CS5242 : Neural Networks and Deep Learning

Lecture 7 : Attention Neural Networks

Semester 2 2024/25

Xavier Bresson

https://twitter.com/xbresson

Department of Computer Science
National University of Singapore (NUS)

Outline

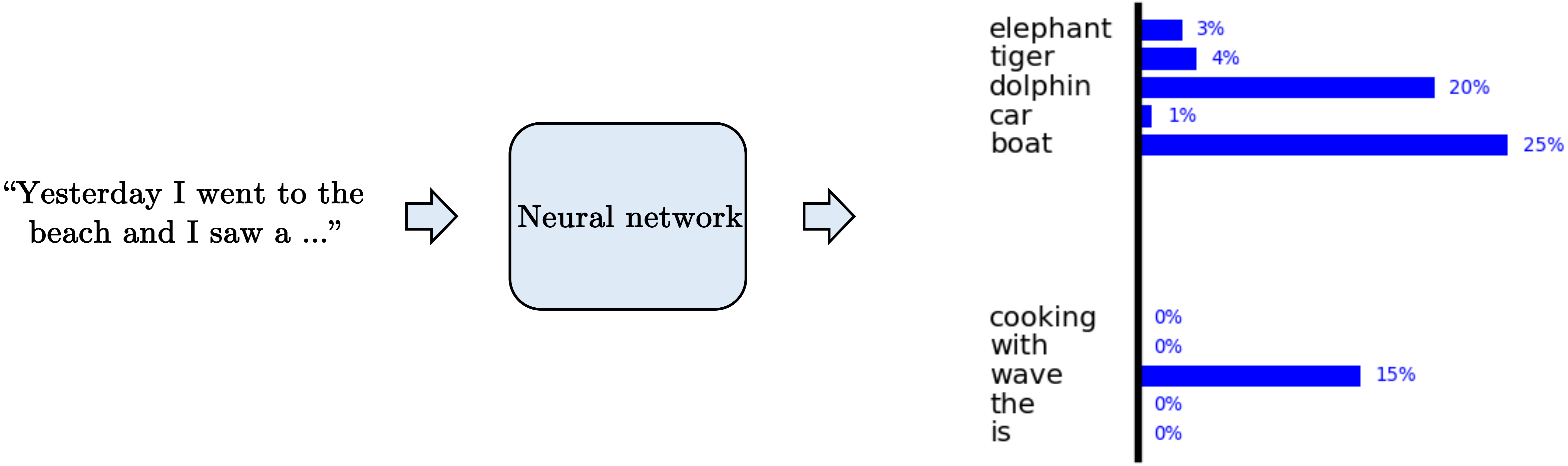
- Language Models
- Memory Networks
- Transformers
- Language Model Transformers
- Sequence-To-Sequence Transformers
- Transfer Learning
- Conclusion

Outline

- **Language Models**
- Memory Networks
- Transformers
- Language Model Transformers
- Sequence-To-Sequence Transformers
- Transfer Learning
- Conclusion

Language models

- Language model predicts the next word given a context window.
- Most fundamental problem in NLP.



“Yesterday I went to the beach and I saw a ...”

Neural network

Input:
Sequence of words

Output:
Probability distribution over the dictionary/vocabulary

Recurrent neural networks

- Data structure
  - Input is an ordered sequence.
  - Input length and output length can be variable.
- RNNs are designed for sequences.
  - They learn a representation of sequence independently of its length.
    - Recurrence formula summarizes the sequence with a vector h :

$$h \leftarrow f_W(h, x)$$

    - Weight sharing across time (translation invariance)
  - They learn to keep or ignore information in the sequence for the downstream task.
    - Gating mechanism to forget/remember the past or the new input :

$$\sigma \odot h$$

Recurrent neural networks

- Performance
  - Significant progress in NLP but not a breakthrough.
  - Dominant in NLP for Machine Translation (MT), Q&A, summarization up to 2018.
- Limitation
  - RNNs cannot learn long-term dependencies (no more than 50 steps).
  - Hard to train because they are non-linear dynamical systems
    - Any small perturbation can amplify or vanish.
  - Slow to train because of their sequential nature (due to recurrence mechanism).
    - Important limitation when training on large-scale datasets.

Outline

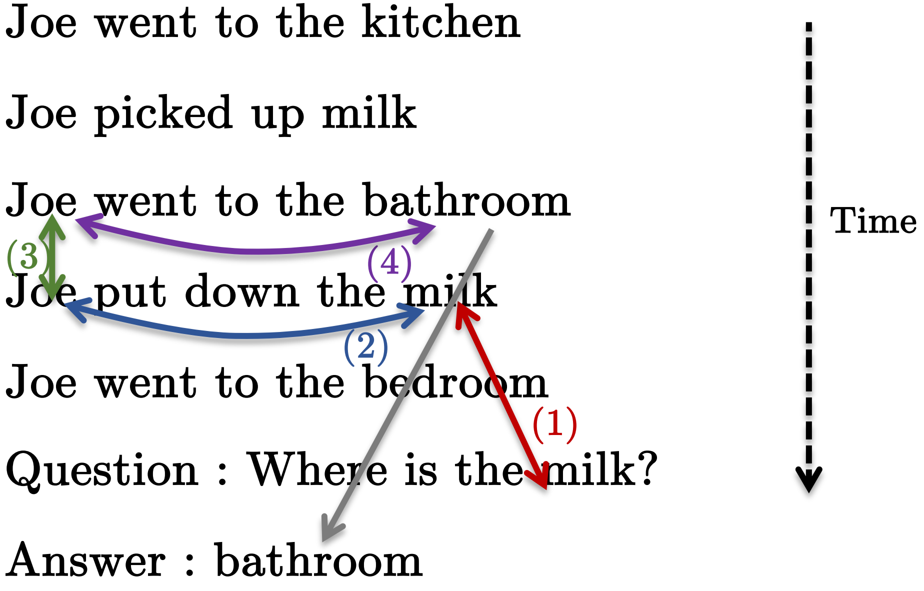
- Language Models
- **Memory Networks**
- Transformers
- Language Model Transformers
- Sequence-To-Sequence Transformers
- Transfer Learning
- Conclusion

Memory networks

- How do these models work?
- Let us consider a simple example from the bAbI dataset (Meta 2015).
- This dataset is used to evaluate simple reasoning property of models.
- Example :
  - Joe went to the kitchen
  - Joe picked up milk
  - Joe went to the bathroom
  - Joe put down the milk
  - Joe went to the bedroom
  - Question : Where is the milk?
  - Answer : ---

Memory networks

- Memory networks (Weston-Chopra-Bordes, Meta, 2014 and Sukhbaatar-Szlam-Weston-Fergus, Meta, 2015) are designed to response to the questions with multi-step word matching :

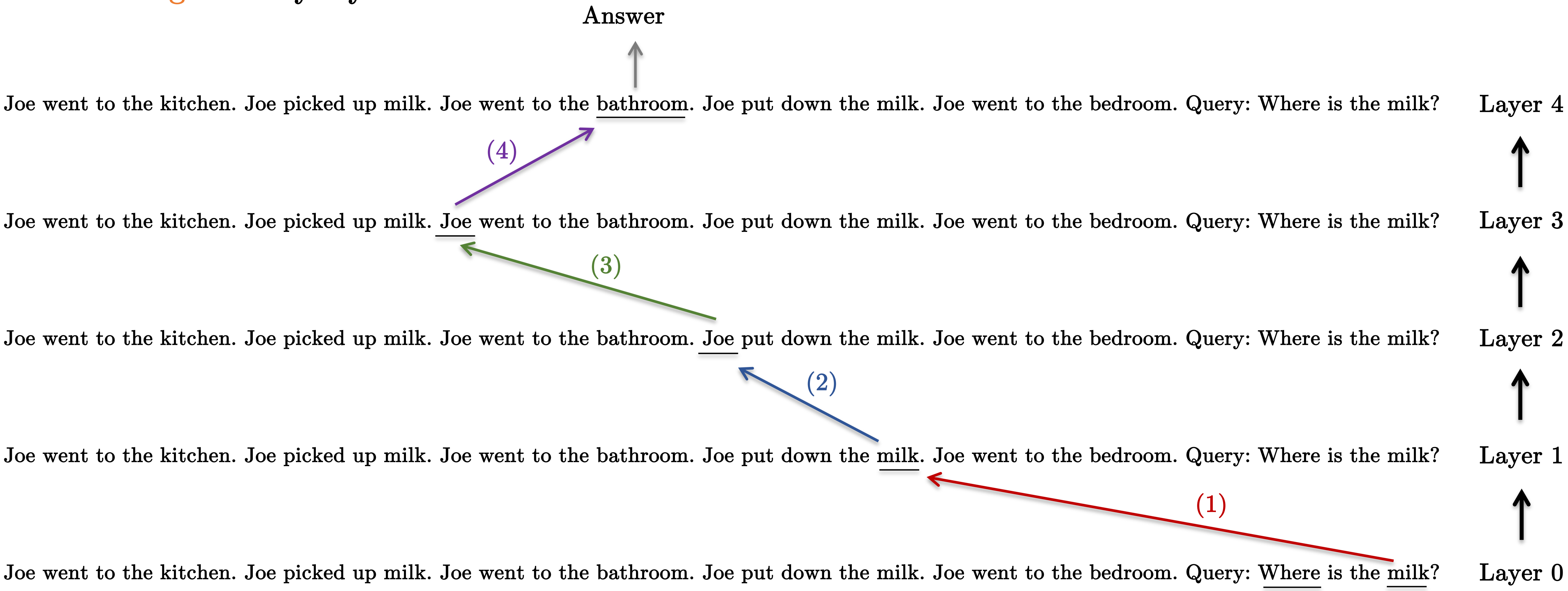


  - Joe went to the kitchen
  - Joe picked up milk
  - Joe went to the bathroom
  - Joe put down the milk
  - Joe went to the bedroom
  - Question : Where is the milk?
  - Answer : bathroom

- This matching process is also called multi-hop attention.
- Multi-hop attention is a mechanism that performs multi-step reasoning.

Multi-hop attention steps

- Unrolling memory layers :



Answer

Joe went to the kitchen. Joe picked up milk. Joe went to the bathroom. Joe put down the milk. Joe went to the bedroom. Query: Where is the milk? Layer 4

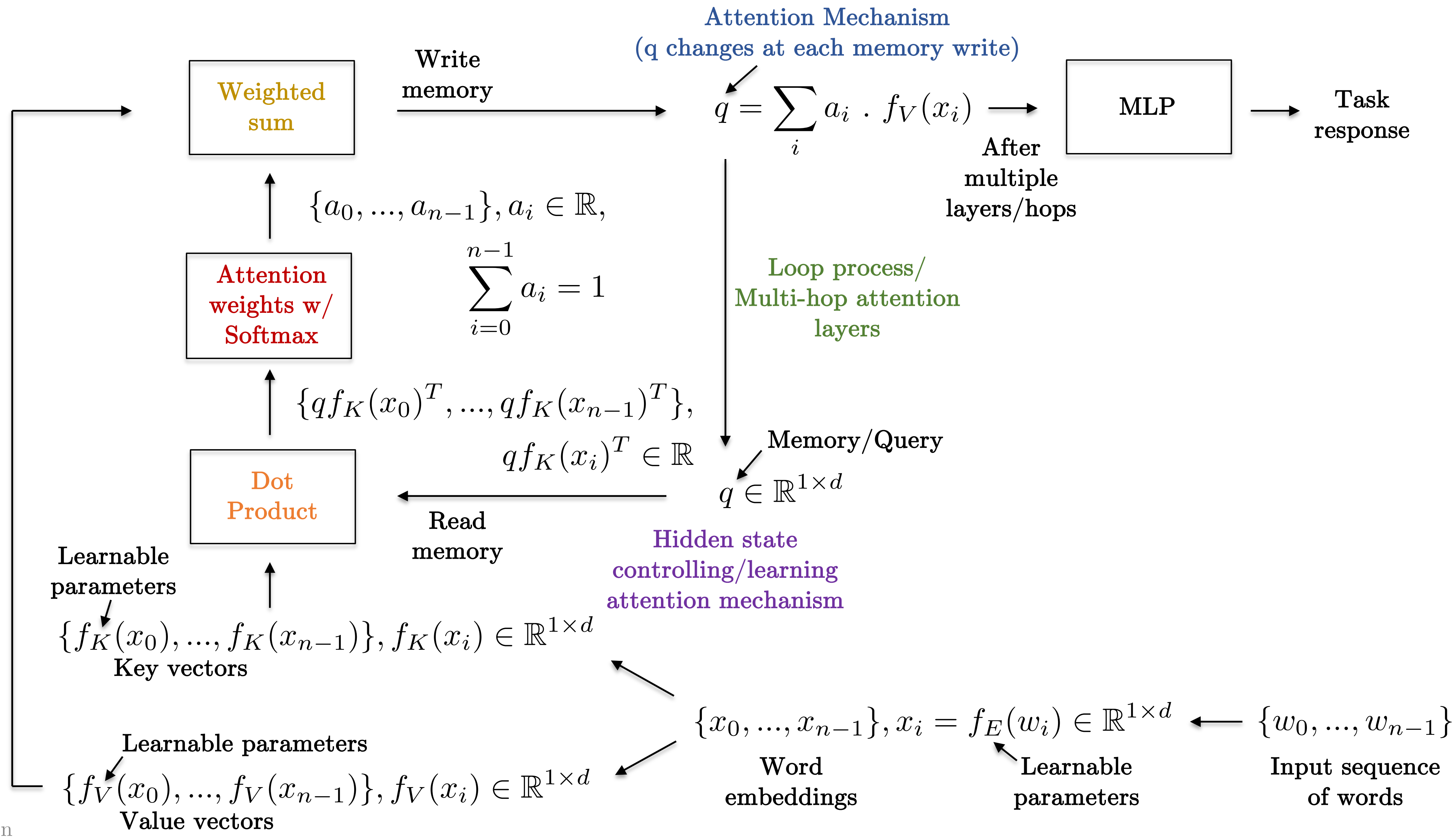
Joe went to the kitchen. Joe picked up milk. Joe went to the bathroom. Joe put down the milk. Joe went to the bedroom. Query: Where is the milk? Layer 3

Joe went to the kitchen. Joe picked up milk. Joe went to the bathroom. Joe put down the milk. Joe went to the bedroom. Query: Where is the milk? Layer 2

Joe went to the kitchen. Joe picked up milk. Joe went to the bathroom. Joe put down the milk. Joe went to the bedroom. Query: Where is the milk? Layer 1

Joe went to the kitchen. Joe picked up milk. Joe went to the bathroom. Joe put down the milk. Joe went to the bedroom. Query: Where is the milk? Layer 0

Implementation



Attention Mechanism
(q changes at each memory write)

Weighted sum

Write memory

$q = \sum_i a_i \,.\, f_V(x_i)$

After multiple layers/hops

MLP

Task response

$\{a_0, \ldots, a_{n-1}\}, a_i \in \mathbb{R}, \quad \sum_{i=0}^{n-1} a_i = 1$

Attention weights w/ Softmax

Loop process/ Multi-hop attention layers

$\{q f_K(x_0)^T, \ldots, q f_K(x_{n-1})^T\},$
$q f_K(x_i)^T \in \mathbb{R}$

Dot Product

Read memory

Memory/Query
$q \in \mathbb{R}^{1 \times d}$

Hidden state controlling/learning attention mechanism

Learnable parameters
$\{f_K(x_0), \ldots, f_K(x_{n-1})\}, f_K(x_i) \in \mathbb{R}^{1 \times d}$
Key vectors

Learnable parameters
$\{f_V(x_0), \ldots, f_V(x_{n-1})\}, f_V(x_i) \in \mathbb{R}^{1 \times d}$
Value vectors

$\{x_0, \ldots, x_{n-1}\}, x_i = f_E(w_i) \in \mathbb{R}^{1 \times d}$
Word embeddings

Learnable parameters

$\{w_0, \ldots, w_{n-1}\}$
Input sequence of words

Formalization

- **Input sequence of words :** $\{w_0, ..., w_{n-1}\}, w_i \in \{0, ..., V-1\}$

- **Continuous representation/ word embedding :** $\{x_0, ..., x_{n-1}\},\ x_i = f_E(w_i) \in \mathbb{R}^d$

- **Hidden features :**

$$\underset{\textbf{Key}}{K} = \begin{bmatrix} f_K(x_0) \\ \vdots \\ f_K(x_{n-1}) \end{bmatrix} \in \mathbb{R}^{n \times d} \qquad \underset{\textbf{Query}}{q \in \mathbb{R}^{1 \times d}} \qquad \underset{\textbf{Value}}{V} = \begin{bmatrix} f_V(x_0) \\ \vdots \\ f_V(x_{n-1}) \end{bmatrix} \in \mathbb{R}^{n \times d}$$

- **Repeat K times :**

$$a \leftarrow \text{Softmax}(qK^T) \ \in \mathbb{R}^{1 \times n}$$
$$q \leftarrow aV = \text{Softmax}(qK^T)V \quad \in \mathbb{R}^{1 \times d}$$

- **Output :** $s = \text{MLP}(q) \in \mathbb{R}^V$

Properties

- This model is seen as differentiable memory computers, i.e. memory operations read and write can be differentiable and thus be used with backpropagation.
- This network can update its memory by stacking multiple-hop attention layers to perform multi-step reasoning.
- This model is based on the principle that intelligence requires an adaptive long-term memory, unlike RNNs which is limited to short-term memory.
- This technique is a precursor of Transformers.

Outline

- Language Models
- Memory Networks
- Transformers
- Language Model Transformers
- Sequence-To-Sequence Transformers
- Transfer Learning
- Conclusion

Limitations of memory networks

- Memory networks were promising, but not ground-breaking.
- Transformers designed the first efficient version of attention networks !
- Transformer improvements over memory networks :
  - Multiple queries (one per word)
  - Multi-head attention mechanism (more learning capacity)
  - Residual blocks (better backpropagation)

Self-attention mechanism

- Input set (continuous representation) :

$$\{x_0, ..., x_{n-1}\}, \quad X = \begin{bmatrix} x_0 \\ \vdots \\ x_{n-1} \end{bmatrix} \in \mathbb{R}^{n \times d}$$

- Initiate hidden state :

$$H = X \in \mathbb{R}^{n \times d}$$

- Repeat K layers :

$$H \leftarrow \text{Softmax}(QK^T)V \quad \in \mathbb{R}^{n \times d}$$

Self-attention layer

Differentiable dictionary

Dict is a standard structure in CS

Dict=(Key,Value)

$$\begin{aligned} K &= HW^K \in \mathbb{R}^{n \times d}, \ W^K \in \mathbb{R}^{d \times d} \\ V &= HW^V \in \mathbb{R}^{n \times d}, \ W^V \in \mathbb{R}^{d \times d} \\ Q &= HW^Q \in \mathbb{R}^{n \times d}, \ W^Q \in \mathbb{R}^{d \times d} \end{aligned}$$

Query the dictionary

for a key and its value.

