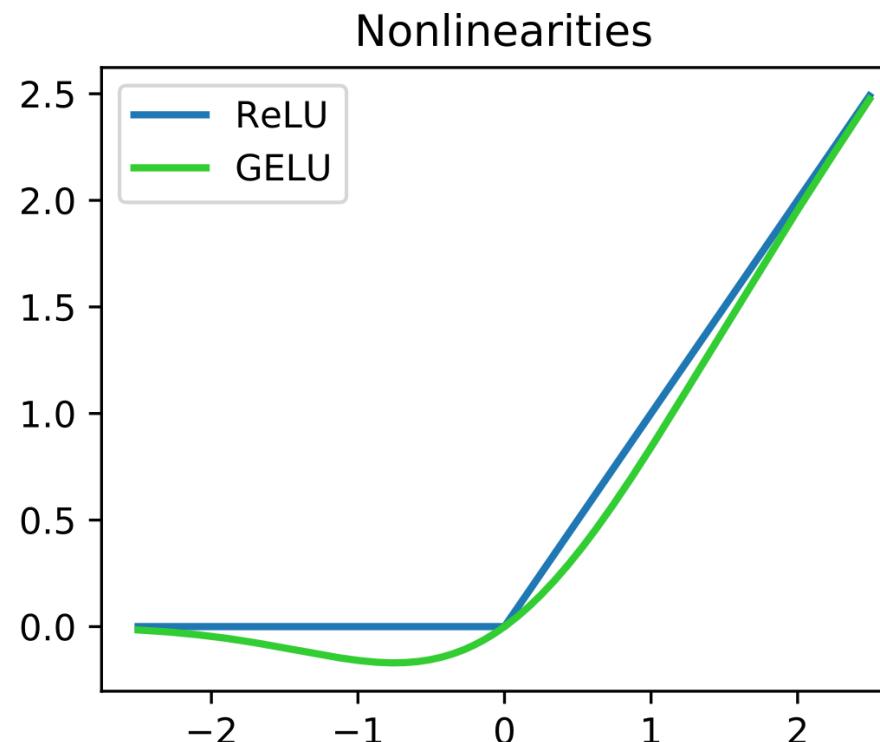
Transformers

In addition to the attention mechanism, the transformer consists of two additional sublayers:

- **Normalization :**
LayerNorm($y + \text{Sublayer}(y)$)
- **Multi Layer Perceptron**
 $\text{FFN}(y) = f_h(y\theta_h + b_h)\theta_f + b_f$
where $\theta_h, b_h, \theta_f, b_f$ are trainable weights, and f_h is an activation function, usually **rectifier**.



Source: Meseguer-Brocal, Gabriel. (2021). MULTIMODAL ANALYSIS: Informed content estimation and audio source separation. 10.48550/arXiv.2104.13276.



Source:
[https://en.wikipedia.org/wiki/Rectifier_\(neural_networks\)](https://en.wikipedia.org/wiki/Rectifier_(neural_networks))

Transformers

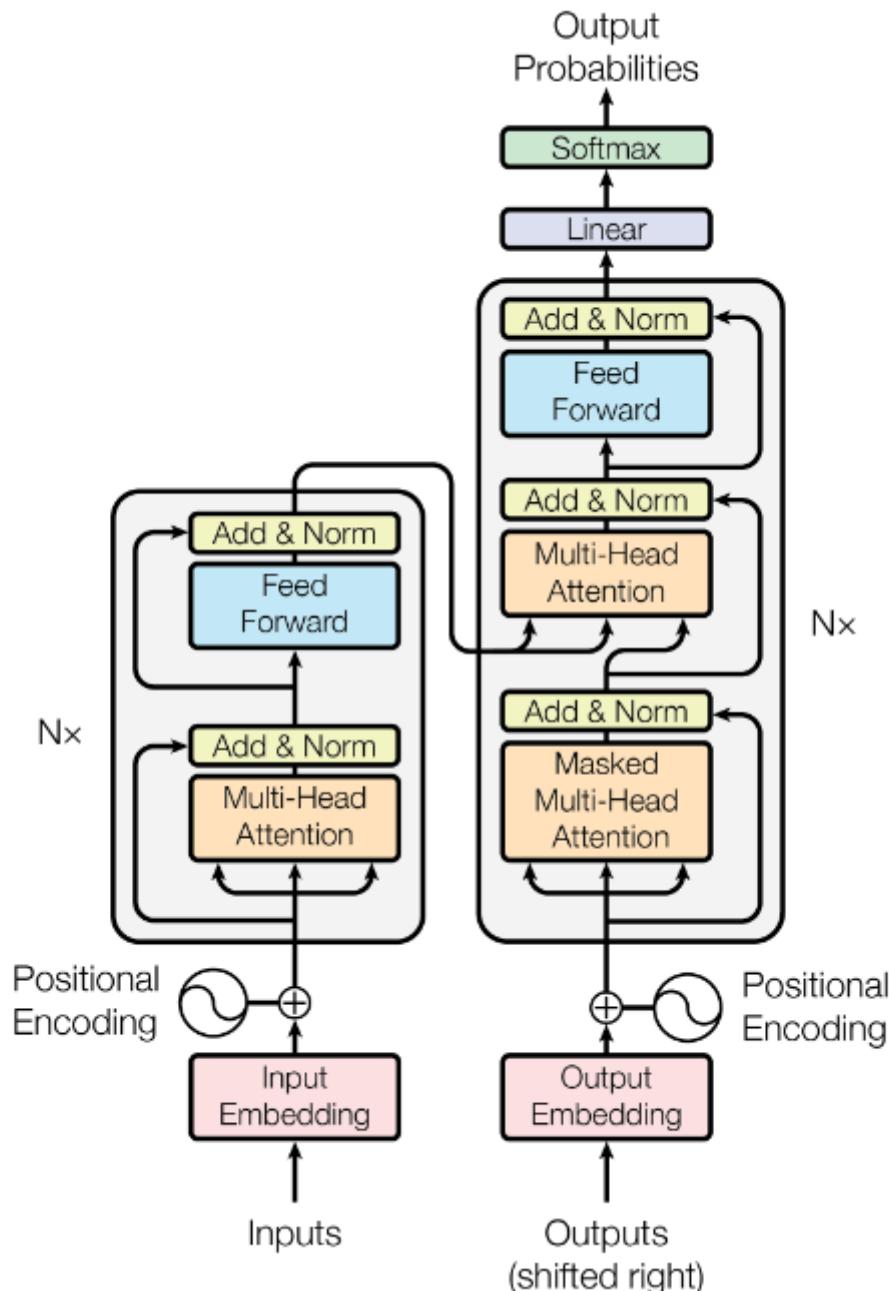
In the most common case Transformer is trained in a **semi-supervised** manner. For every n -element sequence

$x = x_1, x_2, \dots, x_n$, such that $x \in \mathcal{D}$, the model is trained to correctly predict the next token in the sequence:

$$\mathcal{L}(x; \theta) = \sum_{i=1}^m P(x_i | x_0, x_1, \dots, x_{i-1})$$

We want to find weights that maximize the following expression:

$$\theta^* = \operatorname{argmax}_{\theta} \sum_{x \in \mathcal{D}} \mathcal{L}(x; \theta)$$