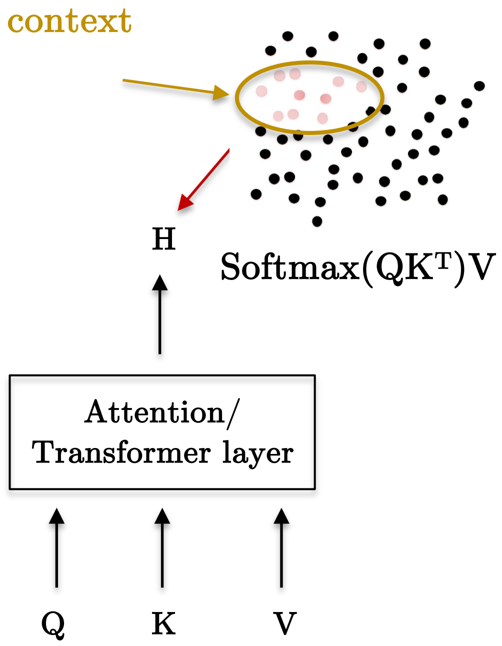
Learnable parameters

Context-to-word representation

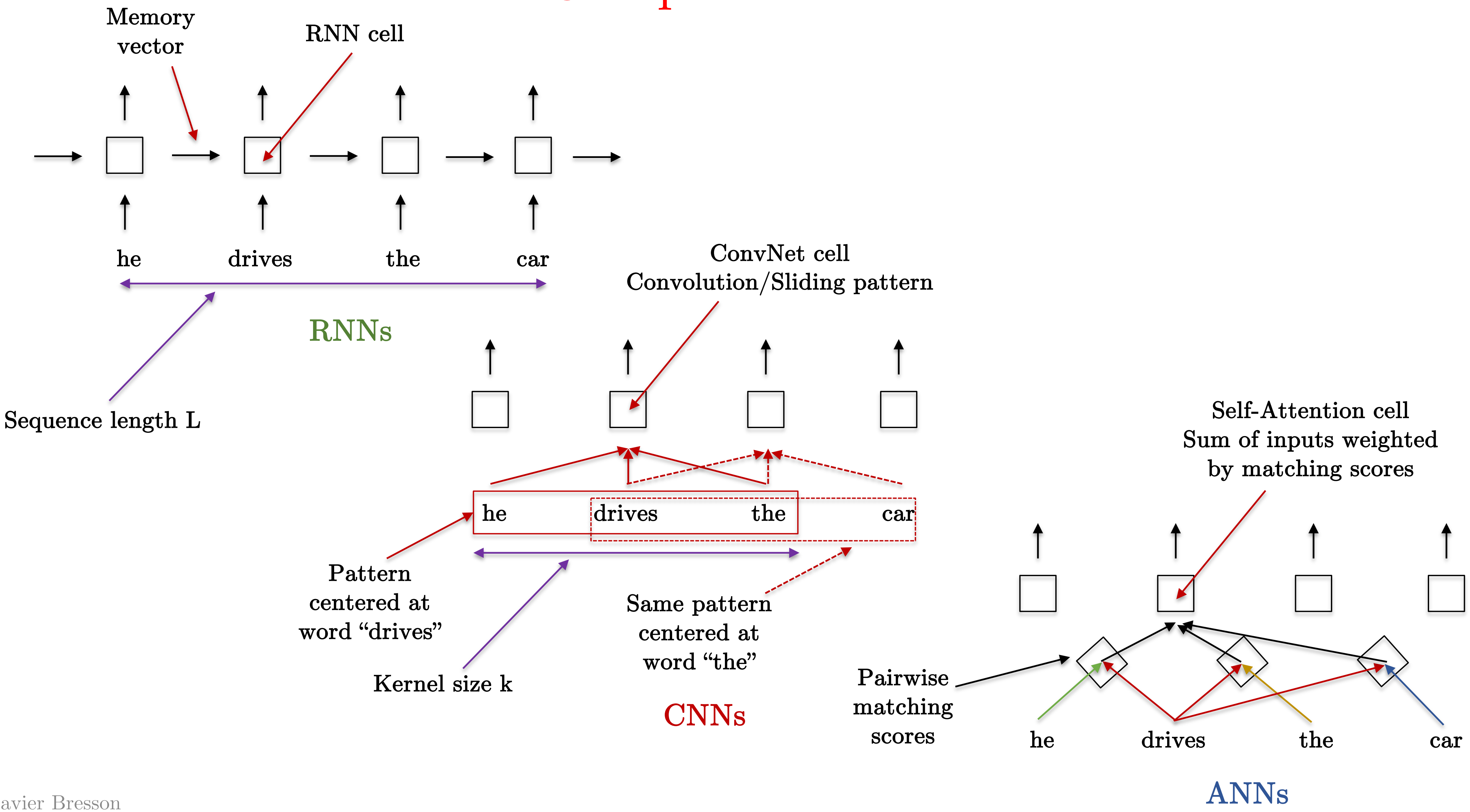
- From fixed word representation to context-to-word representation :

$$H \leftarrow \text{Softmax}(QK^T)V \in \mathbb{R}^{n\times d}$$

- The new data representation is a sum of all input data weighted by the pairwise matching (or attention) scores.
- The subset of data with non-zero attention scores forms the context.
- The attention mechanism allows to dynamically change the word representation according to its context.
- Context-to-word is a powerful idea in NLP because a word may have different meanings, that can only be clarified in a particular context :
    - The vase broke. The news broke. Sandy broke the world record. Sandy broke the law. We broke even. The burglar broke into the house. Etc.



Computational cost



Memory vector

RNN cell

he drives the car

Sequence length L

RNNs

ConvNet cell
Convolution/Sliding pattern

he drives the car

Pattern centered at word “drives”

Kernel size k

Same pattern centered at word “the”

CNNs

Self-Attention cell
Sum of inputs weighted by matching scores

Pairwise matching scores

he drives the car

ANNs

Computational cost

- RNN layer : $O(L.d^2)$
- ConvNet layer : $O(L.d^2.k)$
- Transformer layer : $O(L^2.d)$

Seems bad !

with L : sequence length, d : hidden feature size, k : kernel size

- Attention networks have actually less parameters to learn as long as $L \leq d$!
  - Example 1 : $L = 100$, $d = 1000$, $k = 3$

    RNN: $O(10^8)$, ConvNet: $O(3.10^8)$, Transformer: $O(10^7)$
  - Example 2 : $L = 1000$, $d = 1000$, $k = 3$

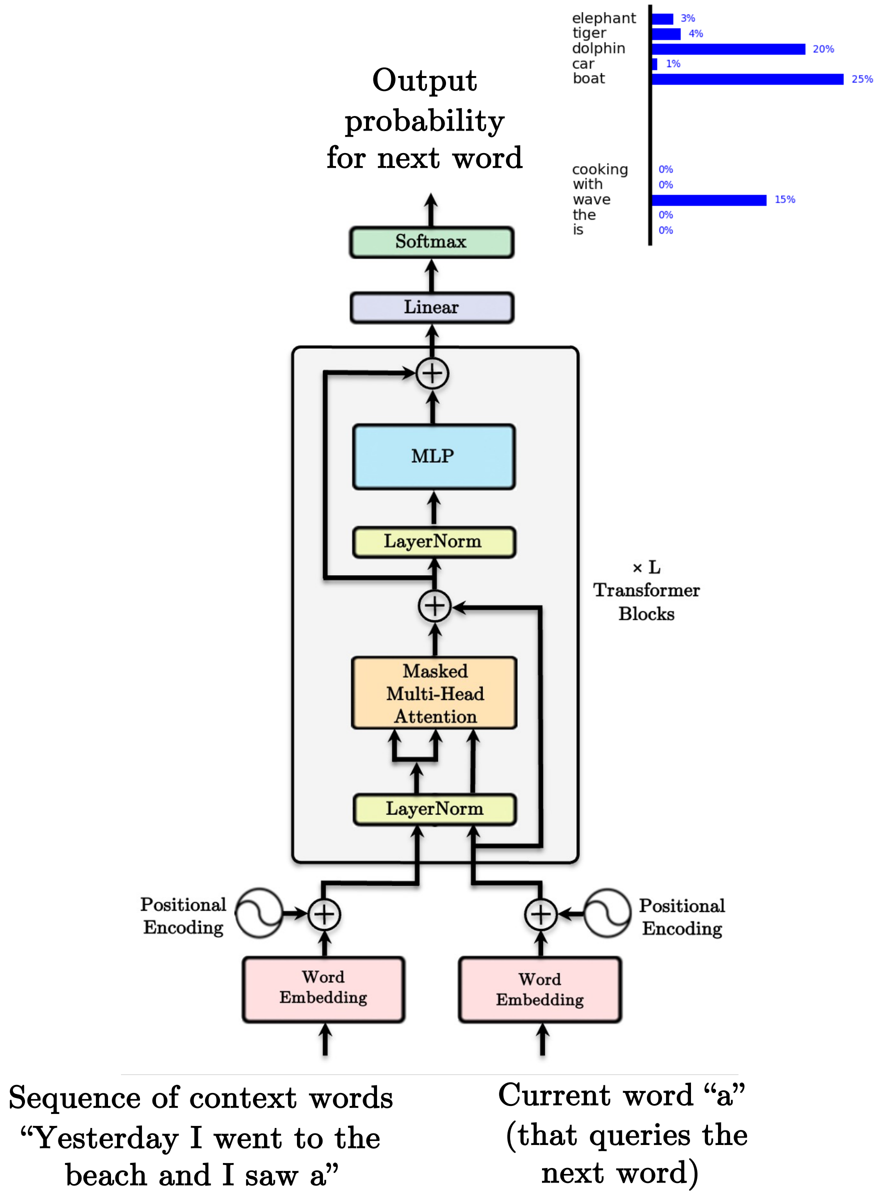
    RNN: $O(10^9)$, ConvNet: $O(3.10^9)$, Transformer: $O(10^9)$

Outline

- Language Models
- Memory Networks
- Transformers
- Language Model Transformers
- Sequence-To-Sequence Transformers
- Transfer Learning
- Conclusion

Language model transformers

- Given a sequence of words, predict the next word.
- This task to be successful requires a word representation that can be changed with different contexts.
- Transformers offer expressive word representation with its word-in-context property.
- A LM transformer is composed of three layers :
  - Word embedding layer
  - Attention layer
  - Classification layer

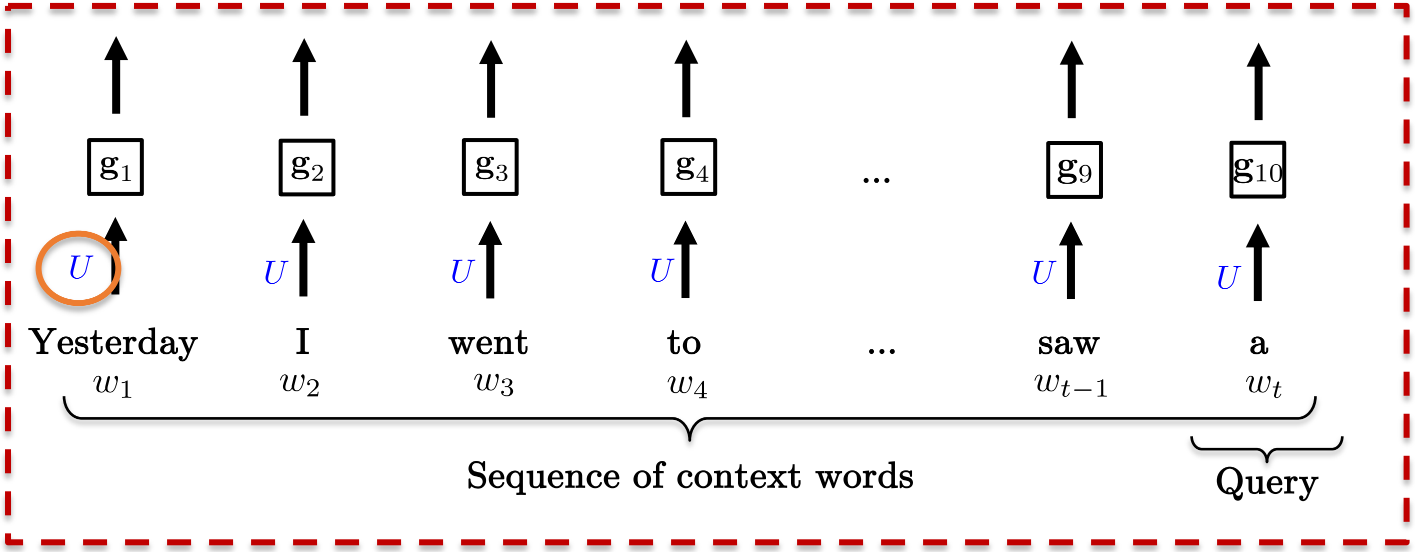


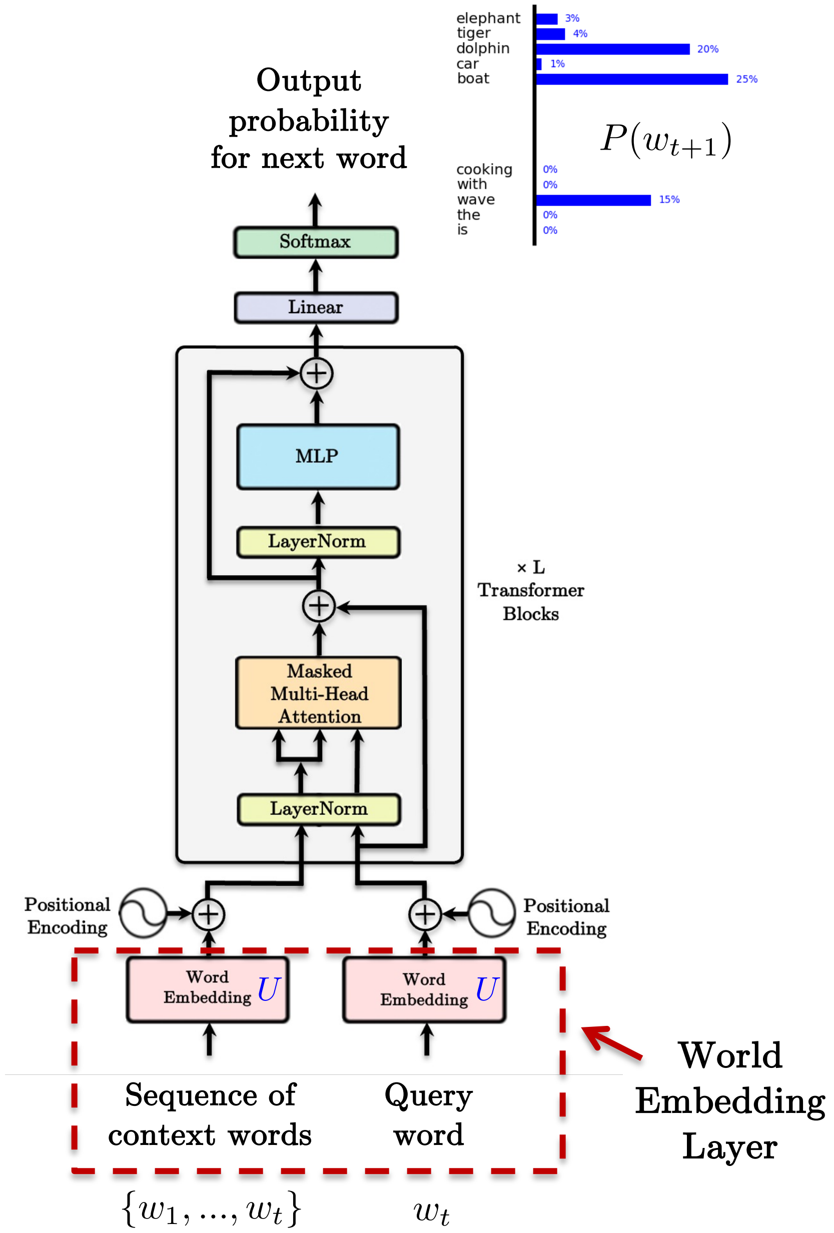
Sequence of context words "Yesterday I went to the beach and I saw a"

Current word "a" (that queries the next word)

Word embedding layer

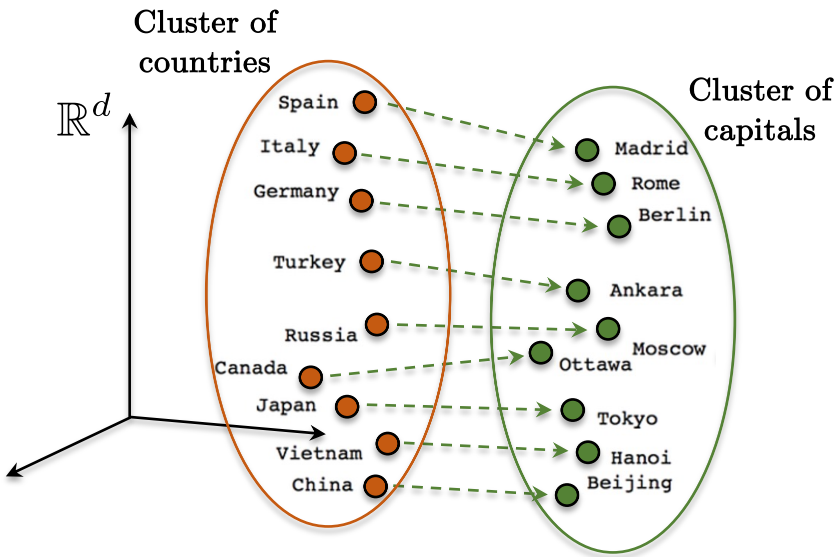
- Categorical variables (dictionary of 10,000 words) are
  - represented by one-hot vectors and then
  - embedded into a linear space.
- This is the same input embedding as in RNNs :
  - PyTorch nn.Embedding()



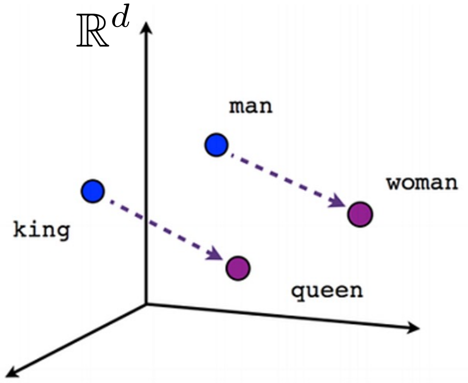


Interpretation

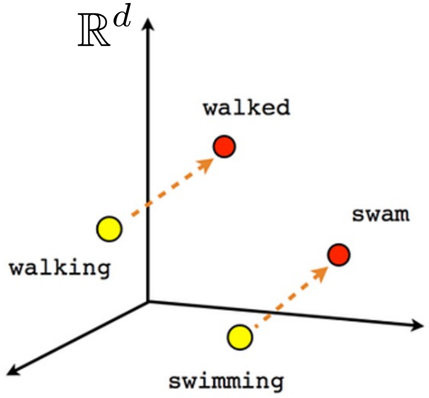
- Word embeddings convert discrete words into continuous vector representations, where similar vectors indicate similar meanings.
- In other words, this process groups semantically related words into clusters — such as countries, sports, or professions.
- Moreover, word embeddings capture relationships between words, enabling analogies like country → capital or male → female, to be represented through consistent vectors.



country → capital



male → female



present participle → past tense

Multi-head attention

- Update equation for one MHA layer :
  - PyTorch nn.MultiheadAttention()

$$\bar{h} = \text{MHA}(\overset{\text{Query}}{q}, \overset{\text{Key}}{K}, \overset{\text{Value}}{V}) \in \mathbb{R}^d,\ \ q \in \mathbb{R}^d, K \in \mathbb{R}^{L\times d}, V \in \mathbb{R}^{L\times d}$$

$$= \Big(\|_{h=1}^{H} \ \text{HA}_h(q,K,V) \Big) W^O,\ \ W^O \in \mathbb{R}^{d\times d}$$

Concatenation operation

We consider sequence of L words

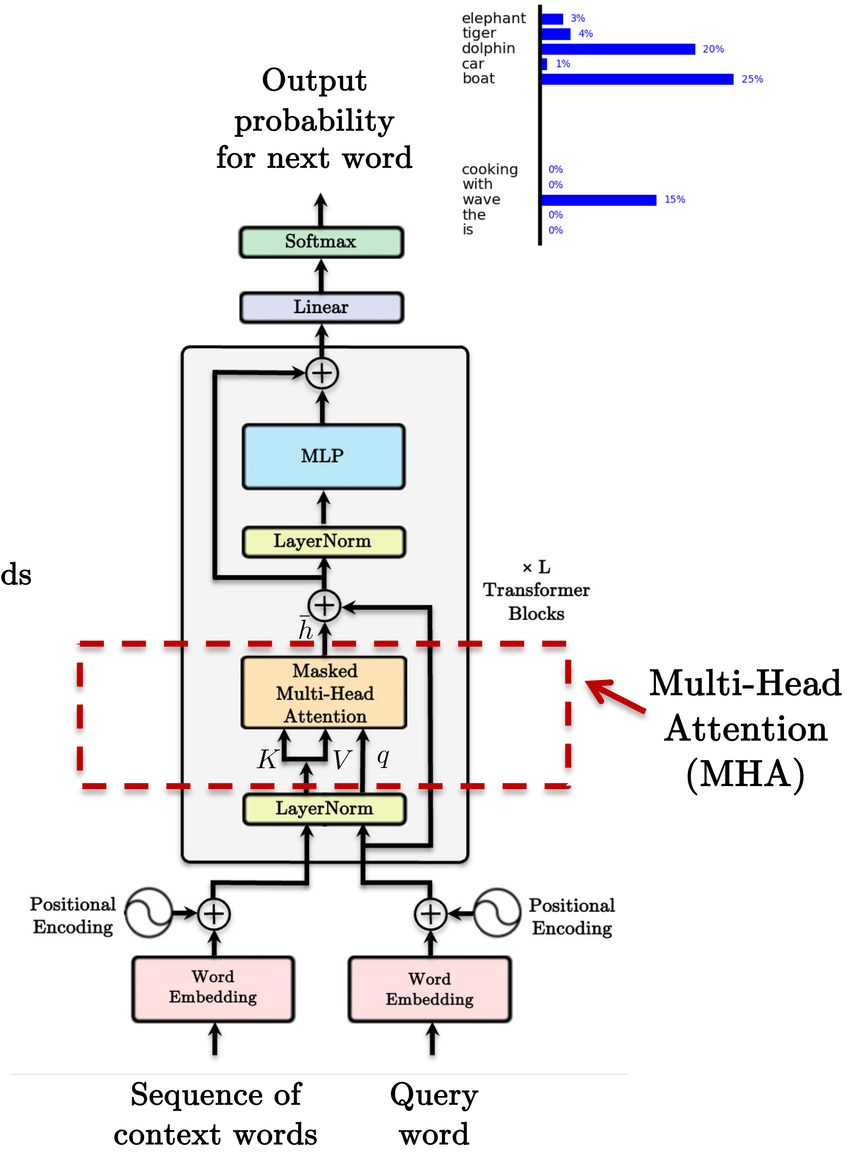
$$\text{with } q = g_t \in \mathbb{R}^d,\ \ K = V = \{g_t, g_{t-1}, ..., g_{t-(L-1)}\} \in \mathbb{R}^{L\times d}$$

$$\text{HA}_h(q,K,V) = \text{Softmax}\Big(\frac{q_h K_h^T}{\sqrt{d/H}}\Big) V_h \in \mathbb{R}^{d/H}$$

$$\text{with } q_h = qW_h^q \in \mathbb{R}^{d/H}, W_h^q \in \mathbb{R}^{d\times d/H}$$

$$K_h = KW_h^K \in \mathbb{R}^{n\times d/H}, W_h^K \in \mathbb{R}^{d\times d/H}$$

$$V_h = VW_h^V \in \mathbb{R}^{n\times d/H}, W_h^V \in \mathbb{R}^{d\times d/H}$$

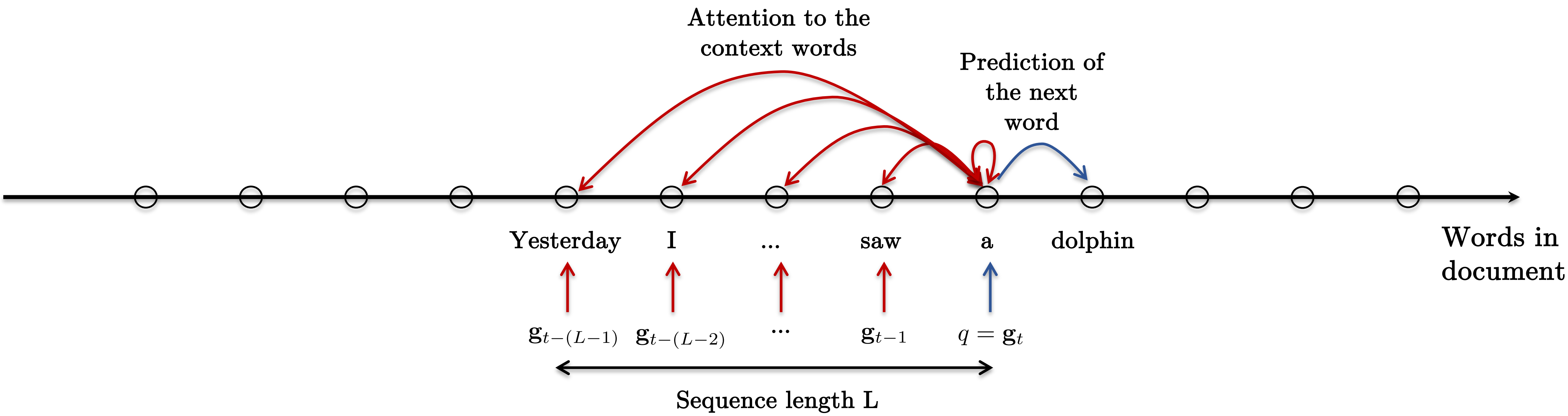


Multi-Head Attention (MHA)

Head attention layer

- **Equation of a single head attention layer :**

$$\mathrm{HA}(q,K,V)=\mathrm{Softmax}\Big(\frac{qK^T}{\sqrt{d}}\Big)V\in\mathbb{R}^{1\times d},\ \ q\in\mathbb{R}^{1\times d},K\in\mathbb{R}^{L\times d},V\in\mathbb{R}^{L\times d}$$



Transformer block

- Transformer or attention block layer (2017) :

$$\bar{h} = \text{LN}\big(q + \text{MHA}(q,K,V)\big) \in \mathbb{R}^d$$

$$h = \text{LN}\big(\underbrace{\bar{h} +}\text{MLP}(\bar{h})\big) \in \mathbb{R}^d$$

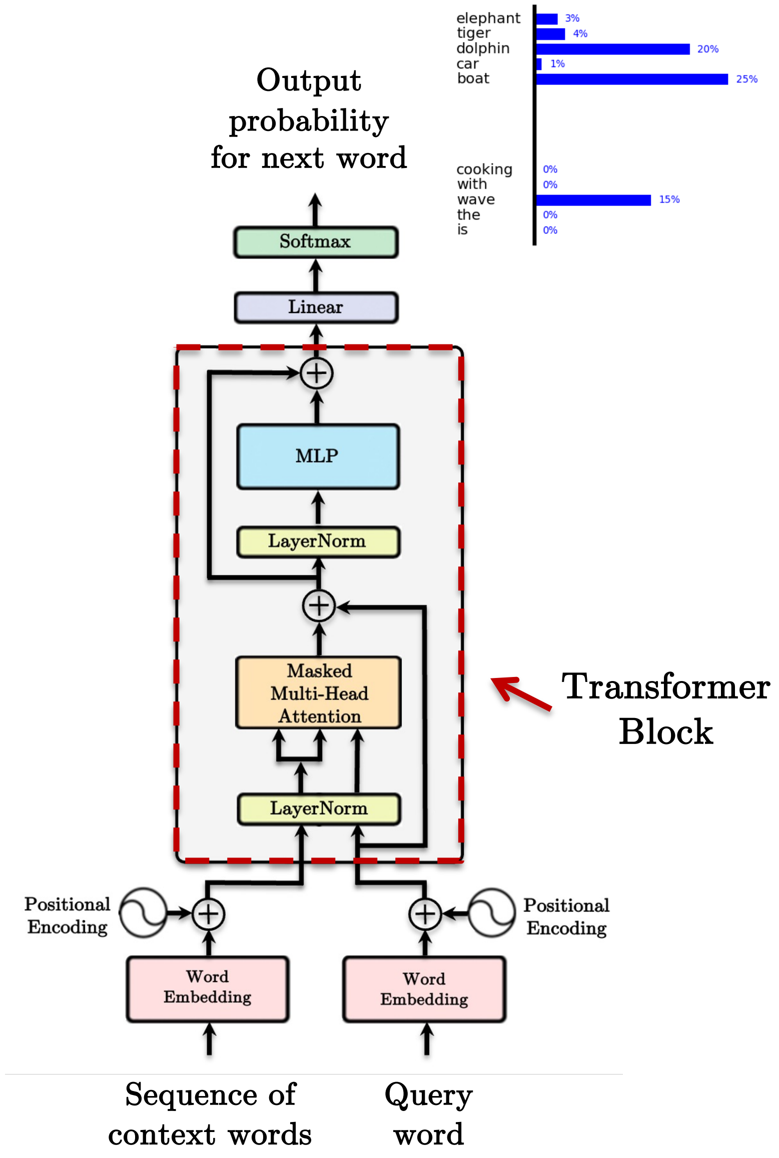
Layer normalization
z-scoring with learnable parameters
nn.LayerNorm()

Residual connection

- In 2019, LayerNorm (LN) was applied before non-linear operations :

$$\bar{h} = q + \text{MHA}(\text{LN}(q), \text{LN}(K), \text{LN}(V)) \in \mathbb{R}^d$$

$$h = \bar{h} + \text{MLP}(\text{LN}(\bar{h})) \in \mathbb{R}^d$$



LayerNorm and residual connection

- Layer normalization :

z-scoring

$$\mathrm{LN}(q) = a \odot \left(\frac{q-\mu}{\sigma}\right) + b \in \mathbb{R}^d$$

$$\text{with } \mu = \mathrm{Mean}(q) \in \mathbb{R}, \sigma = \mathrm{Std}(q) \in \mathbb{R}, a, b \in \mathbb{R}^d$$

- Residual/skip connection :

$$x \longrightarrow \boxed{\cdot + f_W(\cdot)} \longrightarrow y = x + f_W(x)$$

Forward pass

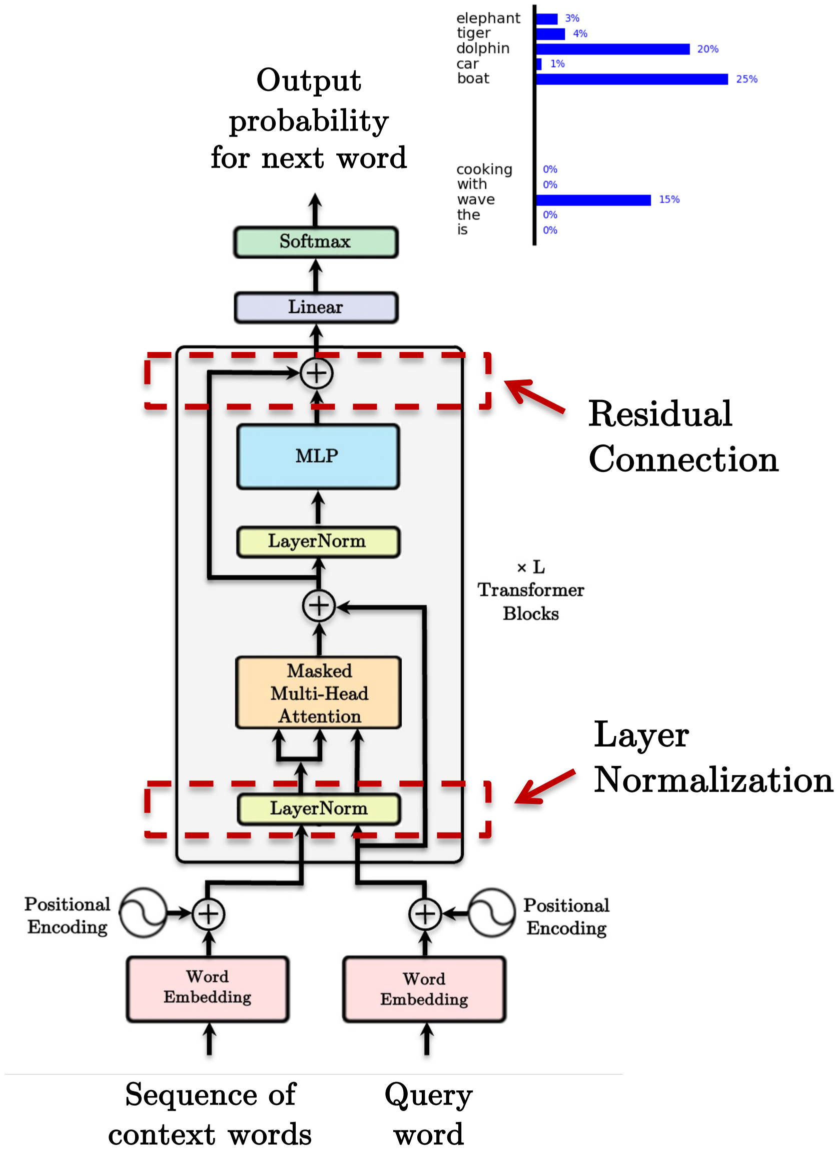
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} \longleftarrow \boxed{\cdot + f_W(\cdot)} \longleftarrow \frac{\partial L}{\partial y}$$

Backward pass

$$= \frac{\partial L}{\partial y}(\mathrm{Id} + \nabla_x f_w)$$

$$= \frac{\partial L}{\partial y} + \frac{\partial L}{\partial y}\nabla_x f_w$$

No vanishing gradient for residual connection

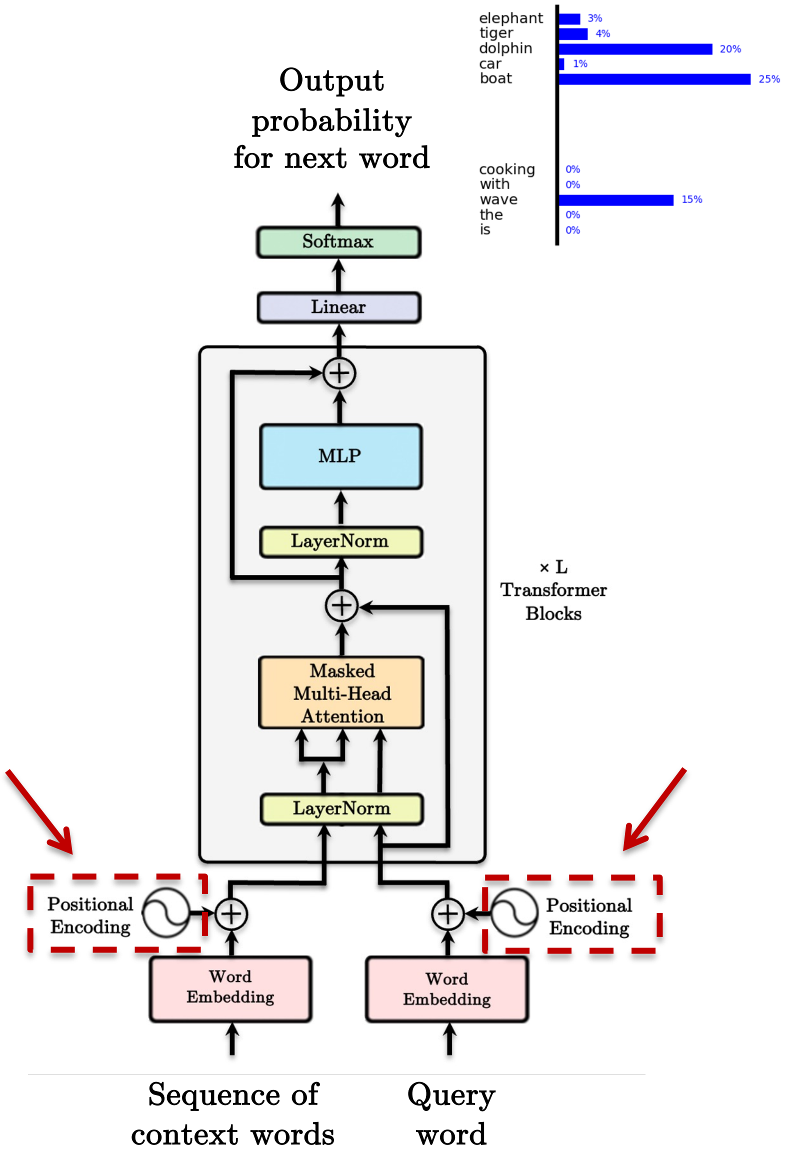


Residual Connection

Layer Normalization

Positional encoding

- Transformers are designed to process sets of vectors but items in a set are not ordered.
- This is an issue for NLP tasks.
  - An additional ordering feature is required to inject causal ordering in the attention mechanism.
- Two classes of Positional Encoding (PE) :
  - Learnable vs non-learnable PE
- Learnable PE :
  - Embedding of discrete ordering index 0,1,2,3,…,L-1, with L is the sequence length.
  - Two issues :
    - Requires to know the maximum L value among all training sequences.
    - Some test sequences may have lengths not present in the train set.



Positional encoding

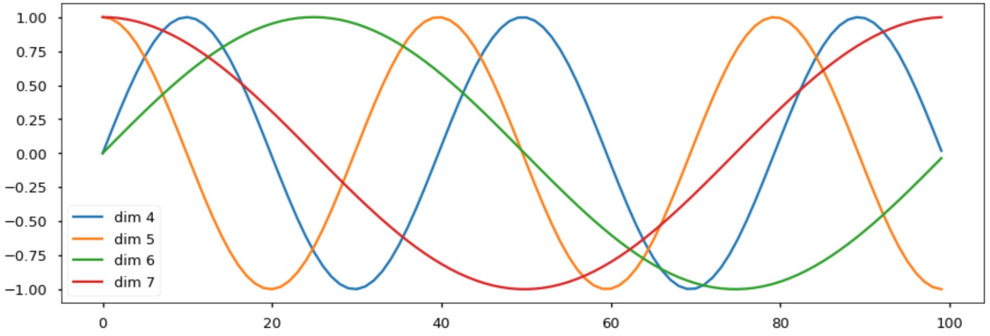
- Non-learnable PE :
  - Continuous ordering with sin and cos functions.
  - Advantages :
    - No training necessary
    - No need to know the maximum length in the train set.
    - Test sequences may have lengths not present in the train set.

$\mathrm{PE}_t \in \mathbb{R}^d$ is defined as

$$\mathrm{PE}_{t,i} = \begin{cases} \sin(2\pi f_i t) \text{ if } i \text{ is even,} \\ \cos(2\pi f_i t) \text{ if } i \text{ is odd,} \end{cases} \quad \text{with } f_i = \frac{10,000^{\frac{d}{\lfloor 2i \rfloor}}}{2\pi}.$$

with $\|\mathrm{PE}_t - \mathrm{PE}_{t'}\|_2 \propto \mathrm{Dist}(t,t') = |t-t'| \in \mathbb{R}_+$

because these PEs are the eigenvectors of the line.



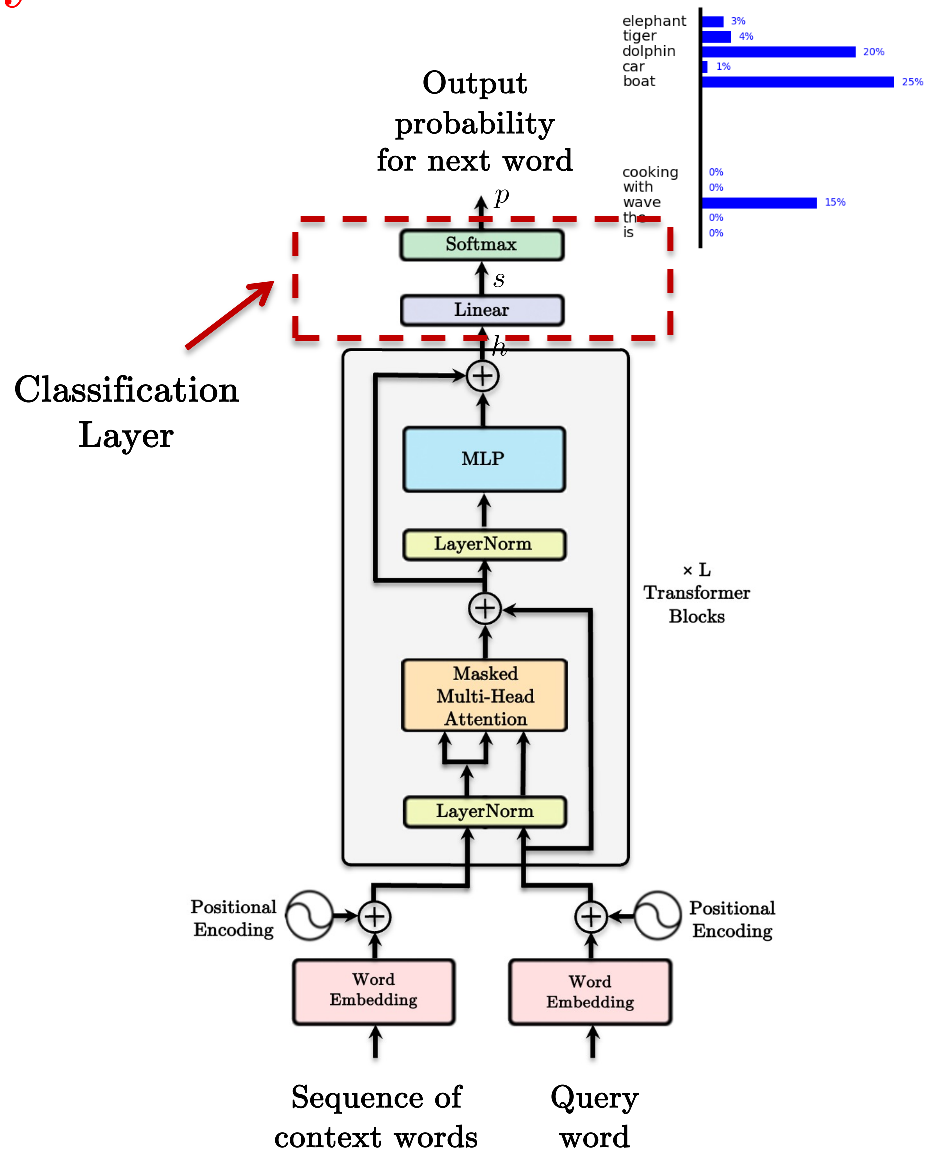
Classification layer

- The last layer is a standard linear layer to compute the scores of the next word in the sequence, followed by a Softmax function to produce the probability vector.

Linear layer

$$s = \mathrm{LL}(h) \in \mathbb{R}^V$$

$$p = \mathrm{Softmax}(s) \in \mathbb{R}^V$$


Output probability for next word

Classification Layer

Sequence of context words

Query word

Efficient training

- It is possible to train in parallel the prediction of the next words in a sequence.
- For this, we need to hide the future words with a masked attention matrix.
- Here is the process :
  - Step 1 : Compute the attention matrix.
  - Step 2 : Mask next word to predict.
  - Step 3 : Softmax calculation.
  - Step 4 : Calculate weighted linear combination.