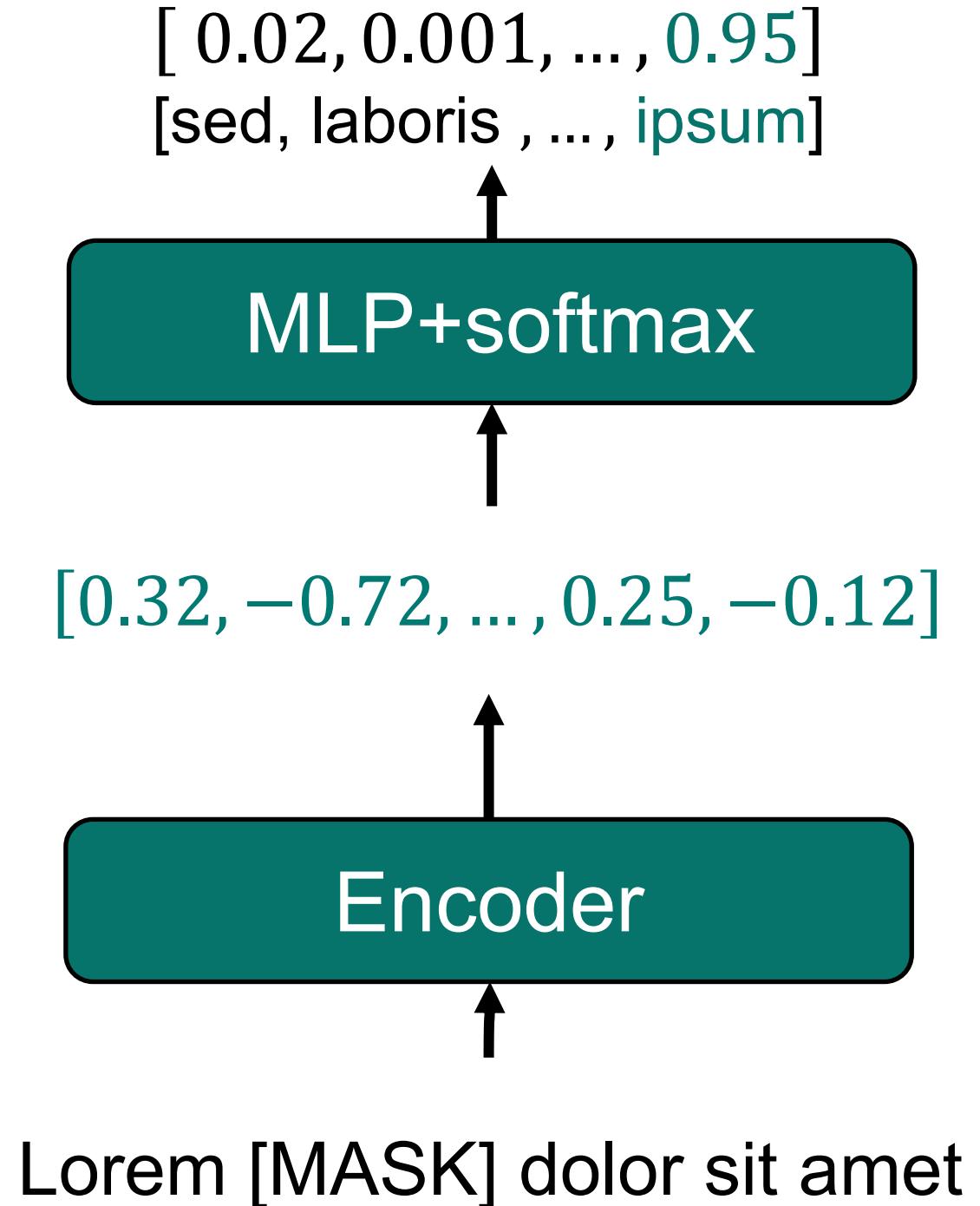
Source: Ashish Vaswani et al.. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

Large Language Models

- **Large Language Models** can be divided into three categories:
 - Encoder only
 - Decoder-Encoder
 - Decoder only

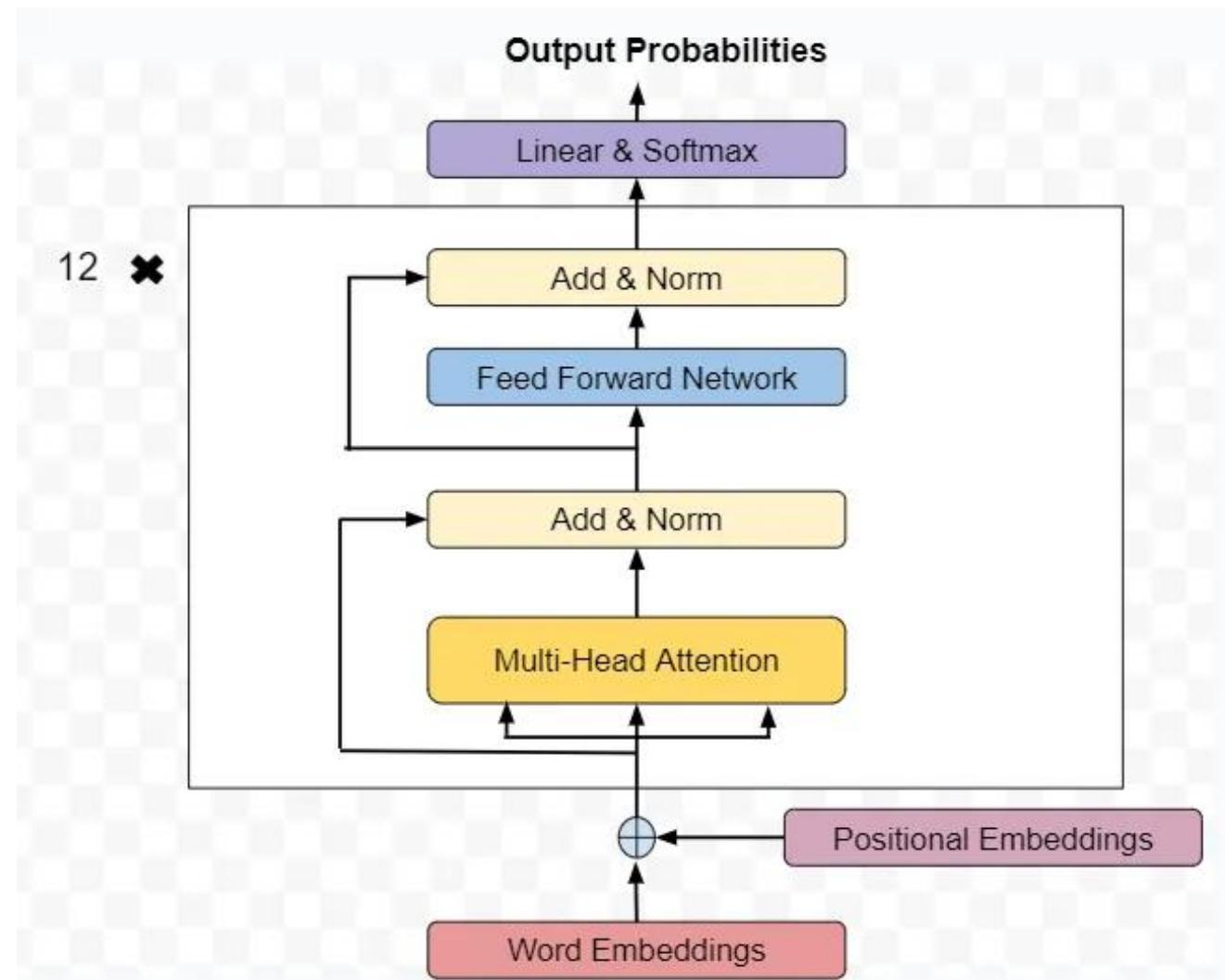
Encoder Only

- Oldest type of LLMs.
- Used primarily as embedding of text data for further processing.
- Encoder only model can be used directly (**transfer learning**) or trained further for a specific task (**fine-tuning**).
- Commonly used for:
 - Sentiment analysis
 - Document classification
 - Data extraction



Encoder Only

- Examples:
 - ELMo
 - **BERT**
 - RoBERTa
 - ALBERT
 - HERBERT
 - XLNet



Source: <https://pub.towardsai.net/bert-in-depth-exploration-of-architecture-workflow-code-and-mathematical-foundations-0c67ad24725b>

Encoder Only

- During training, a sentence is represented as a conditional probability $P(x_i|x_0, x_1, \dots, x_{i-1})$, e.g.:

$$P(\text{Dogs}|I, \text{Like})$$

- As we already know, such a model is unable to learn non-trivial, semantic and syntactic, relationships happening in natural language.

Encoder Only

- Alternatively, one might use **masks** hiding individual words (tokens), which forces the model to predict missing words, based on the given sentence:

Tomorrow it will rain and snow.