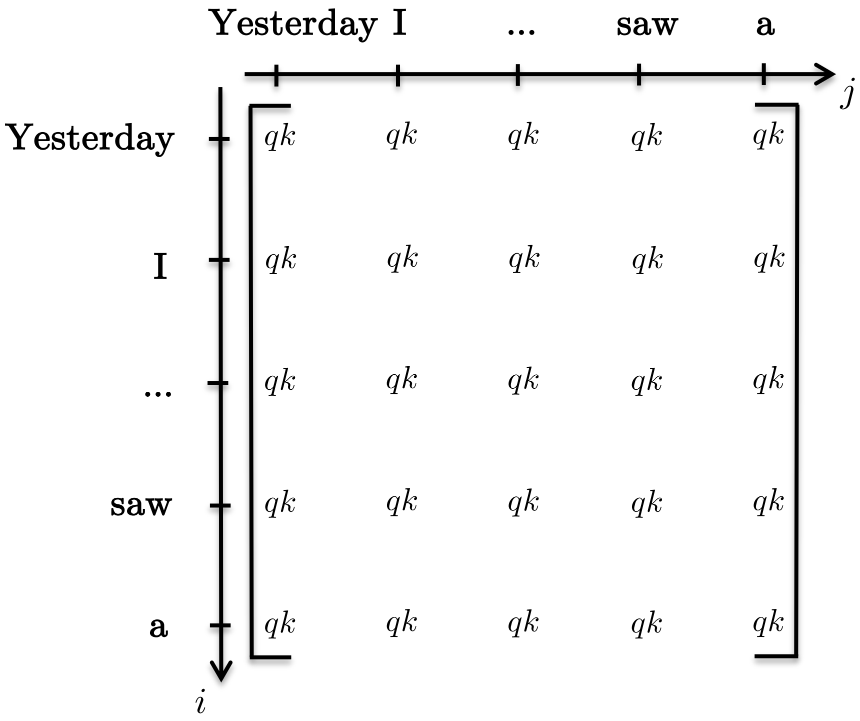
$$\text{Mask-HA}(Q,K,V) = \text{Softmax}\Big(\frac{QK^T}{\sqrt{d}} \odot \text{Mask}\Big)V,$$

$$\text{with } Q,K,V \in \mathbb{R}^{L\times d}, \text{Mask} \in \mathbb{R}^{L\times L},$$

$$\text{and } \text{Mask}_{ij} = \begin{cases} 1 & \text{if attention between } i \text{ and } j \\ -\infty & \text{if no attention} \end{cases}$$

Attention matrix

- Step 1 : Compute the attention matrix A, i.e. A_{ij} is the dot product between word vector q_i and word vector k_j.
- Two similar vectors will receive a high value and inversely, two dissimilar ones a low value.



$$\begin{array}{c|ccccc}
 & \text{Yesterday} & \text{I} & \text{...} & \text{saw} & \text{a} \\
\hline
\text{Yesterday} & qk & qk & qk & qk & qk \\
\text{I} & qk & qk & qk & qk & qk \\
\text{...} & qk & qk & qk & qk & qk \\
\text{saw} & qk & qk & qk & qk & qk \\
\text{a} & qk & qk & qk & qk & qk
\end{array}$$

$$A = QK^T \in \mathbb{R}^{L \times L}$$

$$A_{ij} = q_i^T k_j \in \mathbb{R}$$

Masked attention

- Step 2 : Mask next word to predict

$$\begin{array}{c|ccccc} & \text{Yesterday} & \text{I} & \dots & \text{saw} & \text{a} \\ \hline \text{Yesterday} & qk & qk & qk & qk & qk \\ \text{I} & qk & qk & qk & qk & qk \\ \dots & qk & qk & qk & qk & qk \\ \text{saw} & qk & qk & qk & qk & qk \\ \text{a} & qk & qk & qk & qk & qk \end{array} \quad j, \; i$$

$$A = QK^T \in \mathbb{R}^{L \times L}$$

$$\odot$$

Pointwise multiplication

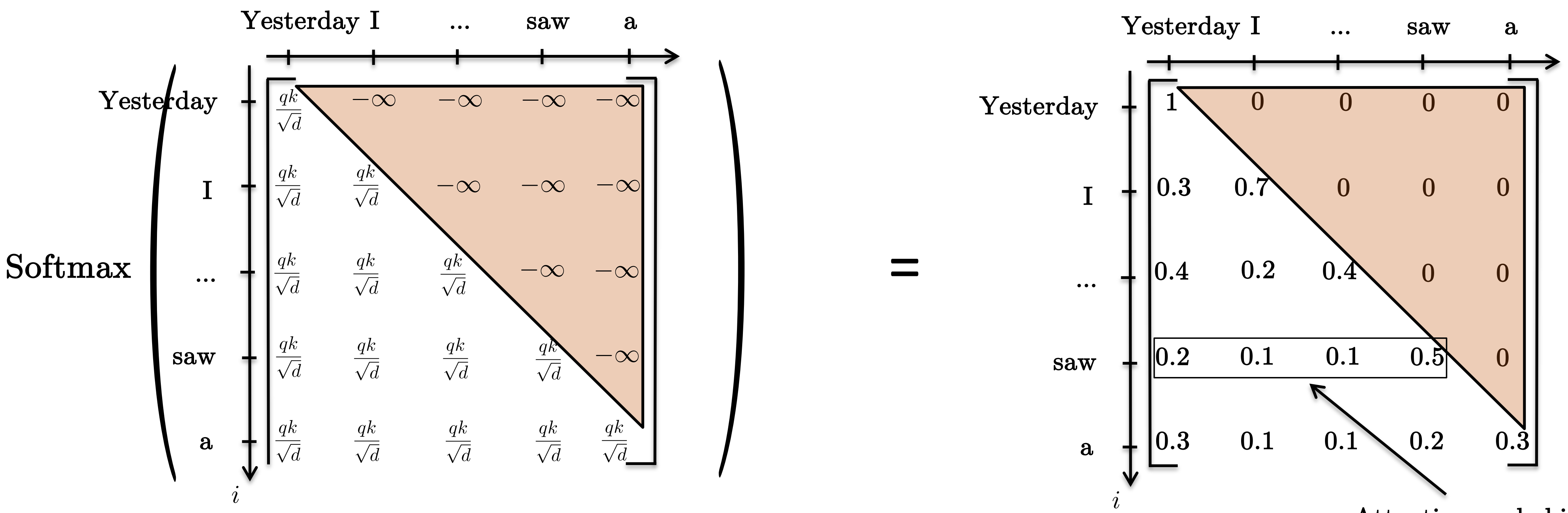
Hide these words during next word prediction

$$\begin{array}{c|ccccc} & \text{Yesterday} & \text{I} & \dots & \text{saw} & \text{a} \\ \hline \text{Yesterday} & 1 & -\infty & -\infty & -\infty & -\infty \\ \text{I} & 1 & 1 & -\infty & -\infty & -\infty \\ \dots & 1 & 1 & 1 & -\infty & -\infty \\ \text{saw} & 1 & 1 & 1 & 1 & -\infty \\ \text{a} & 1 & 1 & 1 & 1 & 1 \end{array} \quad i$$

$$\text{Mask}_{ij} = \begin{cases} 1 & \text{if attention between } i \text{ and } j \\ -\infty & \text{if no attention} \end{cases}$$

Masked attention

- Step 3 : Softmax calculation



$$\text{Softmax}\left(\begin{array}{c|ccccc} & \text{Yesterday} & \text{I} & \dots & \text{saw} & \text{a} \\ \hline \text{Yesterday} & \frac{qk}{\sqrt{d}} & -\infty & -\infty & -\infty & -\infty \\ \text{I} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & -\infty & -\infty & -\infty \\ \dots & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & -\infty & -\infty \\ \text{saw} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & -\infty \\ \text{a} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} & \frac{qk}{\sqrt{d}} \end{array}\right) = \begin{array}{c|ccccc} & \text{Yesterday} & \text{I} & \dots & \text{saw} & \text{a} \\ \hline \text{Yesterday} & 1 & 0 & 0 & 0 & 0 \\ \text{I} & 0.3 & 0.7 & 0 & 0 & 0 \\ \dots & 0.4 & 0.2 & 0.4 & 0 & 0 \\ \text{saw} & \boxed{0.2 \quad 0.1 \quad 0.1 \quad 0.5} & & & & 0 \\ \text{a} & 0.3 & 0.1 & 0.1 & 0.2 & 0.3 \end{array}$$

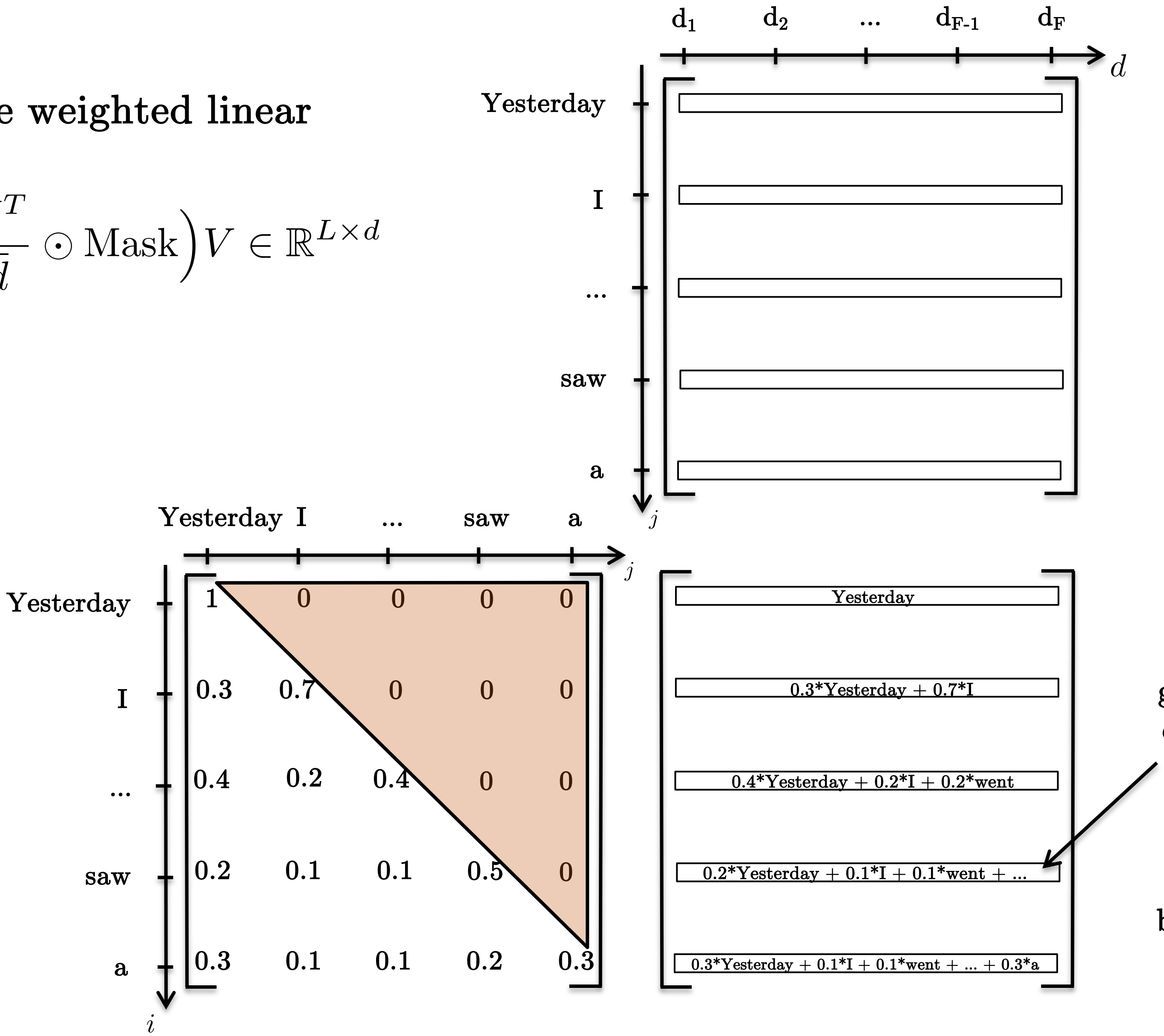
Attention probability between “saw” and “yesterday”, “I”,...”saw”.

$$\text{Softmax}_{\text{row}}\left(\frac{QK^T}{\sqrt{d}} \odot \text{Mask}\right) \in \mathbb{R}^{L\times L}$$

Updated word vectors

- **Step 4 : Calculate weighted linear combination**

$$\text{Softmax}_{\text{row}}\Big(\frac{QK^T}{\sqrt{d}}\odot\text{Mask}\Big)V\in\mathbb{R}^{L\times d}$$



$$\begin{array}{c|ccccc}
 & \text{Yesterday} & \text{I} & \dots & \text{saw} & \text{a} \\
\hline
\text{Yesterday} & 1 & 0 & 0 & 0 & 0 \\
\text{I} & 0.3 & 0.7 & 0 & 0 & 0 \\
\dots & 0.4 & 0.2 & 0.4 & 0 & 0 \\
\text{saw} & 0.2 & 0.1 & 0.1 & 0.5 & 0 \\
\text{a} & 0.3 & 0.1 & 0.1 & 0.2 & 0.3
\end{array}$$

Yesterday

0.3*Yesterday + 0.7*I

0.4*Yesterday + 0.2*I + 0.2*went

0.2*Yesterday + 0.1*I + 0.1*went + ...

0.3*Yesterday + 0.1*I + 0.1*went + ... + 0.3*a

The new vector representation of “saw” is given by a weighted linear combination of the vector representations of “yesterday”, “I”,...”saw”. And the weights are the attention probabilities between the pair of words.

Masked transformer blocks

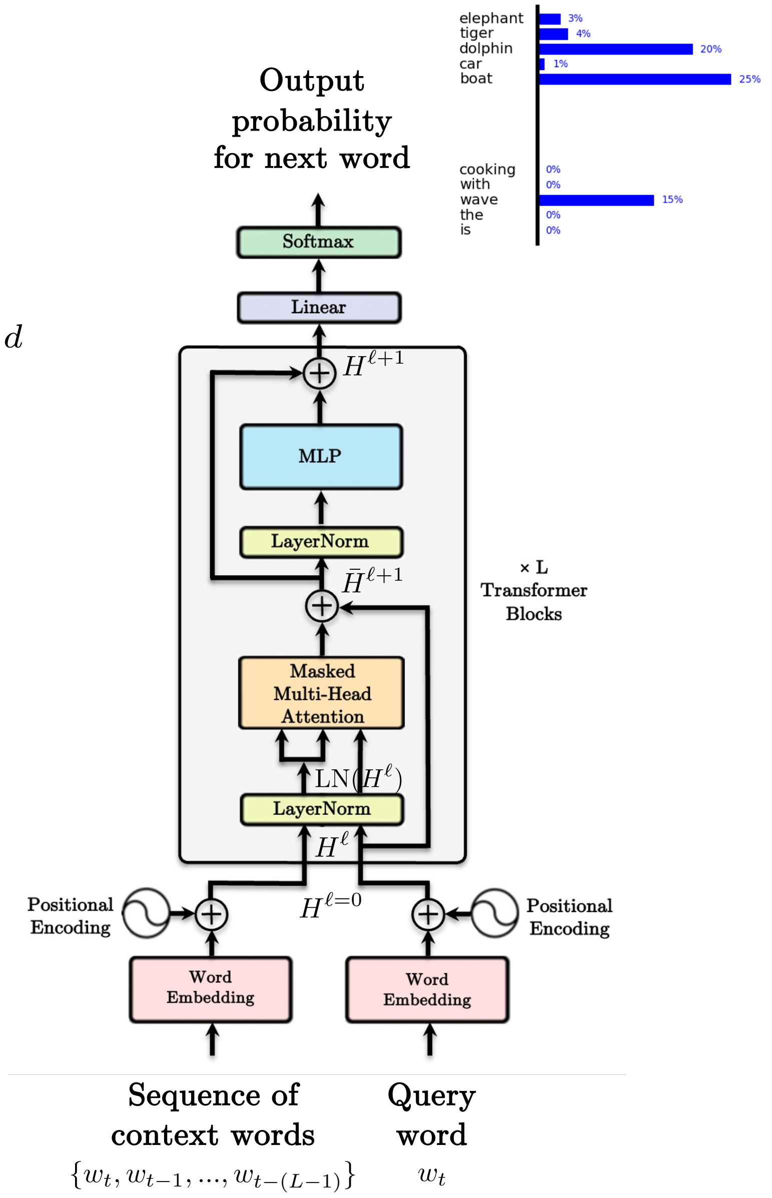
- Stack multiple masked transformer blocks :

For $\ell = 0, 1, 2, .., L-1$

$$\bar{H}^{\ell+1} = H^{\ell} + \text{Mask-MHA}(\text{LN}(H^{\ell}), \text{LN}(H^{\ell}), \text{LN}(H^{\ell})) \in \mathbb{R}^{L \times d}$$

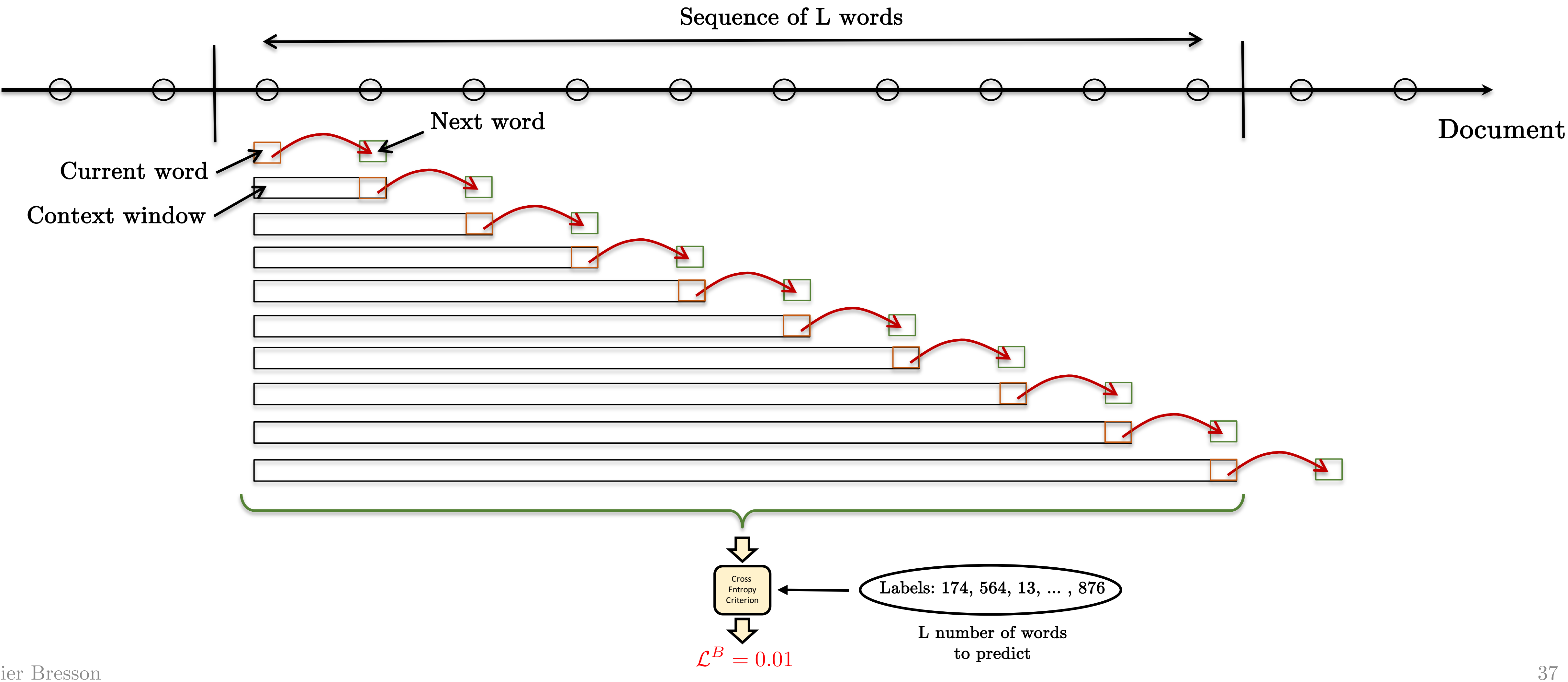
$$H^{\ell+1} = \bar{H}^{\ell+1} + \text{MLP}(\text{LN}(\bar{H}^{\ell+1})) \in \mathbb{R}^{L \times d}$$

with $H^{\ell=0} = \text{LL}(\{w_t, w_{t-1}, ..., w_{t-(L-1)}\}) + \text{PE} \in \mathbb{R}^{L \times d}$



Training with mini-batch

- Training is done with a batch size of B sequences and batch length of L.



Sequence of L words

Next word

Document

Current word

Context window

Cross Entropy Criterion

Labels: 174, 564, 13, ... , 876

L number of words to predict

$\mathcal{L}^B = 0.01$

Generation

- The transformer network is trained in parallel (using the mask to hide the predicted words).
- After training, the mask is not required anymore (there are no future words to hide).
- And the sequence is generated auto-regressively, i.e. one word at a time.