Tomorrow it will [MASK] and [MASK].

- Such an approach is called **Masked Language Modeling**. A loss function is as follows:

$$\mathcal{L}_{MLP} = - \sum_{x \in \mathcal{X}} \sum_{i \in \mathcal{A}(x)} \log p(x_i | \hat{x})$$

where \hat{x} is a masked sequence, \mathcal{X} is a set of tokens, and $\mathcal{A}(x)$ is a set of positions of all the applied masks.

Encoder Only

- Encoders are also trained for **Next Sentence Prediction** task:

$s_1 = \text{Tomorrow it will rain [SEP] I need to take an umbrella [SEP]}$

$s_2 = \text{Tomorrow it will rain [SEP] I need to buy bread [SEP]}$

- A target function is a maximum likelihood estimator of the probability that sentence 2 follows from sentence 1:

$$\mathcal{L}_{NSP} = -\log p(c|s)$$

where s is two sentence sequence, and $c \in \{\text{IsNext}, \text{NotNext}\}$ is a correct label.

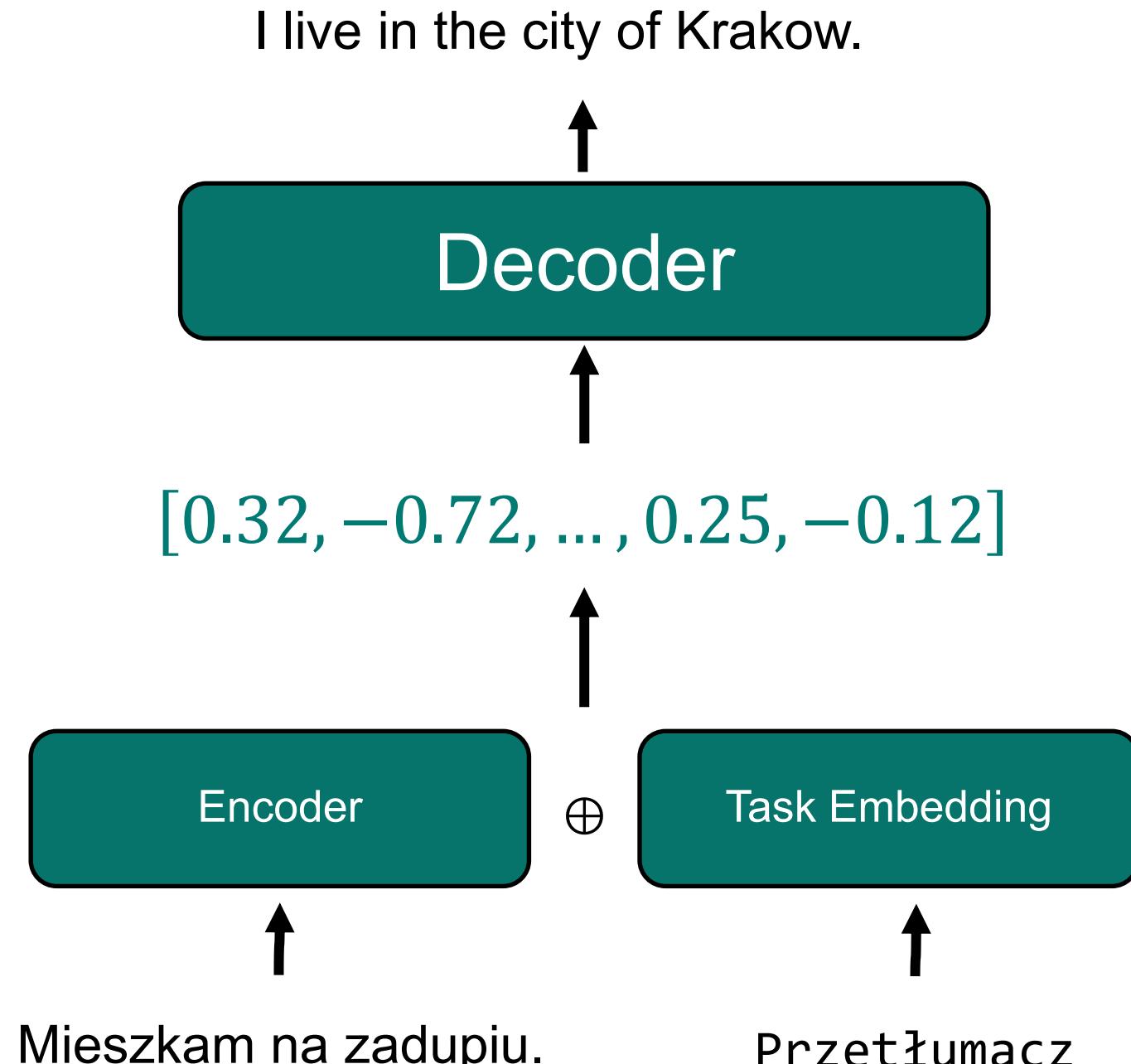
- Losses simply adds up:

$$\mathcal{L} = \mathcal{L}_{MLP} + \mathcal{L}_{NSP}$$

Encoder-Decoder

text-to-text models are the broader family of Large Language Models.

(Raffel et al., 2020) have proven that all the most common NLP tasks are solvable by such a model:



Source: <https://buzz.gazeta.pl/buzz/7,156947,23495711,mieszkam-na-zadupiu-to-mieszkam-w-krakowie-google-zaliczylo.html>

Encoder-Decoder

text-to-text models are the broader family of Large Language Models.

(Raffel et al., 2020) have proven that all the most common NLP tasks are solvable by such a model:

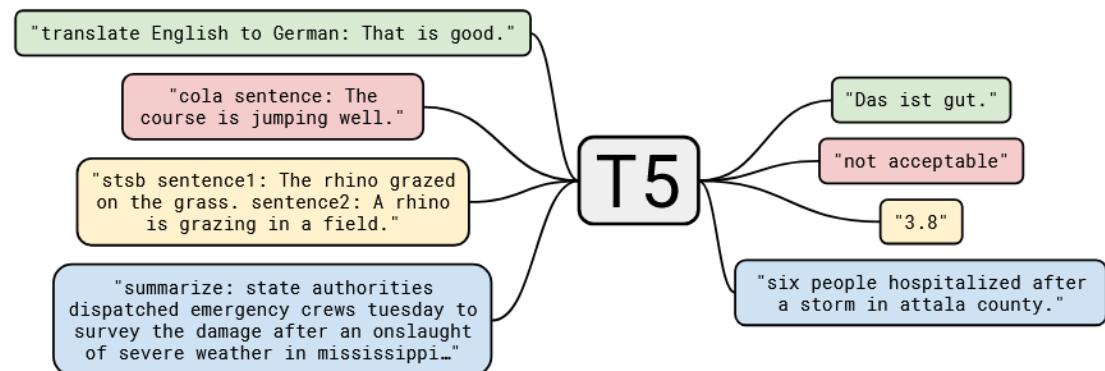


Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. “T5” refers to our model, which we dub the “Text-to-Text Transfer Transformer”.

Source: Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140), 1–67.
<https://jmlr.org/papers/volume21/20-074/20-074.pdf>

Encoder-Decoder

text-to-text models are the broader family of Large Language Models.