Generate the next word w_{t+1} as follows:

Compute output of Transformer net:

for $\ell = 0, \ldots, L-1$

$$\bar{q}^{\ell+1} = q^{\ell} + \mathrm{MHA}(\mathrm{LN}(q^{\ell}), \mathrm{LN}(K^{\ell}), \mathrm{LN}(V^{\ell})) \in \mathbb{R}^d$$

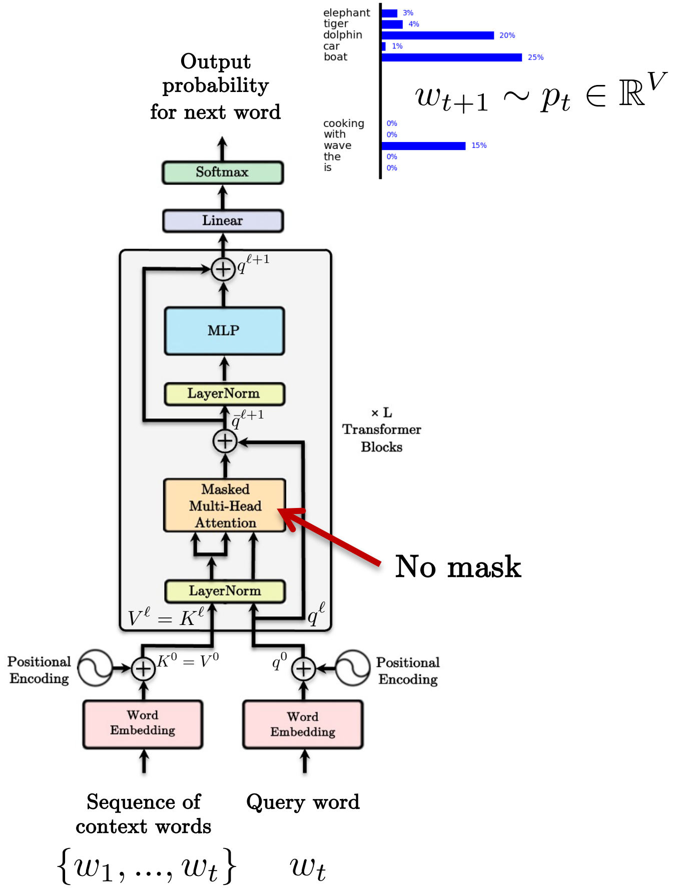
$$q^{\ell+1} = \bar{q}^{\ell+1} + \mathrm{MLP}(\mathrm{LN}(\bar{q}^{\ell+1})) \in \mathbb{R}^d$$

with $q^{\ell=0} = \mathrm{LL}(w_t) + \mathrm{PE}_t \in \mathbb{R}^d$

$$K^{\ell=0} = V^{\ell=0} = \mathrm{LL}(\{w_1, \ldots, w_t\}) + \mathrm{PE} \in \mathbb{R}^{t \times d}$$

Get output probability $p_t = \mathrm{Softmax}(\mathrm{LL}(q^{\ell=L})) \in \mathbb{R}^V$

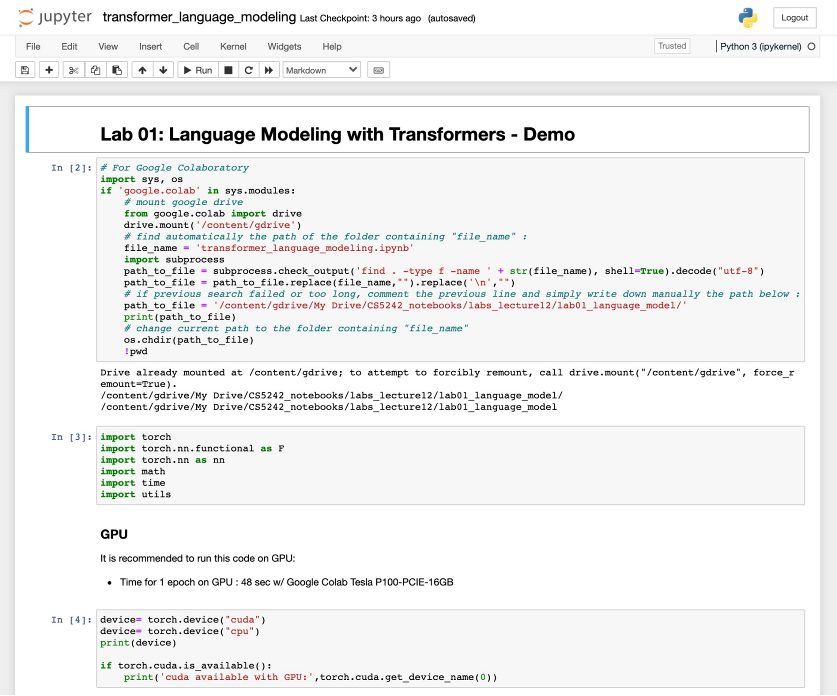
Sample next word probability $w_{t+1} \sim p_t$.



No mask

Lab 01

- PyTorch implementation of Language Model Transformers



Lab 01: Language Modeling with Transformers - Demo

In [2]:
```
# For Google Colaboratory
import sys, os
if 'google.colab' in sys.modules:
    # mount google drive
    from google.colab import drive
    drive.mount('/content/gdrive')
    # find automatically the path of the folder containing "file_name" :
    file_name = 'transformer_language_modeling.ipynb'
    import subprocess
    path_to_file = subprocess.check_output('find . -type f -name ' + str(file_name), shell=True).decode("utf-8")
    path_to_file = path_to_file.replace(file_name,"").replace('\n',"")
    # if previous search failed or too long, comment the previous line and simply write down manually the path below :
    path_to_file = '/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab01_language_model/'
    print(path_to_file)
    # change current path to the folder containing "file_name"
    os.chdir(path_to_file)
    !pwd
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_r
emount=True).
/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab01_language_model/
/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab01_language_model
```

In [3]:
```
import torch
import torch.nn.functional as F
import torch.nn as nn
import math
import time
import utils
```

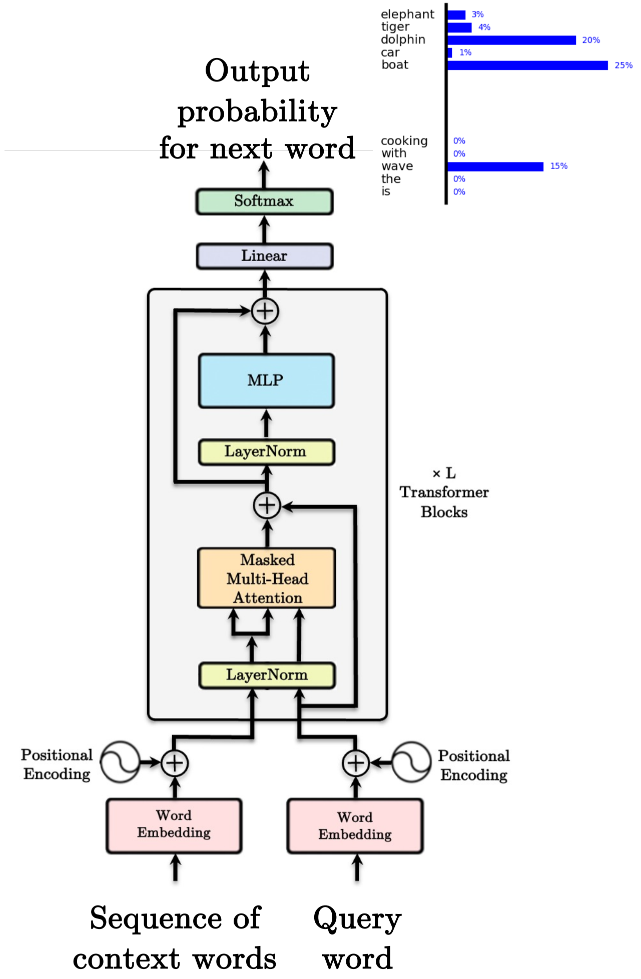
GPU

It is recommended to run this code on GPU:

- Time for 1 epoch on GPU : 48 sec w/ Google Colab Tesla P100-PCIE-16GB

In [4]:
```
device= torch.device("cuda")
device= torch.device("cpu")
print(device)

if torch.cuda.is_available():
    print('cuda available with GPU:',torch.cuda.get_device_name(0))
```



Lab 01

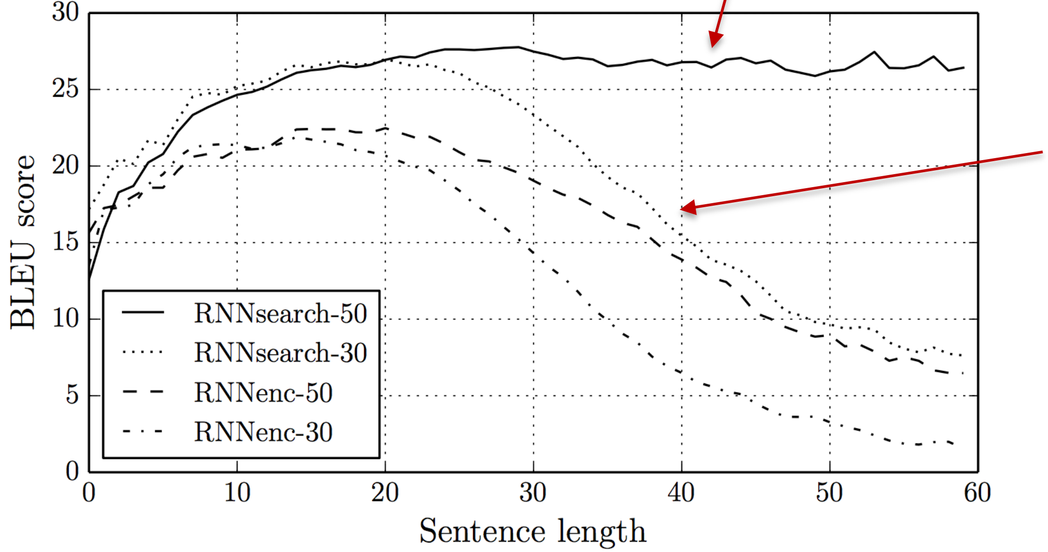
- Numerical results on PTB :
  - Vanilla RNN: $\exp(\text{train\_loss}) = 111$ $\exp(\text{test\_loss}) = 155$ 3 million parameters
  - LSTM: $\exp(\text{train\_loss}) = 59$ $\exp(\text{test\_loss}) = 106$ 7 million parameters
  - LSTM state-of-the-art: $\exp(\text{test\_loss}) = 50$ 50 million parameters
  - Vanilla LM Transformer:

    $\exp(\text{train\_loss}) = 54$ $\exp(\text{test\_loss}) = 174$ 3 million parameters
  - LM Transformer state-of-the-art (with pre-training):

    $\exp(\text{test\_loss}) = 20.5$ 175 billion parameters

Attention mechanism

- It is a breakthrough idea in NLP !
- It is as revolutionary as CNNs in Computer Vision.



Transformers

RNNs
(degrades quickly
after 30 words)

(Bahdanau-Cho-Bengio 2014)

Why attention nets are better?

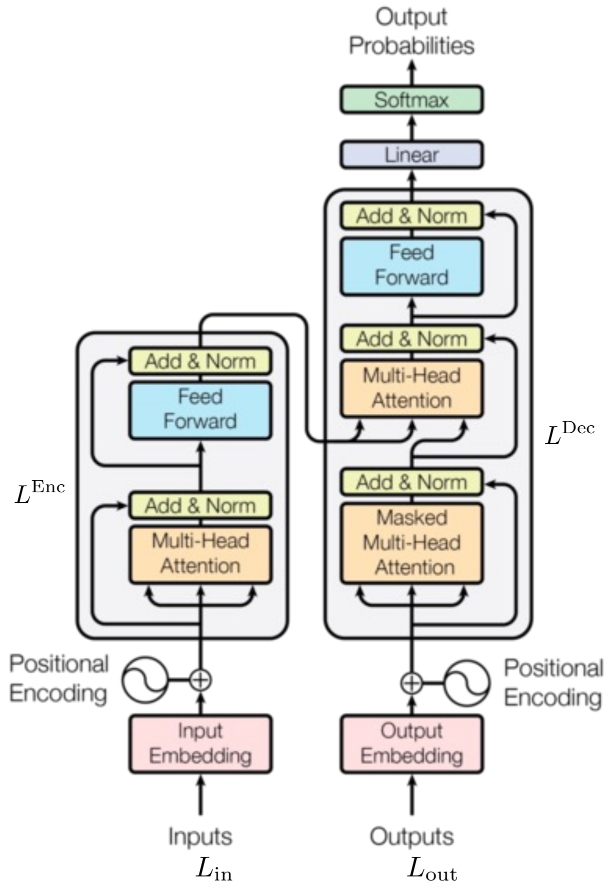
- Why casting the LM task as a soft alignment/attention problem is a great idea ?
- With RNNs, the very long sequence requires to be memorized and represented by a single vector.
- RNN architectures with a single memory vector cannot simply deal with long sequences (limit of non-linear dynamic systems).
- We ask too much to memorize everything with one vector!
- With attention, we distribute the memorization load over each pair of words.
- Each word in the target sequence only needs to find its match with the word (or a few words) in the source target.
- It solves the limitation of long-term dependencies in RNNs (a word in the target sequence communicates with all words in the source sequence).
- The matching is made easy by transforming the words with hidden representations.
- Attention is a key mathematical structure for NLP and several other domains.
- SOTA for all NLP tasks since 2019.

Outline

- Language Models
- Memory Networks
- Transformers
- Language Model Transformers
- **Sequence-To-Sequence Transformers**
- Transfer Learning
- Conclusion

Seq2Seq Transformers

- Given a sequence of words, convert it into a different sequence.
- Basic tasks in NLP
  - Translation
  - Question & Answer
  - Summarization
- A Seq2Sep Transformer is composed of
  - World embedding layer
  - Self-attention encoding layer
  - Self-attention decoding layer
  - Cross-attention layer
  - Classification layer



Input(EN): He drives the red car.

Output(ES): El conduce el coche rojoe.

Seq2Seq Transformer Architecture (Vaswani-et-al Google Brain 2017)

Self-attention encoder

- Encode the complete input sequence with a self-attention layer :

$$\bar{H} = \text{MHA}(H) \in \mathbb{R}^{L_{\text{in}}\times d},\ \ H \in \mathbb{R}^{L_{\text{in}}\times d}$$

$$= \Big(\|_{h=1}^{H}\ \text{HA}_h(H)\Big)W^O,\ \ W^O \in \mathbb{R}^{d\times d}$$

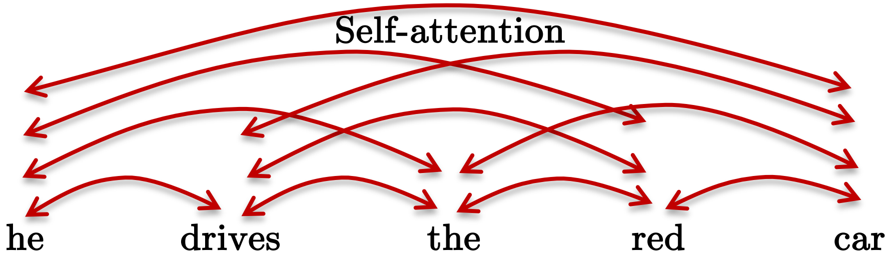
$$\text{with } H = \underbrace{\text{LL}(\{w_1^{\text{in}},...,w_{L_{\text{in}}}^{\text{in}}\})}_{\text{Word embeddings}} \in \mathbb{R}^{L_{\text{in}}\times d}$$

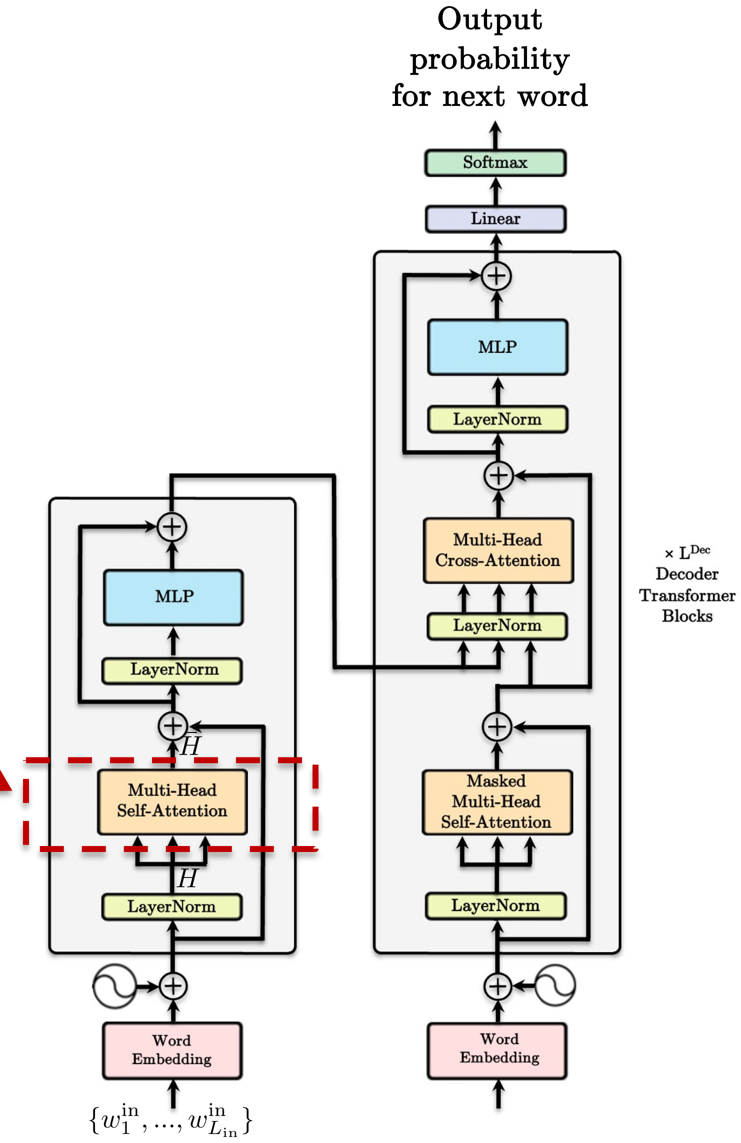
$$\text{HA}_h(H) = \text{Softmax}\Big(\frac{Q_h K_h^T}{\sqrt{d/H}}\Big)V_h \in \mathbb{R}^{L_{\text{in}}\times d/H}$$

$$\text{with } Q_h = HW_h^Q \in \mathbb{R}^{L_{\text{in}}\times d/H}, W_h^Q \in \mathbb{R}^{d\times d/H}$$

$$K_h = HW_h^K \in \mathbb{R}^{L_{\text{in}}\times d/H}, W_h^K \in \mathbb{R}^{d\times d/H}$$

$$V_h = HW_h^V \in \mathbb{R}^{L_{\text{in}}\times d/H}, W_h^V \in \mathbb{R}^{d\times d/H}$$




Output probability for next word
Softmax
Linear
MLP
LayerNorm
Multi-Head Cross-Attention
LayerNorm
Masked Multi-Head Self-Attention
LayerNorm
Word Embedding
× L^Dec Decoder Transformer Blocks
MLP
LayerNorm
$\bar{H}$
Multi-Head Self-Attention
H
LayerNorm
Word Embedding
$\{w_1^{\text{in}},...,w_{L_{\text{in}}}^{\text{in}}\}$

Self-attention encoding layer

Produce a representation of words that depends on the context of surrounding words.

Full encoder layer

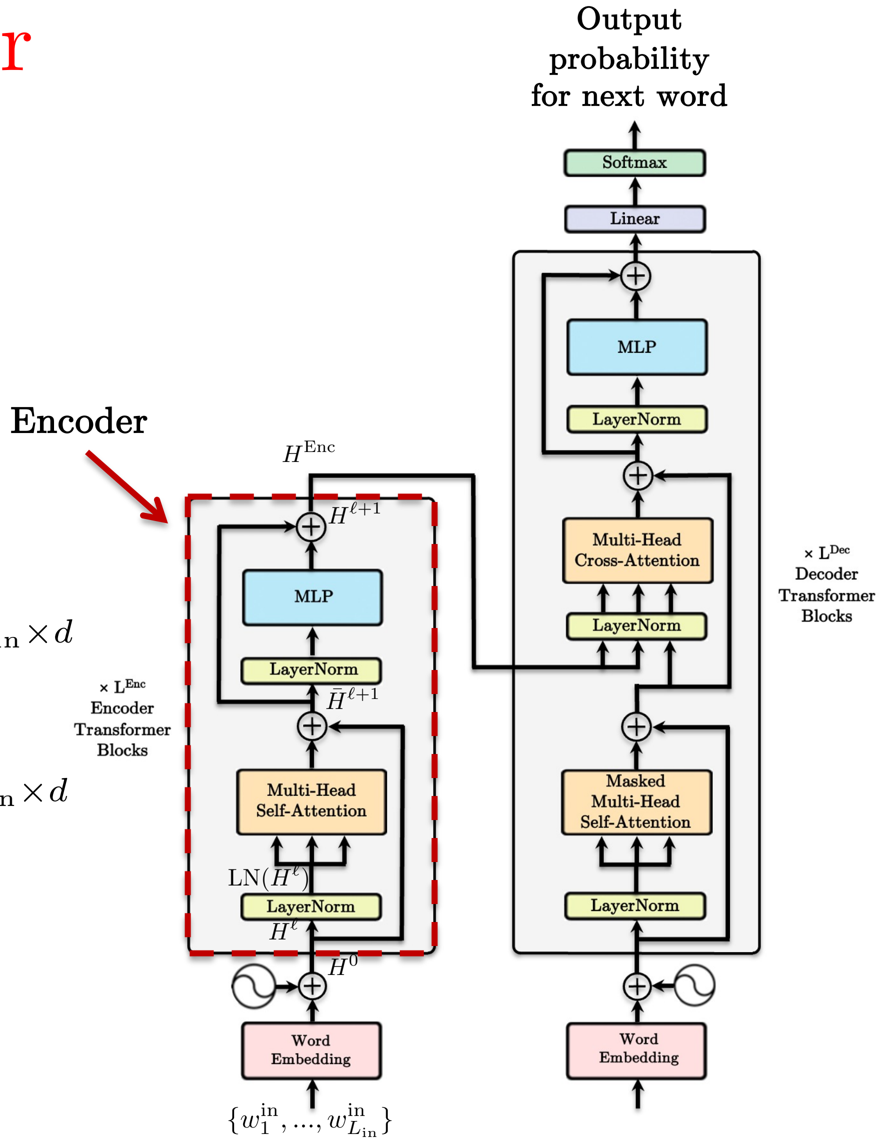
- The encoder is composed of multiple attention blocks.
- Each attention block has MHA, MLP, Residual Connection, Layer Normalization, PE.

$$
\begin{aligned}
&\text{for } \ell = 0, \dots, L^{\text{Enc}} - 1 \\
&\qquad \bar{H}^{\ell+1} = H^{\ell} + \text{MHA}(\text{LN}(H^{\ell}), \text{LN}(H^{\ell}), \text{LN}(H^{\ell})) \in \mathbb{R}^{L_{\text{in}} \times d} \\
&\qquad H^{\ell+1} = \bar{H}^{\ell+1} + \text{MLP}(\text{LN}(\bar{H}^{\ell+1})) \in \mathbb{R}^{L_{\text{in}} \times d} \\
&\text{with initialization } H^{\ell=0} = \text{LL}(\{w_1^{\text{in}}, \dots, w_{L_{\text{in}}}^{\text{in}}\}) + \text{PE} \in \mathbb{R}^{L_{\text{in}} \times d}
\end{aligned}
$$

Linear layer (word embedding)

Positional Encoding

$$\text{and encoder output } H^{\text{Enc}} = H^{\ell=L^{\text{Enc}}} \in \mathbb{R}^{L_{\text{in}} \times d}$$

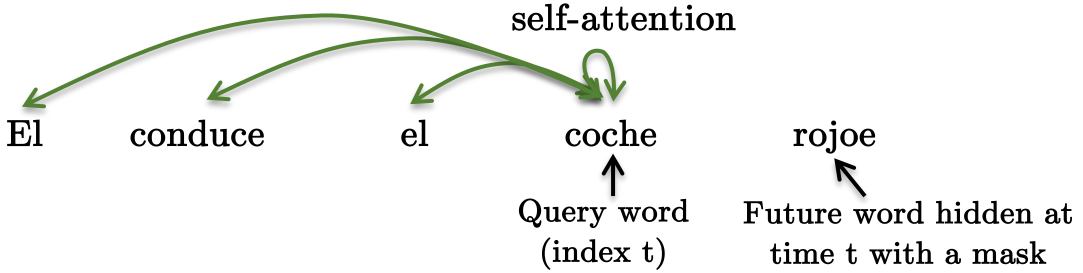

Output probability for next word
Softmax
Linear
MLP
LayerNorm
Multi-Head Cross-Attention
LayerNorm
Masked Multi-Head Self-Attention
LayerNorm
Word Embedding
× L^Dec Decoder Transformer Blocks
Encoder
H^{Enc}
$H^{\ell+1}$
MLP
LayerNorm
$\bar{H}^{\ell+1}$
Multi-Head Self-Attention
$\text{LN}(H^{\ell})$
LayerNorm
H^{ℓ}
H^0
Word Embedding
$\{w_1^{\text{in}}, \dots, w_{L_{\text{in}}}^{\text{in}}\}$
× L^Enc Encoder Transformer Blocks

Masked self-attention decoder

- Compute the query to the next word to predict with a masked self-attention layer to speed up training.
- The mask hides future words in the output sequence, i.e. the words we want to predict.
- Update equation for the masked self-attention decoder layer :

$$\bar{H} = \text{Mask-MHA}(H, H, H) \in \mathbb{R}^{L_{\text{out}} \times d}$$
$$\text{with } H = \text{LL}(\{w_1^{\text{out}}, ..., w_{L_{\text{out}}}^{\text{out}}\}) + \text{PE} \in \mathbb{R}^{L_{\text{out}} \times d}$$

- Self-attention layer produces a representation of the sequence of output words that depend on the context of their surrounding words.

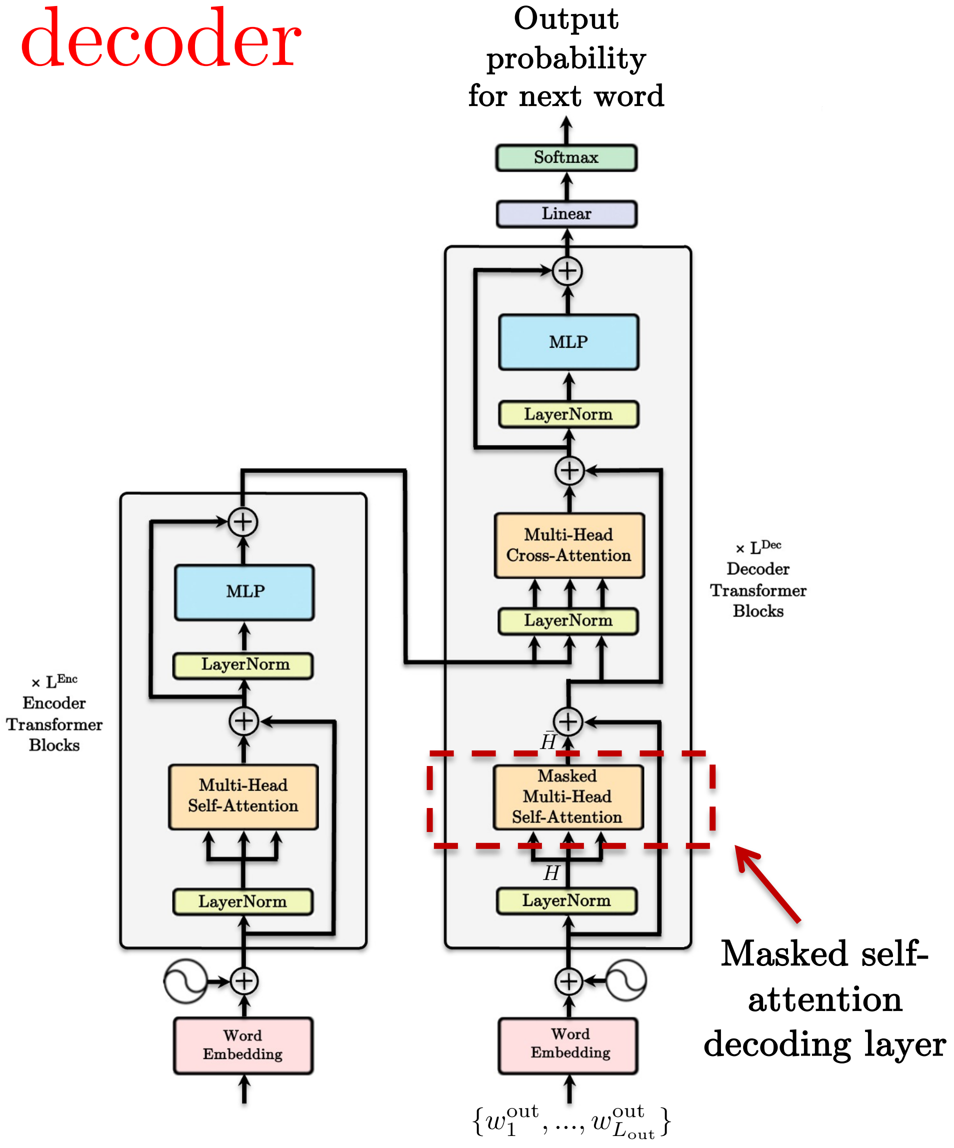


self-attention

El conduce el coche rojoe

Query word (index t)

Future word hidden at time t with a mask



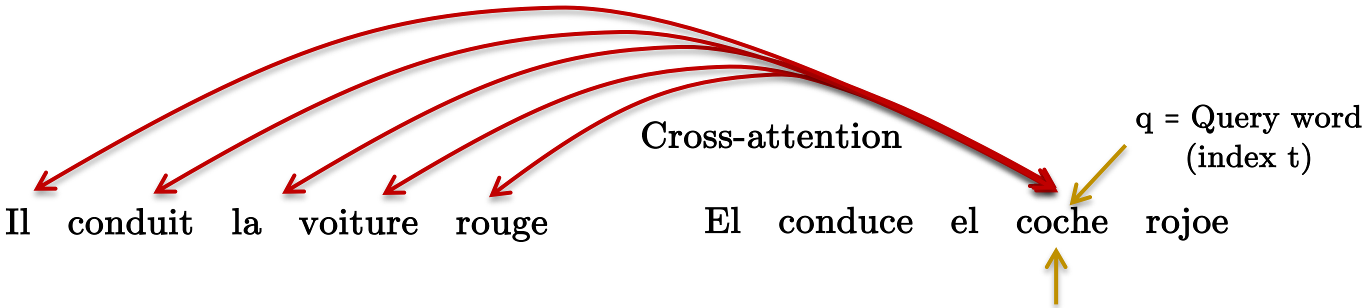
Output probability for next word
Softmax
Linear
MLP
LayerNorm
Multi-Head Cross-Attention
LayerNorm
$\bar{H}$
Masked Multi-Head Self-Attention
H
LayerNorm
Word Embedding
$\{w_1^{\text{out}}, ..., w_{L_{\text{out}}}^{\text{out}}\}$
× L^{Dec} Decoder Transformer Blocks
MLP
LayerNorm
Multi-Head Self-Attention
LayerNorm
Word Embedding
× L^{Enc} Encoder Transformer Blocks

Masked self-attention decoding layer

Cross-attention layer

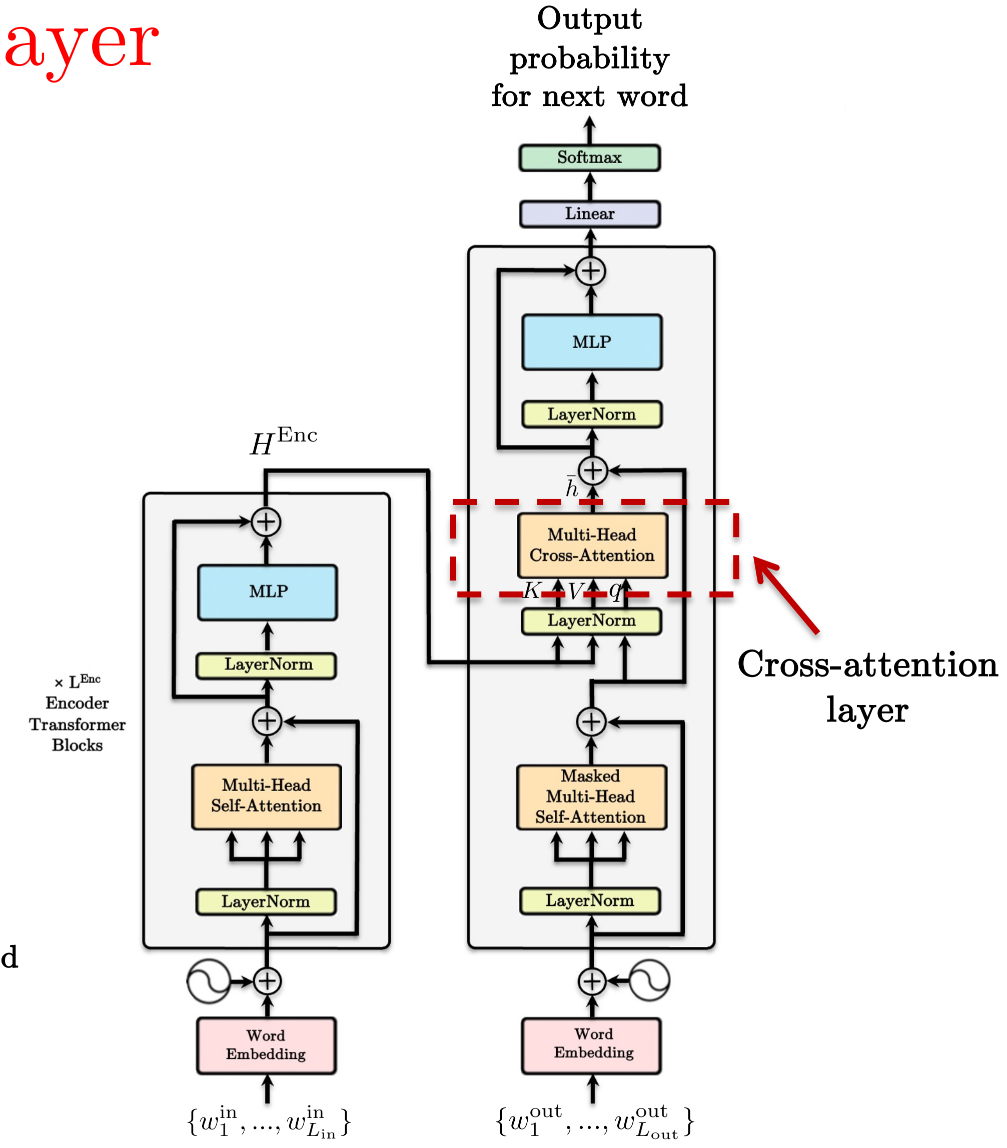
- Compute attention between pairs of words coming from the encoded input sequence and the query from the output sequence.
- Update equation for a single query :

$$\begin{aligned}\bar{h} &= \mathrm{MHA}(q,K,V) \in \mathbb{R}^{d},\ q \in \mathbb{R}^{d}, K,V \in \mathbb{R}^{L_{\text{in}}\times d}\\ &= \Big(\|_{h=1}^{H}\ \mathrm{HA}_h(q,K,V)\Big)W^O,\ W^O \in \mathbb{R}^{d\times d}\\ \text{with } q &= \mathrm{h}_t^{\text{Dec}} \in \mathbb{R}^{d},\ K = V = H^{\text{Enc}} \in \mathbb{R}^{L_{\text{in}}\times d}\end{aligned}$$



Input sequence encoded by self-attention H^{Enc}

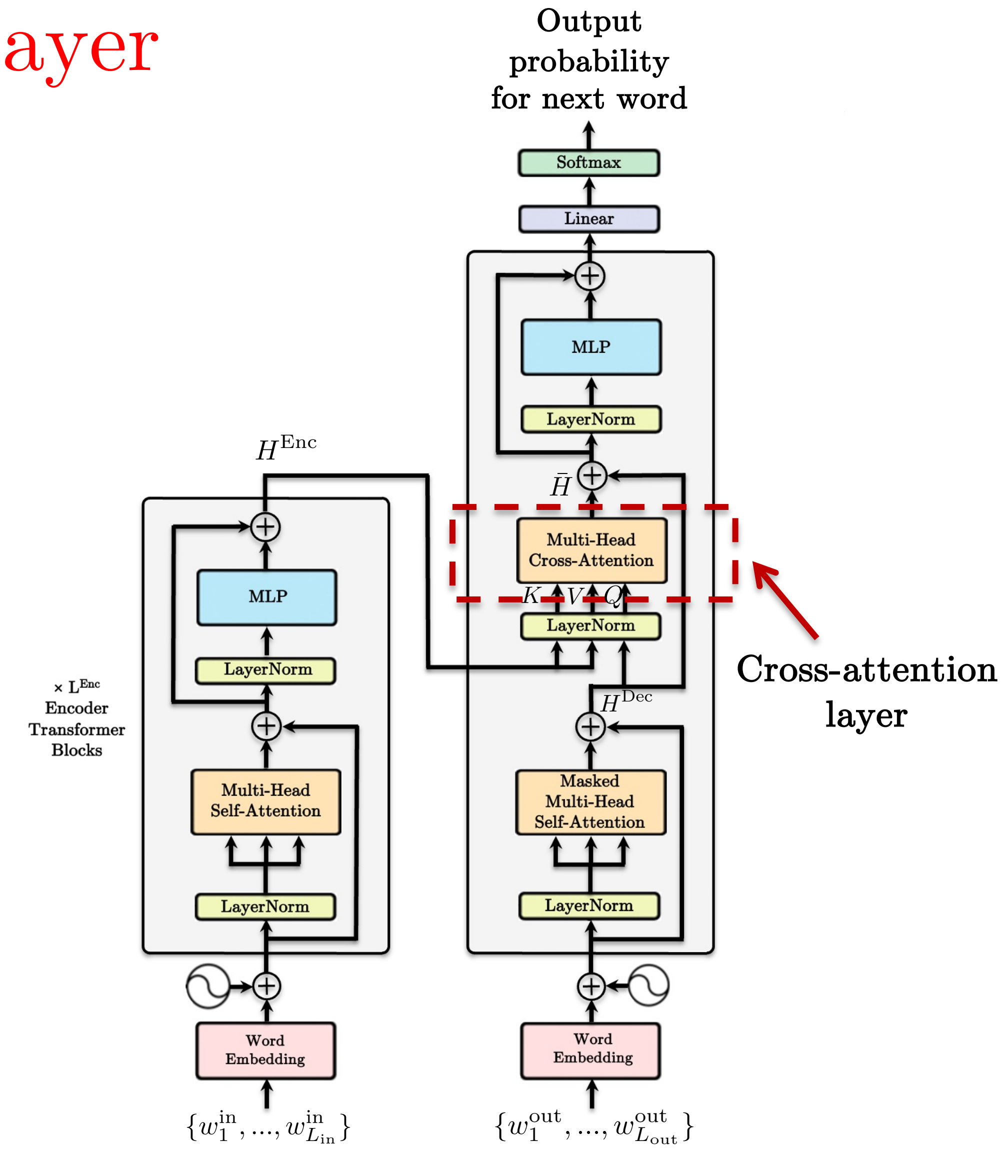
Output word encoded by self-attention $\mathrm{h}_t^{\text{Dec}}$


Output probability for next word
Softmax
Linear
MLP
LayerNorm
$\bar{h}$
Multi-Head Cross-Attention
K V q
LayerNorm
Masked Multi-Head Self-Attention
LayerNorm
Word Embedding
$\{w_1^{\text{out}},\dots,w_{L_{\text{out}}}^{\text{out}}\}$
H^{Enc}
MLP
LayerNorm
Multi-Head Self-Attention
LayerNorm
× L^{Enc} Encoder Transformer Blocks
Word Embedding
$\{w_1^{\text{in}},\dots,w_{L_{\text{in}}}^{\text{in}}\}$
Cross-attention layer

Cross-attention layer

- To speed up training, we compute multiple queries at the same time, i.e. in parallel.
- Update equation for all queries :

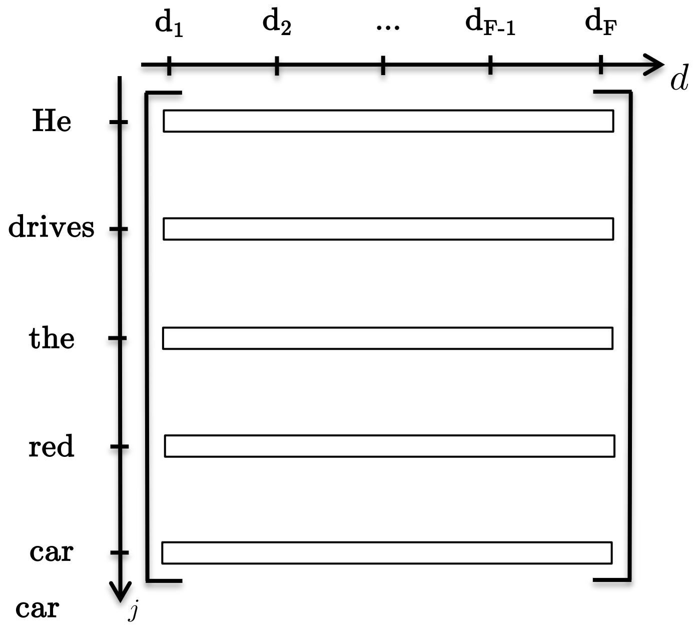
$$\begin{aligned}\bar{H} &= \mathrm{MHA}(Q,K,V) \in \mathbb{R}^{L_{\mathrm{out}}\times d},\ Q \in \mathbb{R}^{L_{\mathrm{out}}\times d}, K,V \in \mathbb{R}^{L_{\mathrm{in}}\times d}\\ &= \Big(\|_{h=1}^{H}\ \mathrm{HA}_h(Q,K,V)\Big)W^O,\ W^O \in \mathbb{R}^{d\times d}\\ &\text{with } Q = H^{\mathrm{Dec}} \in \mathbb{R}^{L_{\mathrm{out}}\times d},\ K=V=H^{\mathrm{Enc}} \in \mathbb{R}^{L_{\mathrm{in}}\times d}\end{aligned}$$



Output probability for next word
Softmax
Linear
MLP
LayerNorm
$\bar{H}$
Multi-Head Cross-Attention
K V Q
LayerNorm
H^{Dec}
Masked Multi-Head Self-Attention
LayerNorm
Word Embedding
$\{w_1^{\mathrm{out}}, ..., w_{L_{\mathrm{out}}}^{\mathrm{out}}\}$
H^{Enc}
MLP
LayerNorm
Multi-Head Self-Attention
LayerNorm
Word Embedding
$\{w_1^{\mathrm{in}}, ..., w_{L_{\mathrm{in}}}^{\mathrm{in}}\}$
× L^{Enc} Encoder Transformer Blocks
Cross-attention layer

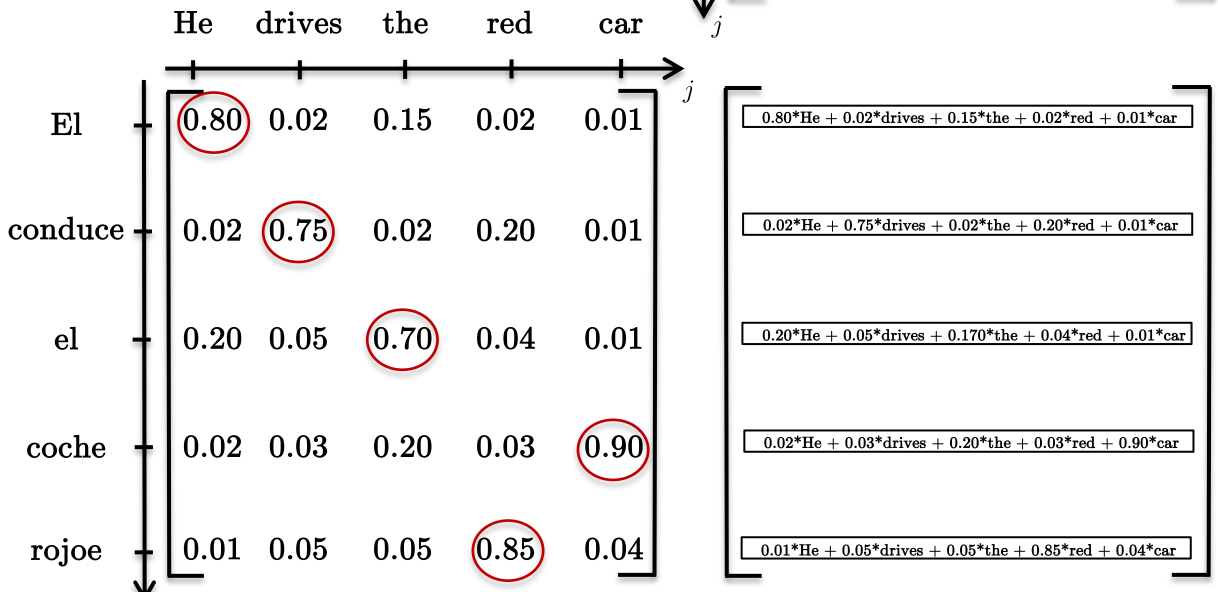
Interpretation

- Cross-attention highlights the matching/alignment between a word in the output sequence and its corresponding word or words in the input sequence.
- Unlike the self-attention mechanism in the decoder layer, the cross-attention layer does not require a mask.
- In fact, applying a mask would be counter-productive, as the query maximizes the prediction of the next word in the output sequence by attending to all tokens in the input sequence.