(Raffel et al., 2020) have proven that all the most common NLP tasks are solvable by such a model:

Encoder-Decoder training is a two-step procedure:

1. First, the network is trained in a semi-supervised manner, similar to Encoder models.

Method	Enc	Dec	E-D	Input	Output
Causal LM		•	•		The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷
Prefix LM		•	•	[C] The kitten is	chasing ¹ the ² ball ³ . ⁴
Masked LM	•		•	[C] The kitten [M] chasing the [M] .	_ _ is _ _ ball _
MASS-style	•		•	[C] The kitten [M] [M] [M] ball .	_ _ is chasing the _ _
BERT-style	•		•	[C] The kitten [M] playing the [M] .	_ kitten is chasing _ ball _
Permuted LM	•			[C] The kitten is chasing the ball .	The ⁵ kitten ⁷ is ⁶ chasing ¹ the ⁴ ball ² . ³
Next Sentence Prediction	•			[C] The kitten is chasing the ball . Birds eat worms .	Pr(IsNext representation-of-[C])
Sentence Comparison	•			Encode a sentence as \mathbf{h}_a and another sentence as \mathbf{h}_b	Score($\mathbf{h}_a, \mathbf{h}_b$)
Token Classification	•			[C] The kitten is chasing the ball .	Pr(· The) Pr(· kitten) ... Pr(· .)
Token Reordering			•	[C] . kitten the chasing The is ball	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷
Token Deletion			•	[C] The kitten is chasing the ball .	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷
Span Masking			•	[C] The kitten [M] is [M] .	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷
Sentinel Masking			•	[C] The kitten [X] the [Y]	[X] ¹ is ² chasing ³ [Y] ⁴ ball ⁵ . ⁶
Sentence Reordering			•	[C] The ball rolls away swiftly . The kitten is chasing the ball .	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷ The ⁸ ball ⁹ rolls ¹⁰ away ¹¹ swiftly ¹² . ¹³
Document Rotation			•	[C] chasing the ball . The ball rolls away swiftly . The kitten is	The ¹ kitten ² is ³ chasing ⁴ the ⁵ ball ⁶ . ⁷ The ⁸ ball ⁹ rolls ¹⁰ away ¹¹ swiftly ¹² . ¹³

Table 1.1: Comparison of pre-training tasks, including language modeling, masked language modeling, permuted language modeling, discriminative training, and denoising autoencoding. [C] = [CLS], [M] = [MASK], [X], [Y] = sentinel tokens. Enc, Dec and E-D indicate whether the approach can be applied to encoder-only, decoder-only, encoder-decoder models, respectively. For generation tasks, superscripts are used to represent the order of the tokens.

Source: Tong Xiao and Jingbo Zhu, 2025. Foundations of Large Language Models, <https://arxiv.org/abs/2501.09223>

Encoder-Decoder

text-to-text models are the broader family of Large Language Models.

(Raffel et al., 2020) have proven that all the most common NLP tasks are solvable by such a model:

Encoder-Decoder training is a two-step procedure:

1. First, the network is trained in a semi-supervised manner, similar to Encoder models.
2. Then, the model is tuned to solve specific tasks with proper datasets:

4.2 Tasks

SQuAD (Rajpurkar et al., 2016) a an extractive question answering task on Wikipedia paragraphs. Answers are text spans extracted from a given document context. Similar to BERT (Devlin et al., 2019), we use concatenated question and context as input to the encoder of BART, and additionally pass them to the decoder. The model includes classifiers to predict the start and end indices of each token.

MNLI (Williams et al., 2017), a bitext classification task to predict whether one sentence entails another. The fine-tuned model concatenates the two sentences with appended an EOS token, and passes them to both the BART encoder and decoder. In contrast to BERT, the representation of the EOS token is used to classify the sentences relations.

ELI5 (Fan et al., 2019), a long-form abstractive question answering dataset. Models generate answers conditioned on the concatenation of a question and supporting documents.

XSum (Narayan et al., 2018), a news summarization dataset with highly abstractive summaries.

ConvAI2 (Dinan et al., 2019), a dialogue response generation task, conditioned on context and a persona.

CNN/DM (Hermann et al., 2015), a news summarization dataset. Summaries here are typically closely related to source sentences.

Source: Mike Lewis et al.. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 7871–7880, Online. Association for Computational Linguistics.

Encoder-Decoder

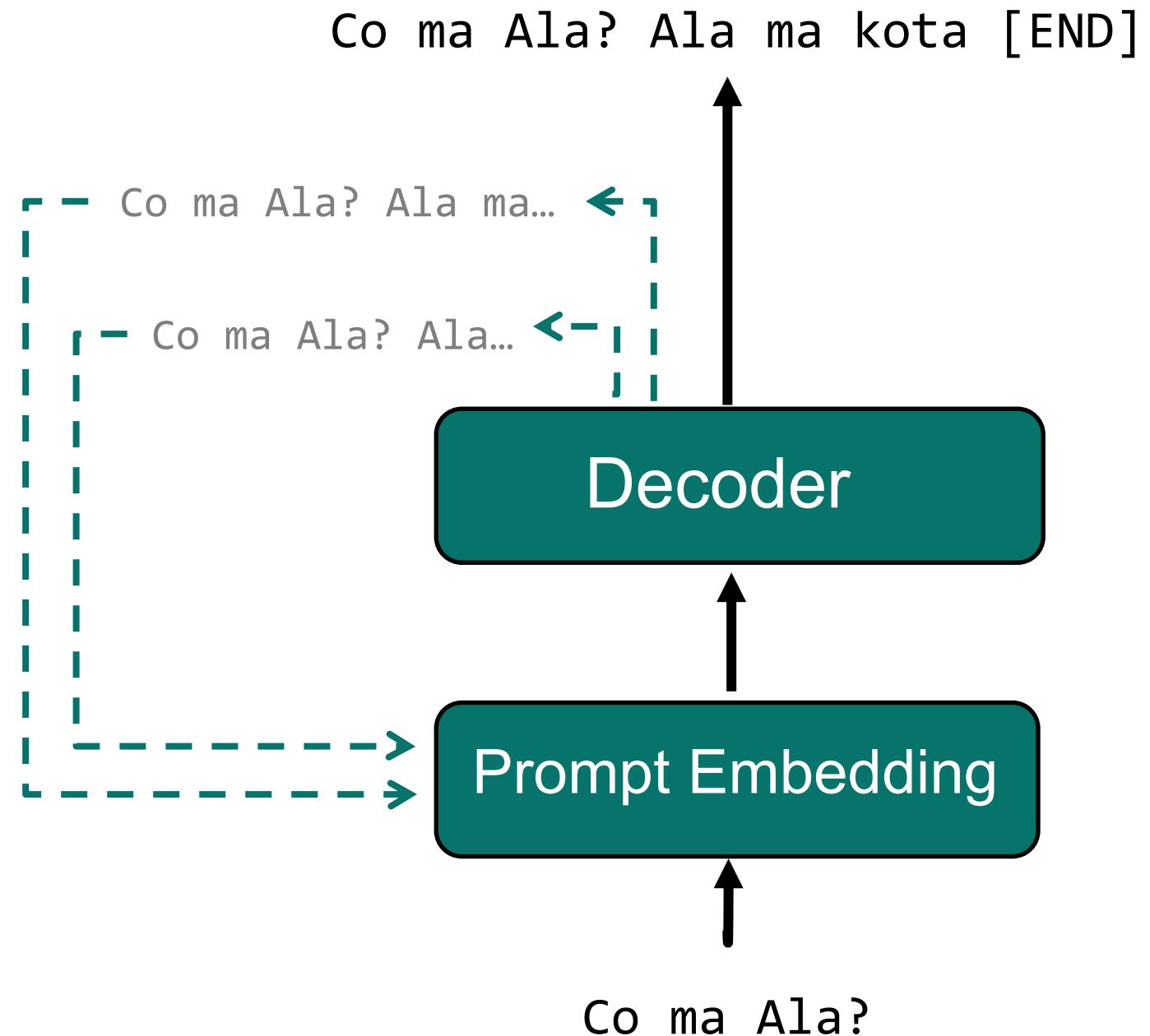
- Encoder-Decoder models are the most computationally demanding and hard-to-train class of LLM models.
 - This is due to the two-step training process and the number of tasks that the model must be taught independently.
- Encoder-Decoder models outperform other types of LLMs in tasks related to text translation and synthesis. However, they are inferior at generating new text.
- Yet still, they show some **emergent abilities** (*zero-shot, one-shot, few-shot learning*) (Fu et al., 2023).
- Examples:
 - T5
 - MT5
 - AlexaTM
 - BART

Decoder-only

The most popular class of LLM models.