The new vector representation of "coche" is given by a weighted linear combination of the vector representations of "He", "drives",... "car", where the weights are the attention probabilities between the pair of words.

Here, we have "coche" ≈ "car" as the two words match the same meaning in EN and ES.



	He	drives	the	red	car
El	0.80	0.02	0.15	0.02	0.01
conduce	0.02	0.75	0.02	0.20	0.01
el	0.20	0.05	0.70	0.04	0.01
coche	0.02	0.03	0.20	0.03	0.90
rojoe	0.01	0.05	0.05	0.85	0.04

0.80*He + 0.02*drives + 0.15*the + 0.02*red + 0.01*car

0.02*He + 0.75*drives + 0.02*the + 0.20*red + 0.01*car

0.20*He + 0.05*drives + 0.170*the + 0.04*red + 0.01*car

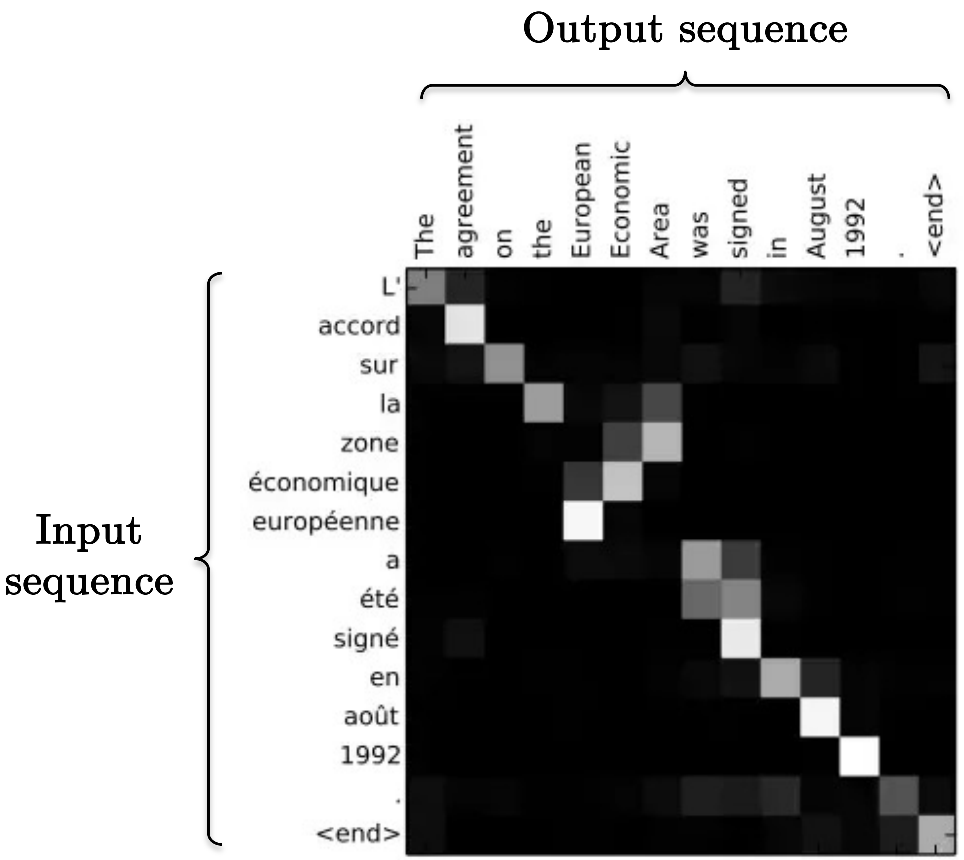
0.02*He + 0.03*drives + 0.20*the + 0.03*red + 0.90*car

0.01*He + 0.05*drives + 0.05*the + 0.85*red + 0.04*car

Cross-attention score matrix

Interpretation

- The cross-attention score matrix provides the matching between words (or tokens) between two sequences (input and output sequences) :



Cross-attention score matrix

$$\text{Softmax}_{\text{row}}\left(\frac{QK^T}{\sqrt{d}}\right) \in \mathbb{R}^{L_{in}\times L_{out}}$$

where

$$Q = H_{in}W^Q \in \mathbb{R}^{L_{in}\times d}$$

$$K = H_{out}W^K \in \mathbb{R}^{L_{out}\times d}$$

Full decoder layer

- Output of the decoder :

$$\text{for } \ell = 0, \ldots, L^{\text{Dec}} - 1$$

$$\bar{H}^{\ell+1} = H^{\ell} + \underbrace{\text{Mask-MHA}(\text{LN}(H^{\ell}), \text{LN}(H^{\ell}), \text{LN}(H^{\ell}))}_{\text{Self-attention}} \in \mathbb{R}^{L_{\text{out}} \times d}$$

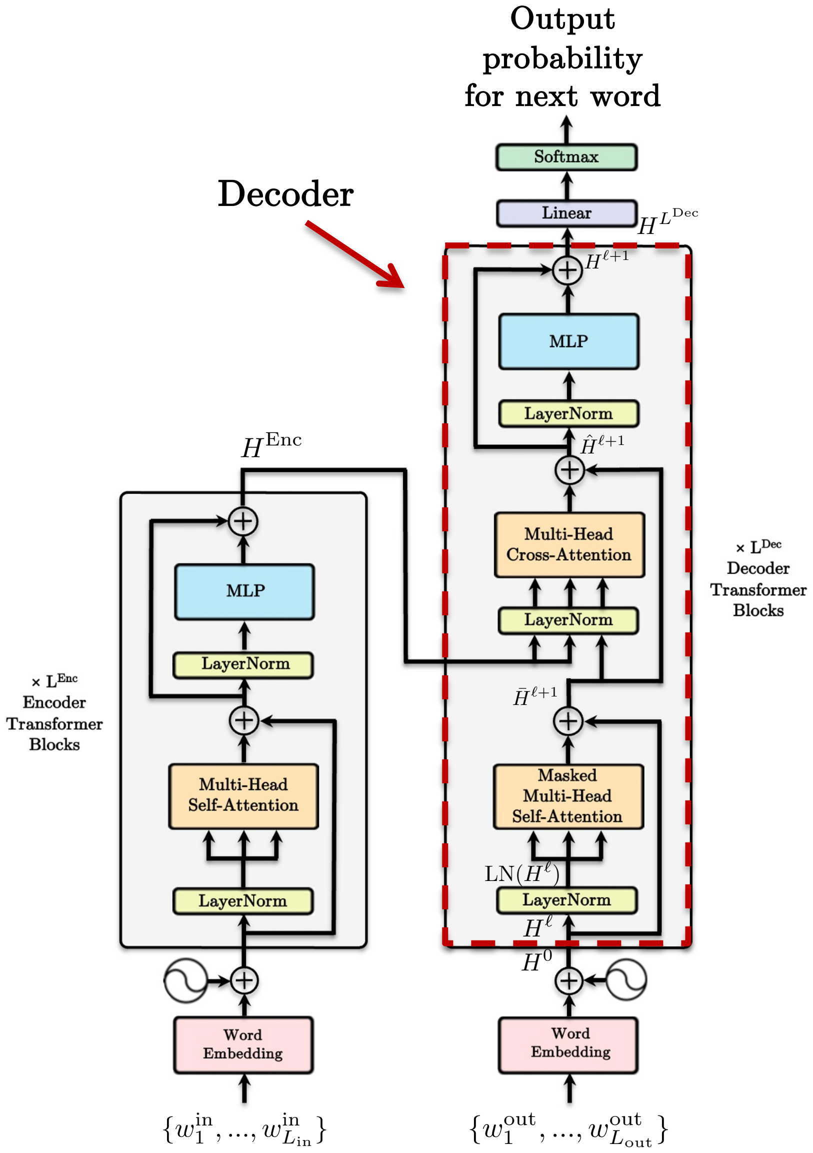
$$\hat{H}^{\ell+1} = \bar{H}^{\ell+1} + \underbrace{\text{MHA}(\text{LN}(\bar{H}^{\ell+1}), \text{LN}(H^{\text{Enc}}), \text{LN}(H^{\text{Enc}}))}_{\text{Cross-attention}} \in \mathbb{R}^{L_{\text{out}} \times d}$$

$$H^{\ell+1} = \hat{H}^{\ell+1} + \text{MLP}(\text{LN}(\hat{H}^{\ell+1})) \in \mathbb{R}^{L_{\text{out}} \times d}$$

$$\text{with } H^{\ell=0} = \text{LL}(\{w_1^{\text{out}}, \ldots, w_{L_{\text{out}}}^{\text{out}}\}) + \text{PE} \in \mathbb{R}^{L_{\text{out}} \times d}$$

$$\text{Decoder output } H^{\ell=L^{\text{Dec}}} \in \mathbb{R}^{L_{\text{out}} \times d}$$

$$\text{Probability output } P = \text{Softmax}(\text{LL}(H^{L^{\text{Dec}}})) \in \mathbb{R}^{L_{\text{out}} \times V}$$



Generation

- At inference, the input sequence is first encoded H^{Enc}.
- Then, the output sequence is generated auto-regressively, i.e. one word at a time (no mask is used).

Generate the next word w_{t+1} as follows:.

Compute output of Transformer model:

for $\ell = 0, ..., L^{\text{Dec}} - 1$

$$\bar{q}^{\ell+1} = q^{\ell} + \text{MHA}(\text{LN}(q^{\ell}), \text{LN}(K^{\ell}), \text{LN}(V^{\ell})) \in \mathbb{R}^d$$

$$\hat{q}^{\ell+1} = \bar{q}^{\ell} + \text{MHA}(\text{LN}(\bar{q})^{\ell}, \text{LN}(H^{\text{Enc}}), \text{LN}(H^{\text{Enc}})) \in \mathbb{R}^d$$

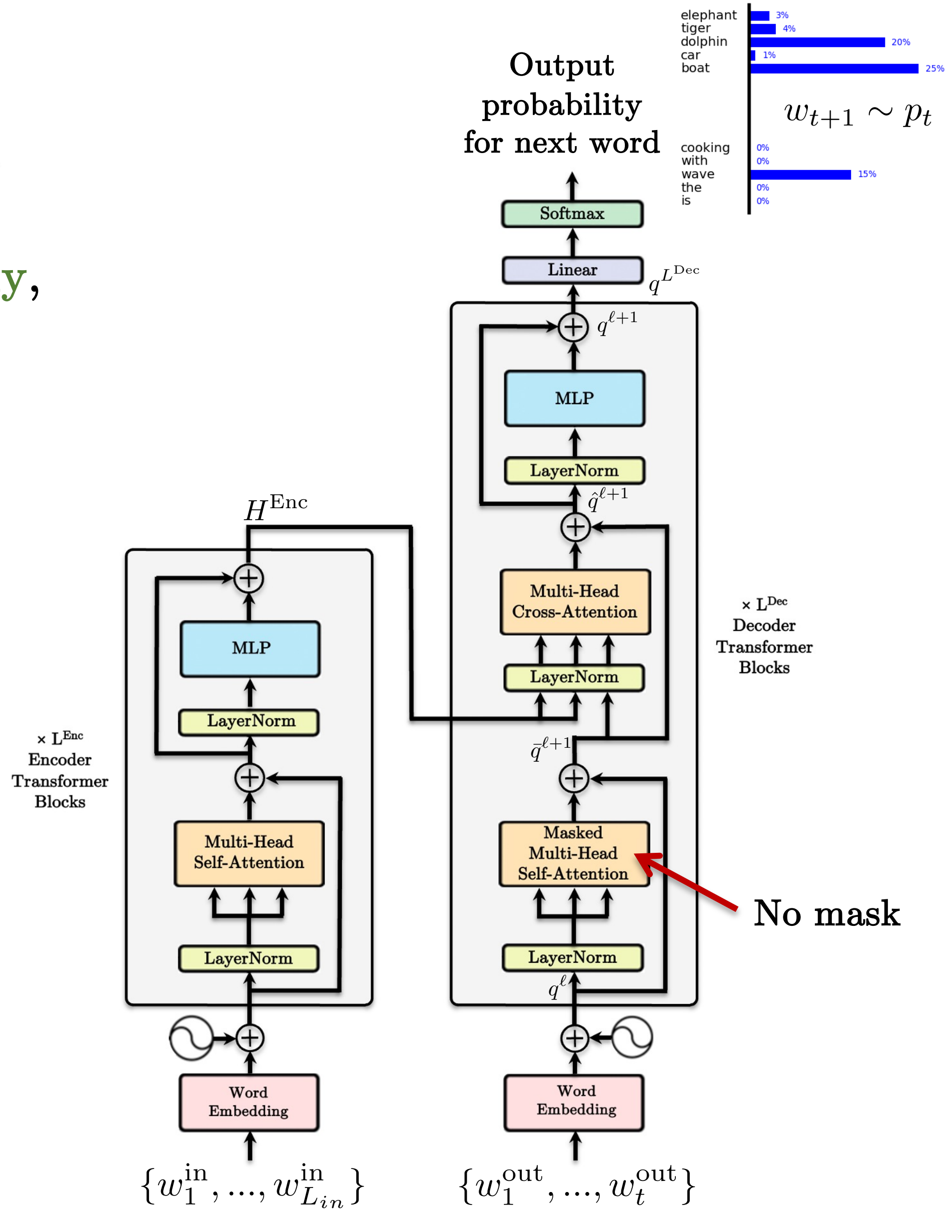
$$q^{\ell+1} = \hat{q}^{\ell+1} + \text{MLP}(\text{LN}(\hat{q}^{\ell+1})) \in \mathbb{R}^d$$

with $q^{\ell=0} = \text{LL}(w_t^{\text{out}}) + \text{PE}_t \in \mathbb{R}^d$

$$K^{\ell=0} = V^{\ell=0} = \text{LL}(\{w_1^{\text{out}}, ..., w_t^{\text{out}}\}) + \text{PE} \in \mathbb{R}^{t \times d}$$

Get output probability $p_t = \text{Softmax}(\text{LL}(q^{\ell=L^{\text{Dec}}})) \in \mathbb{R}^V$

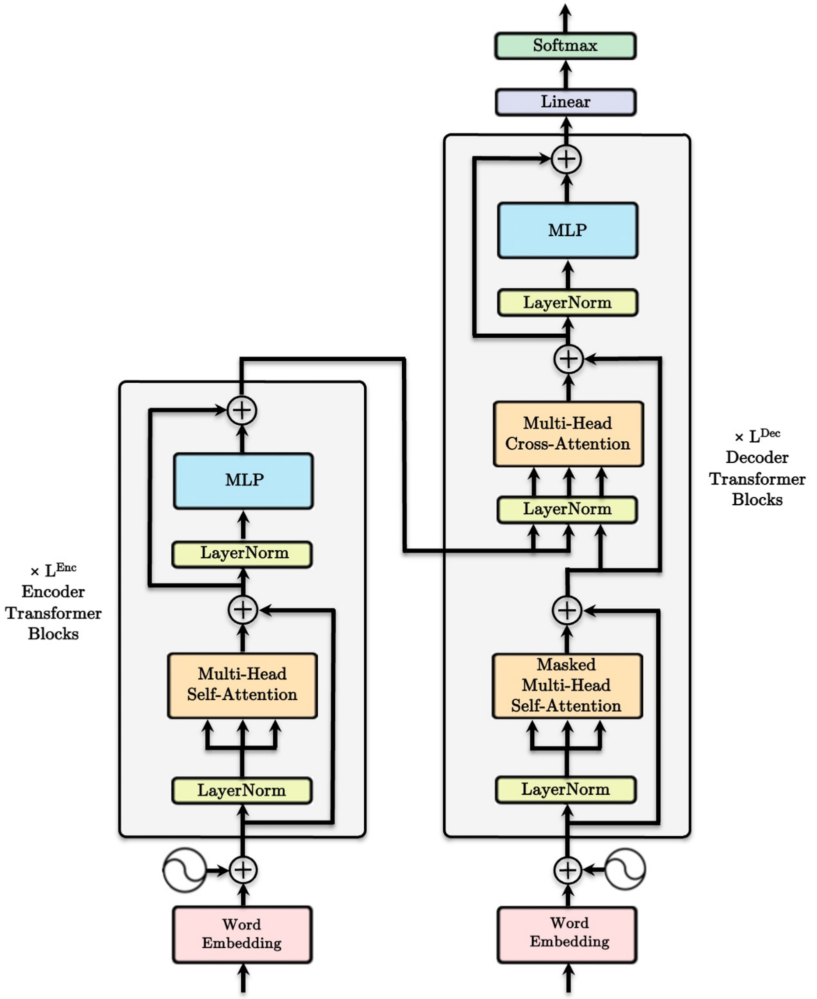
Sample next word probability $w_{t+1} \sim p_t$.



No mask

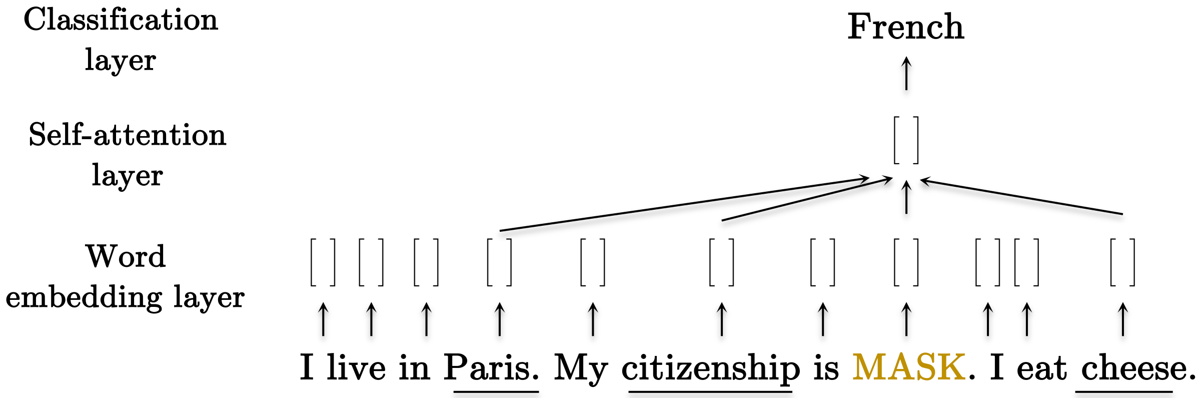
Understanding attention layers

- There are two types of attention layers.
- Self-attention layer
  - Used to represent a word in (learned) context.
- Cross-attention layer
  - Used to query with (learned) matching mechanism.