Examples:

- GPT
- LLaMa
- PaLM
- DeepSeek
- Chinchilla
- Gemini
- Claude
- Grok
- ...and much more



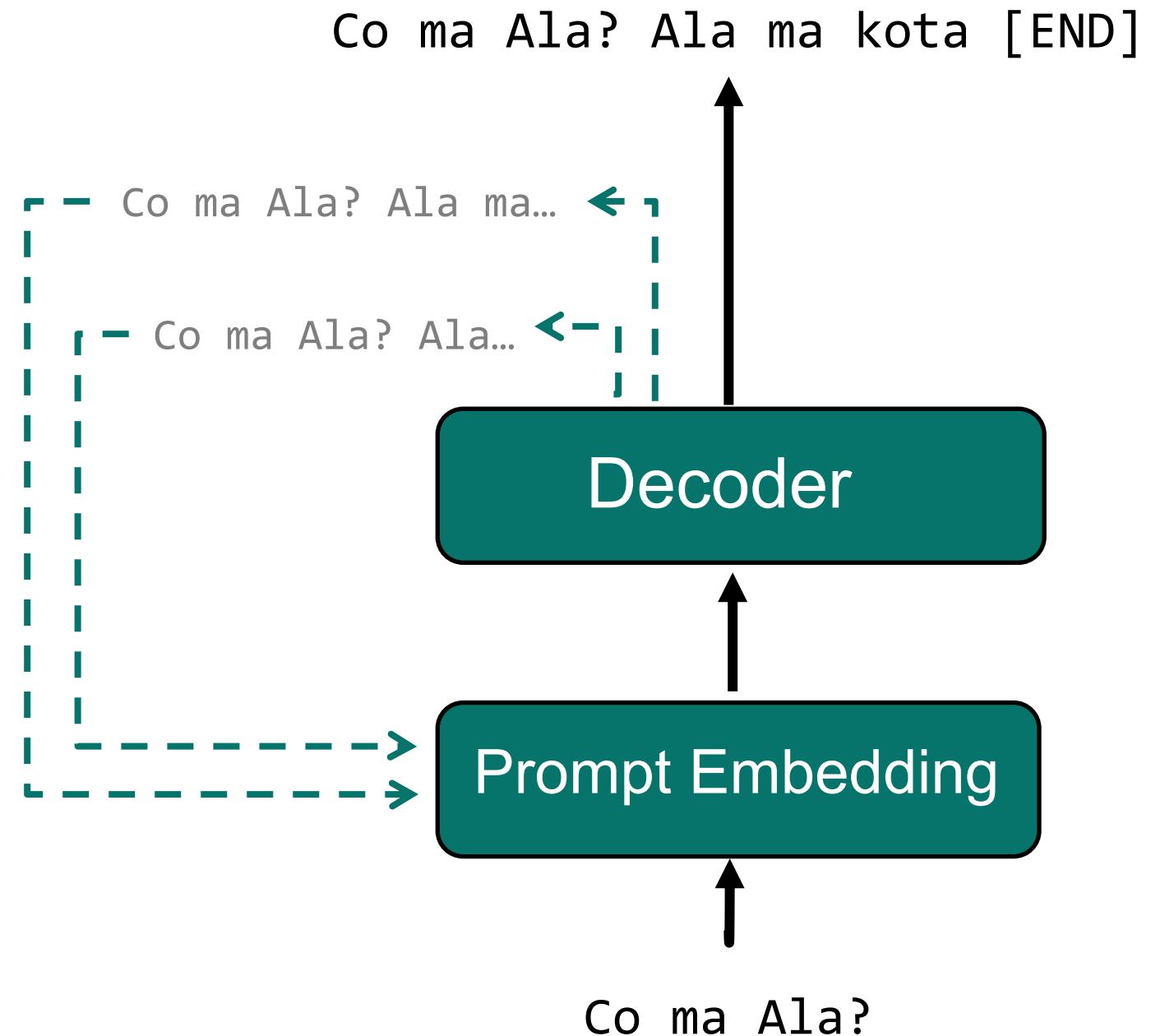
Decoder-only

Decoder only models are **non-parametric** generative models. They are trained to maximize the probability of correctly predicting the next token in the sequence:

$$\theta^* = \operatorname{argmax}_{\theta} p_{\theta}(x_i | x_0, x_1, \dots, x_{i-1})$$

This is a non-trivial task, solvable due to:

1. The architecture of the model: **transformers** store the position of the token (thanks to **positional encoding**), not a current state of the model.
2. Model is **autoregressive**.



Decoder-only

- As a result, the target of optimization is subdivided into independent sub-problems:

$$\begin{aligned}\theta^* &= \operatorname{argmax}_{\theta} p_{\theta}(x_1|x_0) \cdot p_{\theta}(x_2|x_0, x_1) \cdot \dots \cdot p_{\theta}(x_m|x_0, x_1, \dots, x_{m-1}) \\ &= \operatorname{argmax}_{\theta} \prod_{i=1}^m p_{\theta}(x_i|x_0, x_1, \dots, x_{i-1})\end{aligned}$$

- After logarithmization:

$$\theta^* = \operatorname{argmax}_{\theta} \sum_{i=1}^m \log p_{\theta}(x_i|x_0, x_1, \dots, x_{i-1})$$

Decoder-only

- Training is straightforward; the model is trained for the **next token prediction** task, which means that the loss function is just a **cross-entropy** between the output of the model and the next token in a sequence. For example:

$$\hat{x} = \operatorname{argmax}_{x \in \mathcal{X}} G_{\theta}(x | I, Like)$$

$$\mathcal{L}(\hat{x}, Dogs) = \text{LogLoss}(\hat{x}, Dogs)$$

- It is also possible to **fine-tune** Decoder-only model for the specific task.

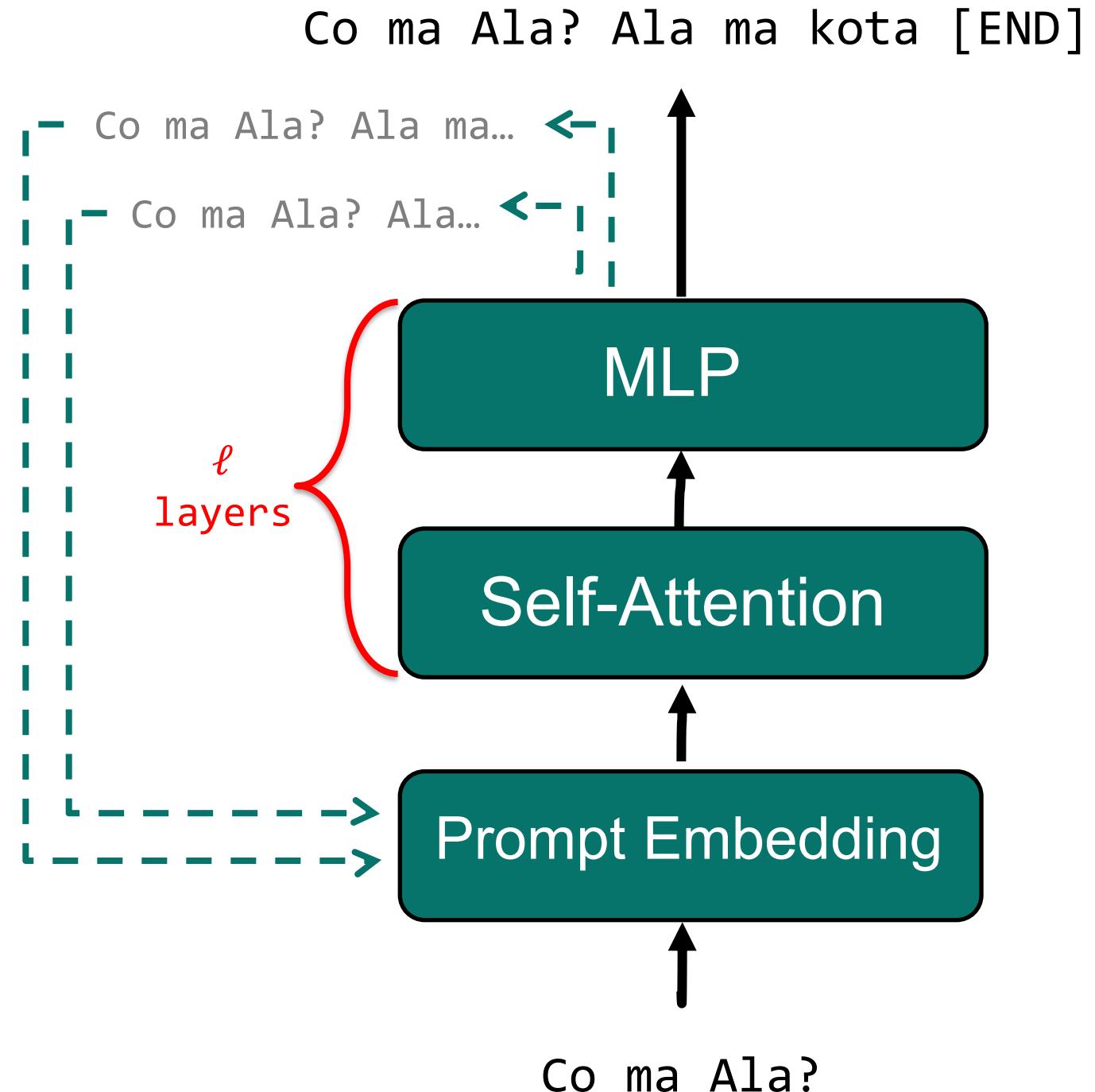
Decoder-only

Decoder-only architecture is based on ℓ consecutive **transformer** layers.

Experiments (Geva et al, 2020, Merullo et al. 2023) show that Decoder-only models possess some unique properties.

LLMs show “division of roles” between sub-layers:

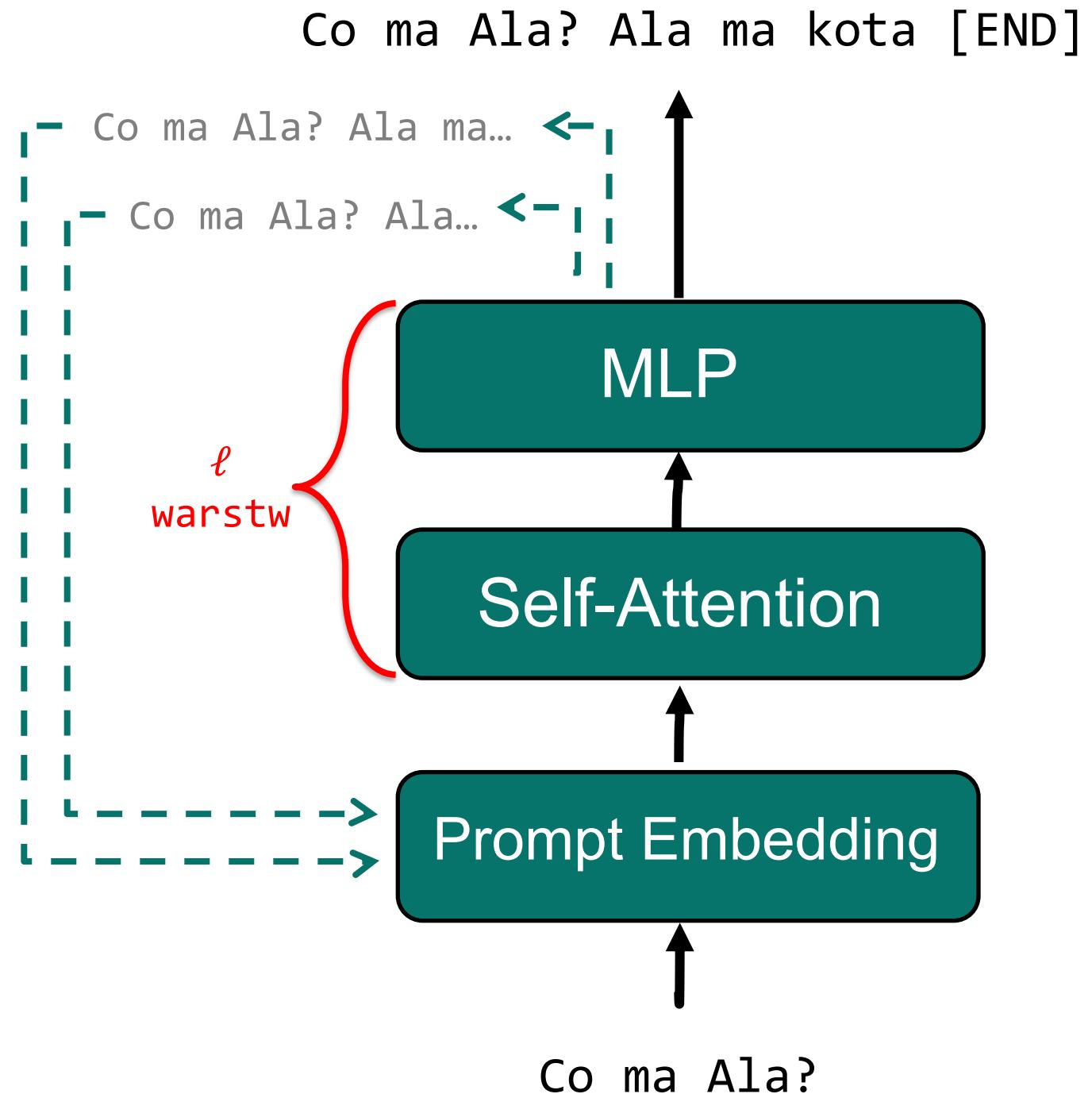
1. **Self-attention mechanisms** learns syntactic relations between words and extract the meaning of a sentence.
2. MLP layers store the information extracted from the data, which is crucial for generative tasks.



Decoder-only

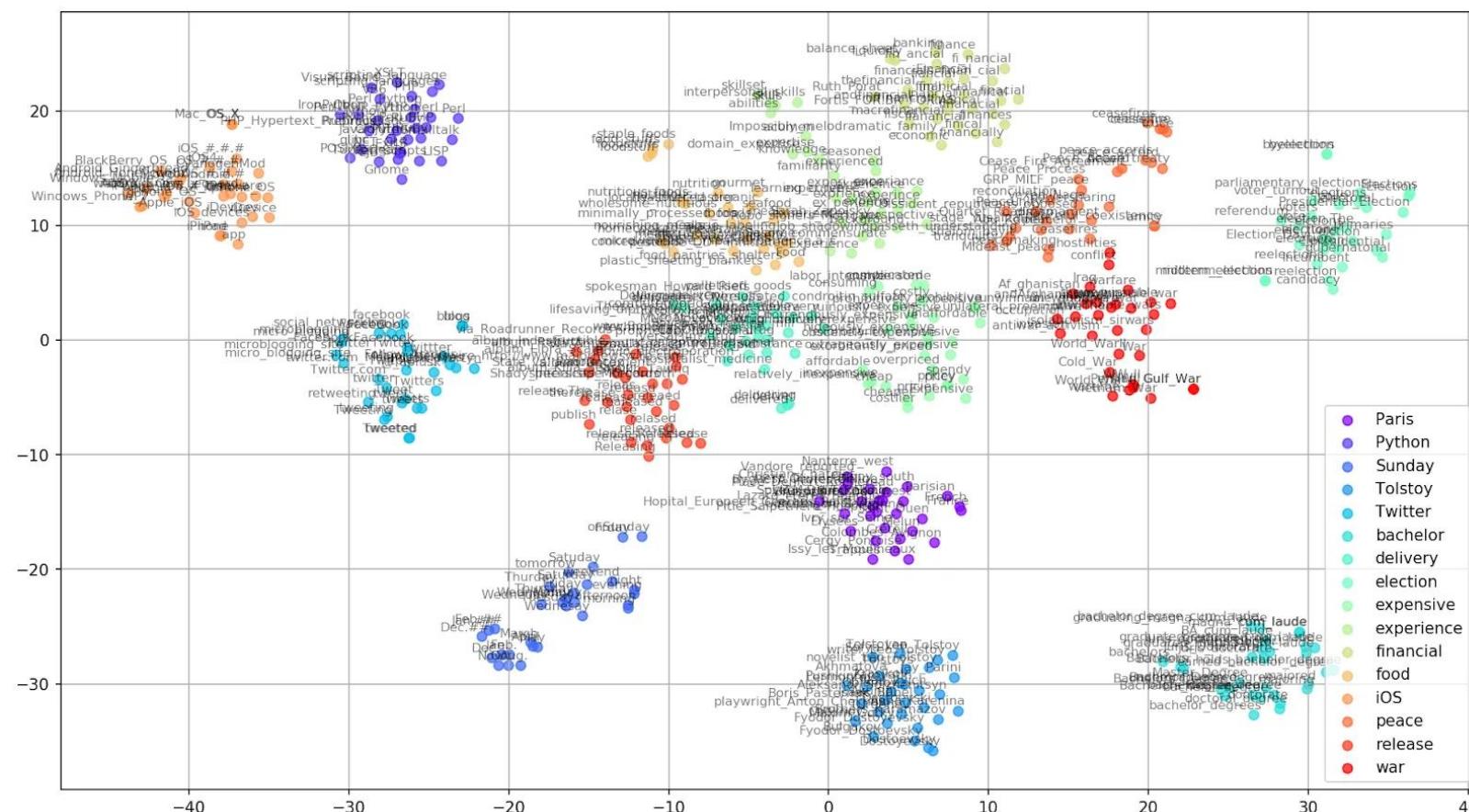
Moreover, in a case of language processing tasks, some MLP layers can be removed without losing the quality of fit.

However, it is more interesting to examine how LLMs generate answers to questions that require new knowledge.



Decoder-only

- It turns out that LLMs behave similarly to **semantic similarity** models, such as **word2vec**!



Source: <https://medium.com/data-science/google-news-and-leo-tolstoy-visualizing-word2vec-word-embeddings-with-t-sne-11558d8bd4d>

Decoder-only

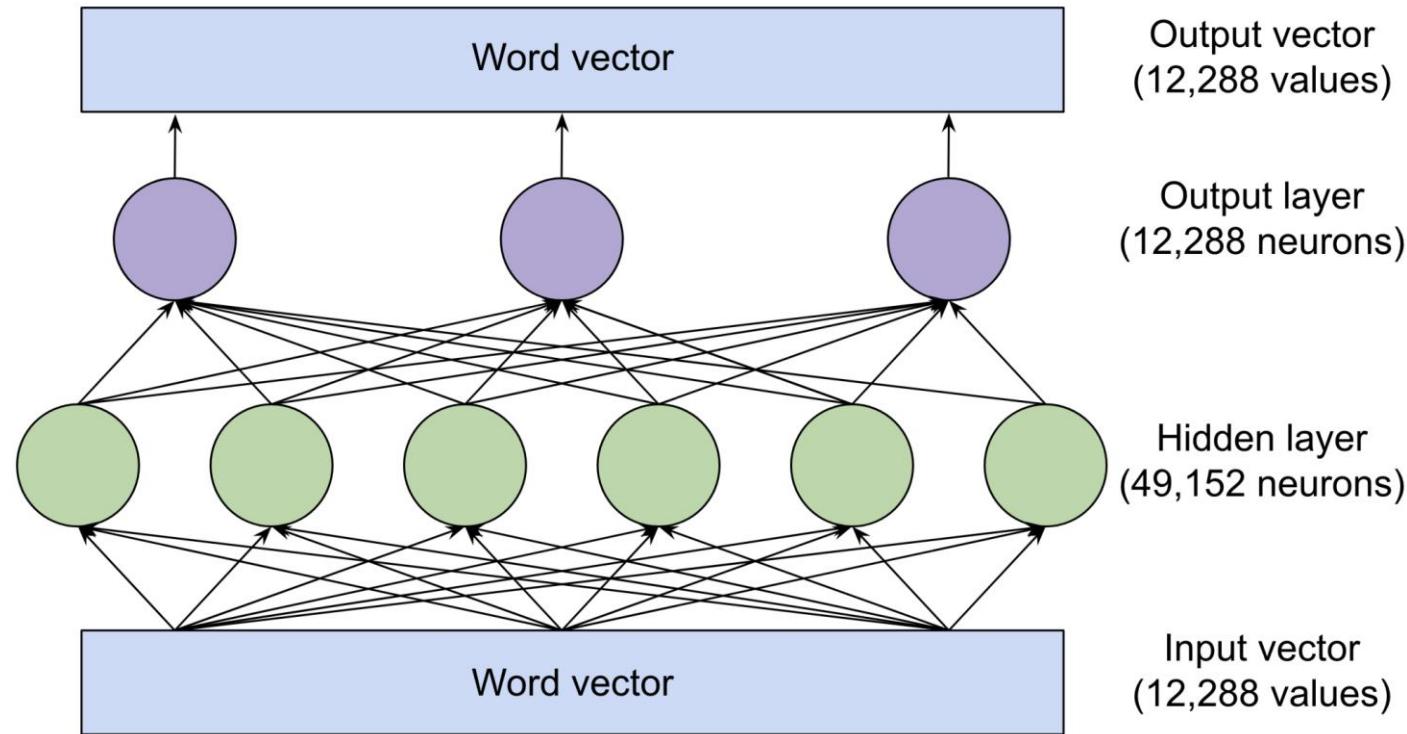
Let's briefly discuss the reasons.

MLP submodule of GPT-3 takes input of length 12,288; there are 49,152 neurons on the output of the first layer. Final output is also a vector of length 12,288,

This means that each layer has about 1.2 billion parameters.

There are 96 layers.

Which gives approximately 116 billion parameters.



Source: <https://arstechnica.com/science/2023/07/a-jargon-free-explanation-of-how-ai-large-language-models-work/>

Decoder-only

But how does the size of Large Language Models impact their theoretical properties?

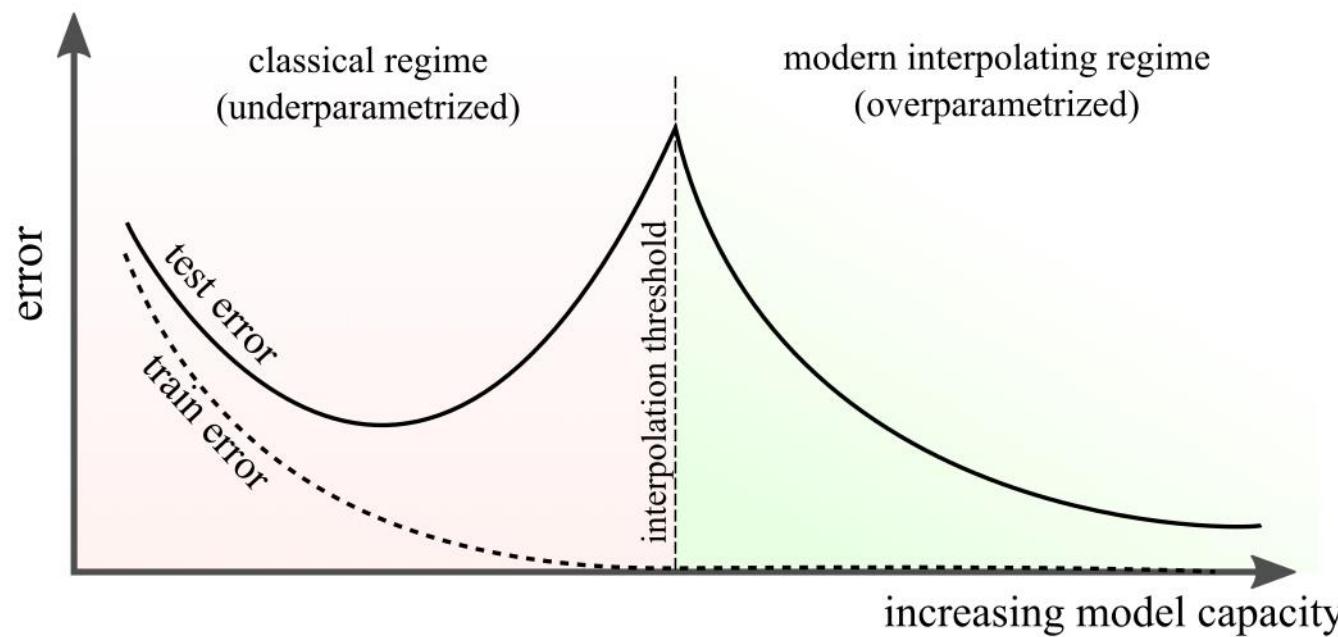
LLM	# of Parameters	Depth L	Width d	# of Heads (Q/KV)
GPT-1 [Radford et al., 2018]	0.117B	12	768	12/12
GPT-2 [Radford et al., 2019]	1.5B	48	1,600	25/25
GPT-3 [Brown et al., 2020]	175B	96	12,288	96/96
LLaMA2 [Touvron et al., 2023b]	7B 13B 70B	32 40 80	4,096 5,120 8,192	32/32 40/40 64/64
LLaMA3/3.1 [Dubey et al., 2024]	8B 70B 405B	32 80 126	4,096 8,192 16,384	32/8 64/8 128/8
Gemma2 [Team et al., 2024]	2B 9B 37B	26 42 46	2,304 3,584 4,608	8/4 16/8 32/16
Qwen2.5 [Yang et al., 2024]	0.5B 7B 72B	24 28 80	896 3,584 8,192	14/2 28/4 64/8
DeepSeek-V3 [Liu et al., 2024a]	671B	61	7,168	128/128
Falcon [Penedo et al., 2023]	7B 40B 180B	32 60 80	4,544 8,192 14,848	71/71 128/128 232/232
Mistral [Jiang et al., 2023a]	7B	32	4,096	32/32

Table 2.2: Comparison of some LLMs in terms of model size, model depth, model width, and number of heads (a/b means a heads for queries and b heads for both keys and values).

Source: Tong Xiao and Jingbo Zhu, 2025. Foundations of Large Language Models, <https://arxiv.org/abs/2501.09223>

Overparametrized Models

- **Double descent** is a phenomenon observed during the training of complex models:



Źródło: Wikipedia

Overparametrized Models

- **Overparameterized** models are often far more efficient than traditional models.
- They are working in the interpolation regime, thus are far more efficient in generating (or predicting) new observations.

Overparametrized Models

- <https://spectrum.ieee.org/midjourney-copyright>

Output from GPT-4:

exempted it from regulations, subsidized its operations and promoted its practices, records and interviews showed.

Their actions turned one of the best-known symbols of New York — its yellow cabs — into a financial trap for thousands of immigrant drivers. More than 950 have filed for bankruptcy, according to a Times analysis of court records, and many more struggle to stay afloat.

"Nobody wanted to upset the industry," said David Klahr, who from 2007 to 2016 held several management posts at the Taxi and Limousine Commission, the city agency that oversees medallions. "Nobody wanted to kill the golden goose."

New York City in particular failed the taxi industry, The Times found. Two former mayors, Rudolph W. Giuliani and Michael R. Bloomberg, placed political allies inside the Taxi and Limousine Commission and directed it to sell medallions to help them balance budgets and fund key initiatives.

During that period, much like in the mortgage lending crisis, a group of industry leaders enriched themselves by artificially inflating medallion prices. They encouraged medallion buyers to borrow as much as possible and ensnared them in interest-only loans and other one-sided deals that often required borrowers to pay hefty fees, forfeit their legal rights and give up most of their monthly incomes.

When the market collapsed, the government largely abandoned the drivers who bore the brunt of the crisis. Officials did not bail out borrowers or persuade banks to soften loan

Actual text from NYTimes:

exempted it from regulations, subsidized its operations and promoted its practices, records and interviews showed.

Their actions turned one of the best-known symbols of New York — its signature yellow cabs — into a financial trap for thousands of immigrant drivers. More than 950 have filed for bankruptcy, according to a Times analysis of court records, and many more struggle to stay afloat.

"Nobody wanted to upset the industry," said David Klahr, who from 2007 to 2016 held several management posts at the Taxi and Limousine Commission, the city agency that oversees cabs. "Nobody wanted to kill the golden goose."

New York City in particular failed the taxi industry, The Times found. Two former mayors, Rudolph W. Giuliani and Michael R. Bloomberg, placed political allies inside the Taxi and Limousine Commission and directed it to sell medallions to help them balance budgets and fund priorities. Mayor Bill de Blasio continued the policies.

Under Mr. Bloomberg and Mr. de Blasio, the city made more than \$855 million by selling taxi medallions and collecting taxes on private sales, according to the city.

But during that period, much like in the mortgage lending crisis, a group of industry leaders enriched themselves by artificially inflating medallion prices. They encouraged medallion buyers to borrow as much as possible and ensnared them in interest-only loans and other one-sided deals that often required them to pay hefty fees, forfeit their legal rights and give up most of their monthly incomes.



videogame plumber --ar 16:9 --v 6.0 --style raw

Bibliography

- Surveys about LLMs:
 - Minaee, S., Mikolov, T., Nikzad, N., Chenaghlu, M., Socher, R., Amatriain, X., & Gao, J. (2025). *Large Language Models: A Survey*.
<https://arxiv.org/abs/2402.06196>
 - Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2024). *A Comprehensive Overview of Large Language Models*.
<https://arxiv.org/abs/2307.06435>
 - Xiao, T., & Zhu, J. (2025). *Foundations of Large Language Models*.
<https://arxiv.org/abs/2501.09223>

Bibliography

- Seminal papers:
 1. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
<https://arxiv.org/abs/1810.04805>
 2. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2023). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. <https://arxiv.org/abs/1910.10683>
 3. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). *LLaMA: Open and Efficient Foundation Language Models*. <https://arxiv.org/abs/2302.13971>
 4. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). *Language Models are Few-Shot Learners*.
<https://arxiv.org/abs/2005.14165>

Bibliography

- Attention Mechanism:
 1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). Attention Is All You Need. <https://arxiv.org/abs/1706.03762>
 2. Cheng, J., Dong, L., & Lapata, M. (2016). *Long Short-Term Memory-Networks for Machine Reading*. <https://arxiv.org/abs/1601.06733>
 3. Bengio, Y., Ducharme, R., & Vincent, P. (2000). A Neural Probabilistic Language Model. In T. Leen, T. Dietterich, & V. Tresp (Eds.), *Advances in Neural Information Processing Systems* (Vol. 13). MIT Press. https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf
 4. Luong, M.-T., Pham, H., & Manning, C. D. (2015). *Effective Approaches to Attention-based Neural Machine Translation*. <https://arxiv.org/abs/1508.04025>
 5. Bahdanau, D., Cho, K., & Bengio, Y. (2016). *Neural Machine Translation by Jointly Learning to Align and Translate*. <https://arxiv.org/abs/1409.0473>
 6. Weng, L. (2018). Attention? Attention! *Lilianweng.Github.io*. <https://lilianweng.github.io/posts/2018-06-24-attention/>

Bibliography

- Experimental Results:
 1. Geva, M., Schuster, R., Berant, J., & Levy, O. (2021). *Transformer Feed-Forward Layers Are Key-Value Memories*. <https://arxiv.org/abs/2012.14913>
 2. Merullo, J., Eickhoff, C., & Pavlick, E. (2024). *Language Models Implement Simple Word2Vec-style Vector Arithmetic*. <https://arxiv.org/abs/2305.16130>
 3. Stechly, K., Valmeekam, K., Gundawar, A., Palod, V., & Kambhampati, S. (2025). *Beyond Semantics: The Unreasonable Effectiveness of Reasonless Intermediate Tokens*. <https://arxiv.org/abs/2505.13775>
 4. Herel, D., & Mikolov, T. (2024). *Collapse of Self-trained Language Models*. <https://arxiv.org/abs/2404.02305>