Softmax
Linear
MLP
LayerNorm
Multi-Head Cross-Attention
LayerNorm
Masked Multi-Head Self-Attention
LayerNorm
Word Embedding
× L^Dec Decoder Transformer Blocks
MLP
LayerNorm
Multi-Head Self-Attention
LayerNorm
Word Embedding
× L^Enc Encoder Transformer Blocks

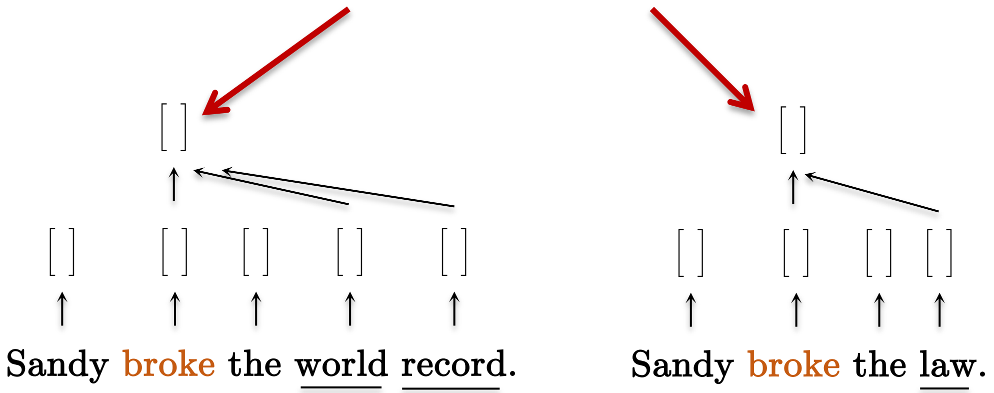
Understanding self-attention

- Illustration of self-attention :
  - Language model during training
  - Language model at inference



During training, the network learns to give attention to the words (the context) that make sense to predict the masked word.

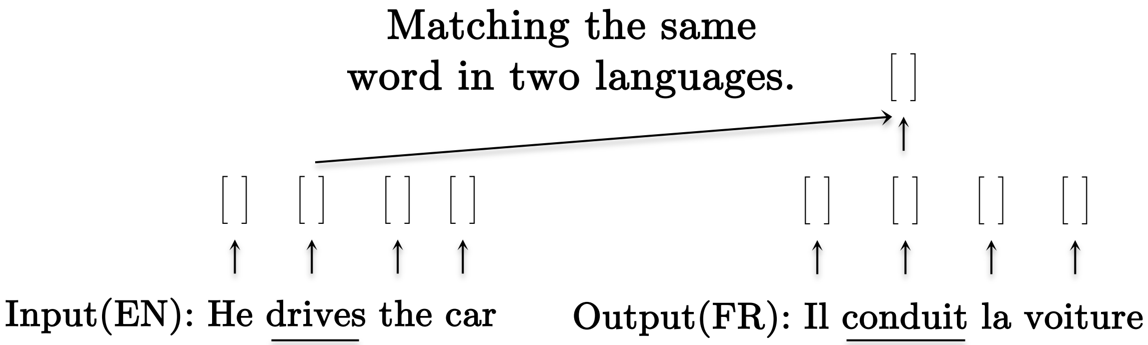
These two representations of the word "broke" are different.



At inference, the network computes the word representation depending on the context.

Understanding cross-attention

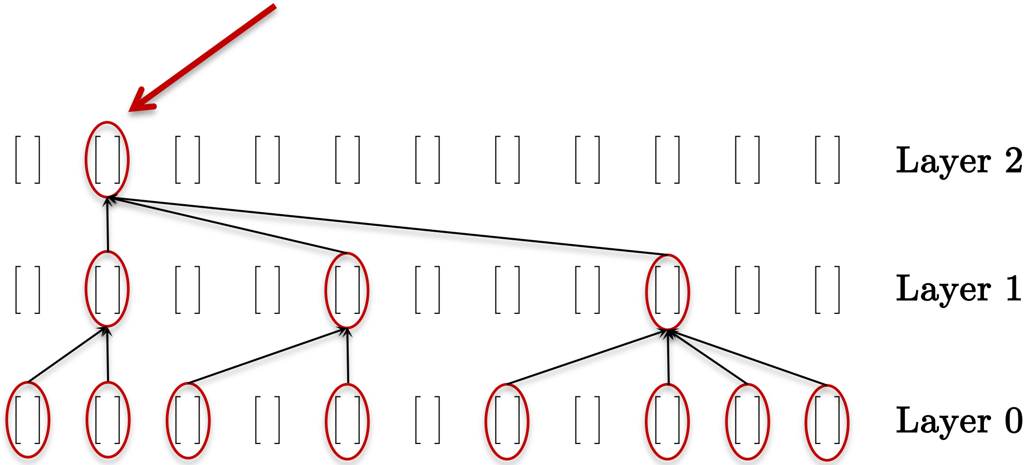
- Illustration of cross-attention :
    - Machine translation


Matching the same word in two languages.
Input(EN): He drives the car
Output(FR): Il conduit la voiture

Reception field

- Multiple layers increase the reception field.

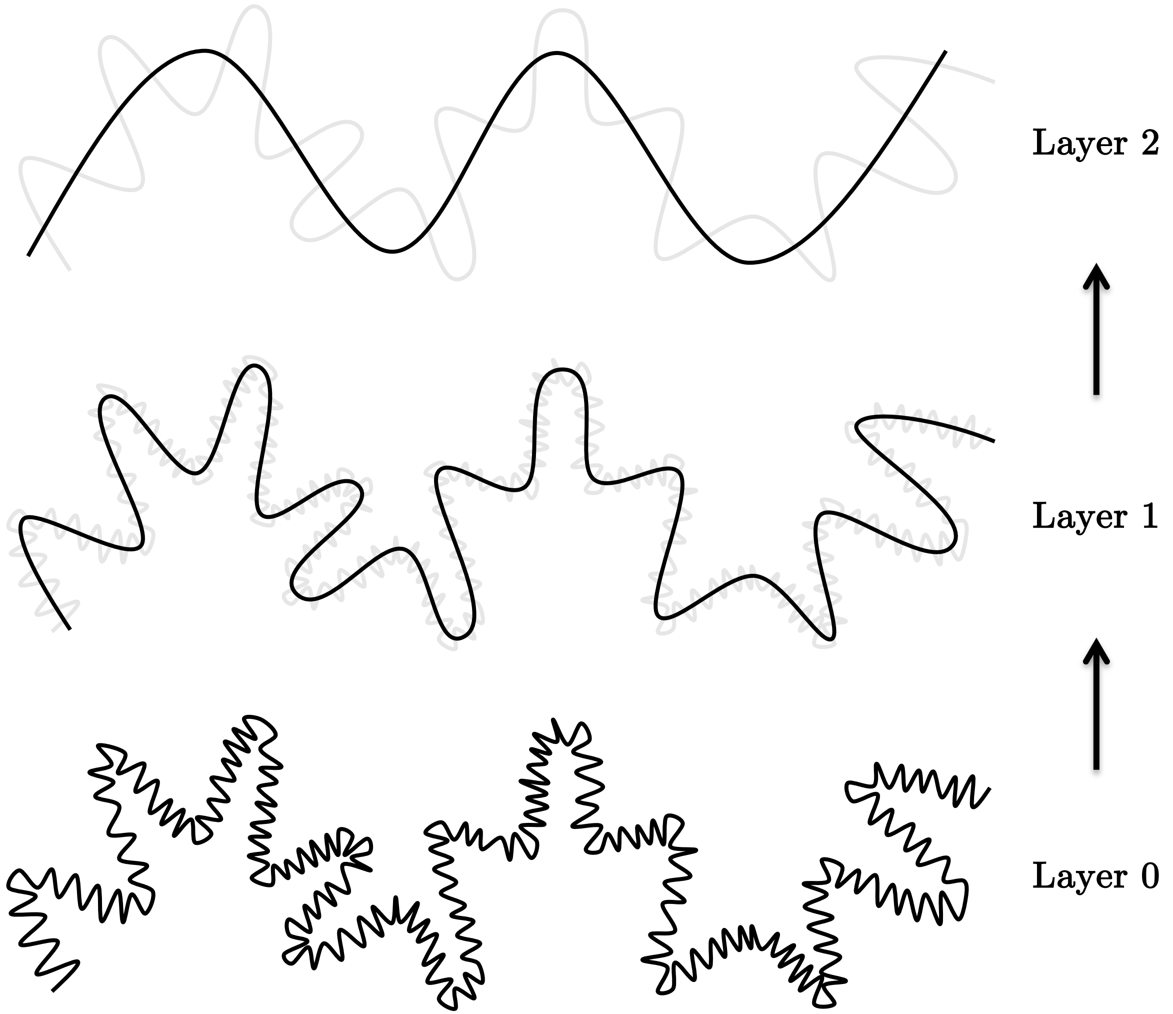
This word depends on the context
of 3 words in Layer 1 and the
context of 8 words in Layer 0.



The size of the original reception field/context of attention increases at each layer. For each word, the context size in the previous layer is variable (but small due to sparse softmax).

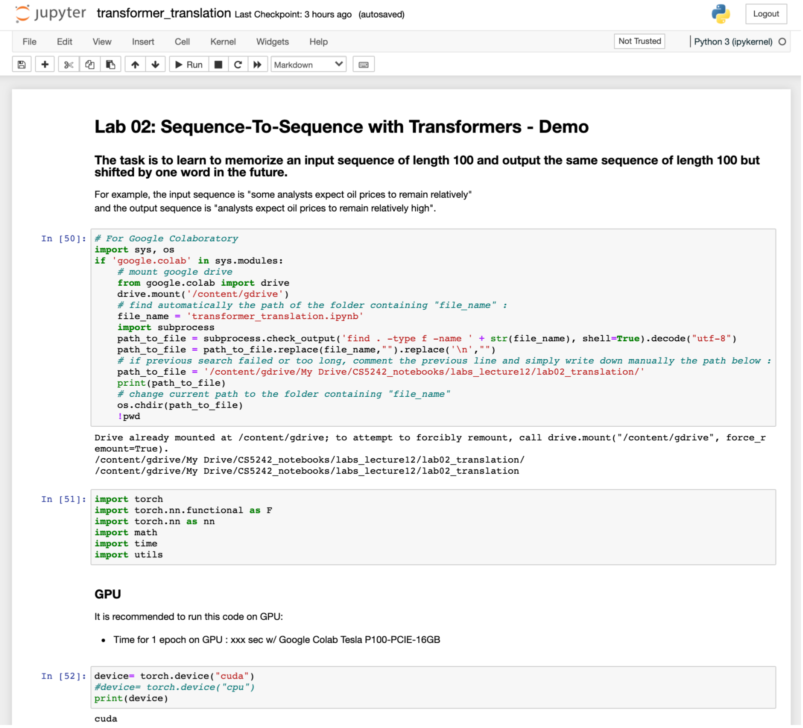
Hierarchical representation

- Multiple layers capture hierarchical representation.
- A simple illustration
  - Given the distribution of data (Layer 0).
  - Suppose that the attention context is defined by the closest data points.
  - At each layer, the self-attention mechanism smooths out the distribution with the context.
  - Successive layers provide a multi-scale/hierarchical representation of the data.



Lab 02

- PyTorch implementation of Seq2Seq Transformers



Lab 02: Sequence-To-Sequence with Transformers - Demo

The task is to learn to memorize an input sequence of length 100 and output the same sequence of length 100 but shifted by one word in the future.

For example, the input sequence is "some analysts expect oil prices to remain relatively"
and the output sequence is "analysts expect oil prices to remain relatively high".

```
In [50]: # For Google Colaboratory
         import sys, os
         if 'google.colab' in sys.modules:
             # mount google drive
             from google.colab import drive
             drive.mount('/content/gdrive')
             # find automatically the path of the folder containing "file_name" :
             file_name = 'transformer_translation.ipynb'
             import subprocess
             path_to_file = subprocess.check_output('find . -type f -name ' + str(file_name), shell=True).decode("utf-8")
             path_to_file = path_to_file.replace(file_name,"").replace('\n',"")
             # if previous search failed or too long, comment the previous line and simply write down manually the path below :
             path_to_file = '/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab02_translation/'
             print(path_to_file)
             # change current path to the folder containing "file_name"
             os.chdir(path_to_file)
             !pwd
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_r
emount=True).
/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab02_translation/
/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab02_translation
```

```
In [51]: import torch
         import torch.nn.functional as F
         import torch.nn as nn
         import math
         import time
         import utils
```

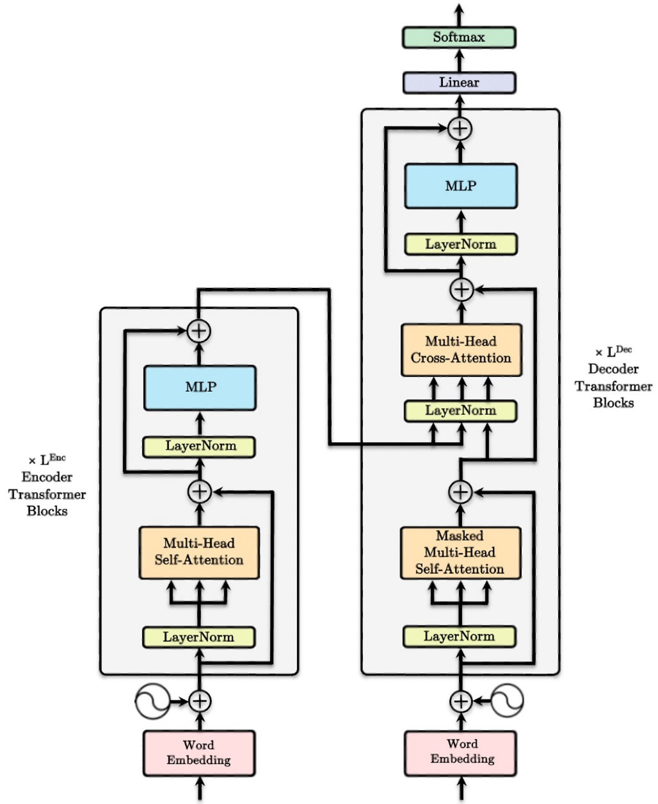
GPU

It is recommended to run this code on GPU:

- Time for 1 epoch on GPU : xxx sec w/ Google Colab Tesla P100-PCIE-16GB

```
In [52]: device= torch.device("cuda")
         #device= torch.device("cpu")
         print(device)
```

```
cuda
```


Softmax
Linear
MLP
LayerNorm
Multi-Head Cross-Attention
LayerNorm
Masked Multi-Head Self-Attention
LayerNorm
Word Embedding
× L^Dec Decoder Transformer Blocks
MLP
LayerNorm
Multi-Head Self-Attention
LayerNorm
Word Embedding
× L^Enc Encoder Transformer Blocks

Transformers in 2017

- Machine Translation
  - WMT-2014 dataset
  - BLEU score

LSTM + Attention

Google Brain's neural machine translation system in 2016

	EN-DE	EN-FR
GNMT	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer	28.4	41.8

CNN

Facebook Research's Convolutional sequence to sequence learning

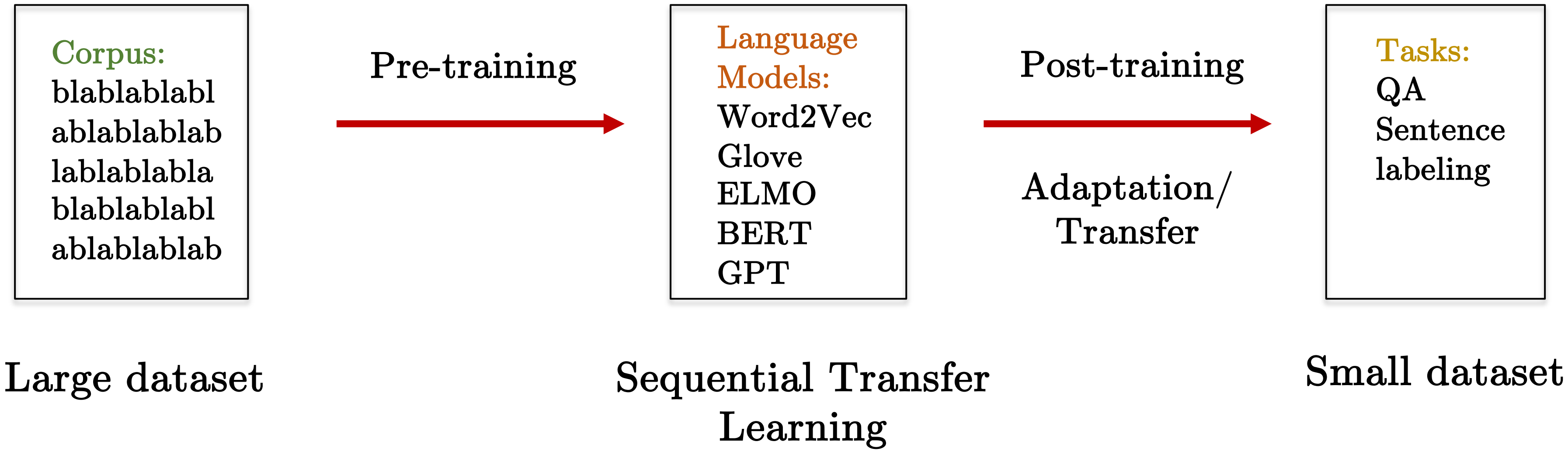
- Transformer is 3x faster to train than LSTM and CNN.
- Transformer has 24 layers vs LSTM w/ 3 layers and CNN w/ 40 layers.

Outline

- Language Models
- Memory Networks
- Transformers
- Language Model Transformers
- Sequence-To-Sequence Transformers
- Transfer Learning
- Conclusion

Transfer learning with language models

- A major theme in NLP since 2019 is sequential transfer learning.
  - Pre-trained language models on large-scale corpus (for capturing language prior) and
  - Post-trained to new tasks s.a. document classification, Q&A, named-entity recognition, etc by fine-tuning some layers on top of the pre-trained network.
  - Best performance for NLP tasks.



Corpus:
blablablabl
ablablablab
lablablabla
blablablabl
ablablablab

Pre-training

Language Models:
Word2Vec
Glove
ELMO
BERT
GPT

Post-training

Adaptation/
Transfer

Tasks:
QA
Sentence
labeling

Large dataset

Sequential Transfer Learning

Small dataset

Language modeling

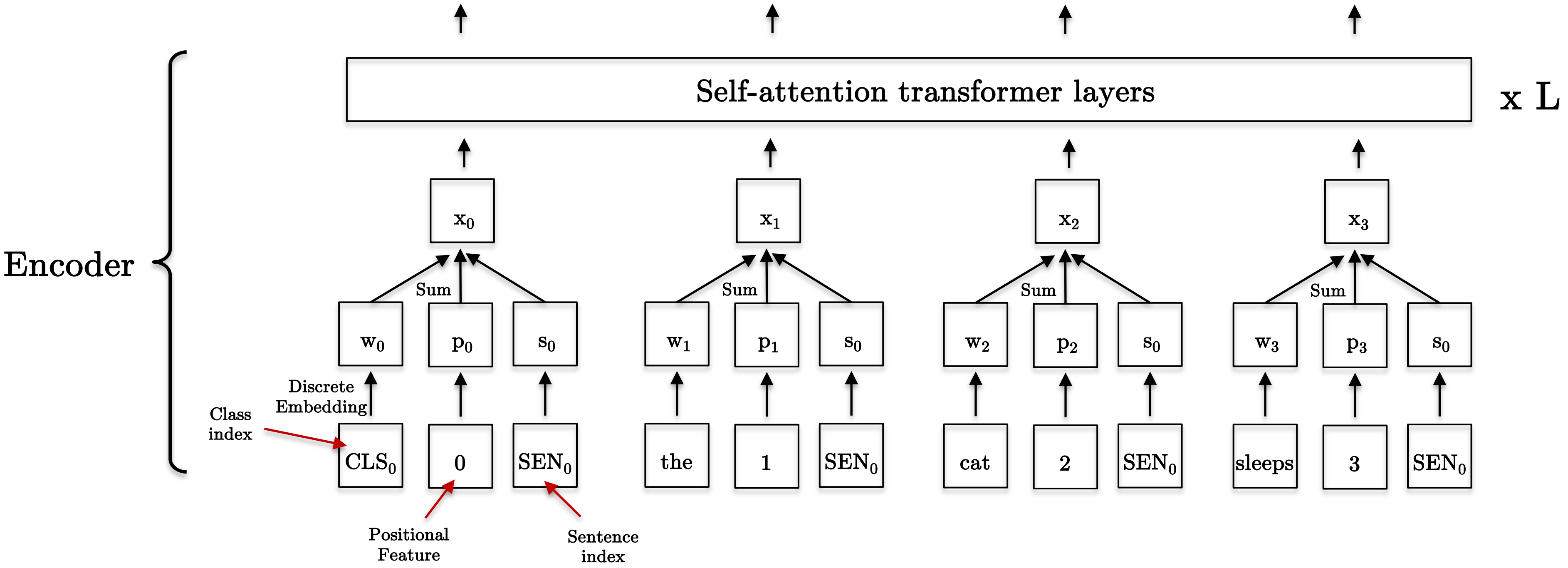
- Language modeling is a self-supervised task.
- It is unsupervised in the sense that there is no need for human labeled data.
- Additionally, large-scale unlabeled datasets are available for training.
- Train with billions or trillions of words from public datasets s.a. Wikipedia, Reddit, etc.

English Wikipedia alone has
3.9 billion words (2021)

Words Posted to Reddit:
72 billion (2015)

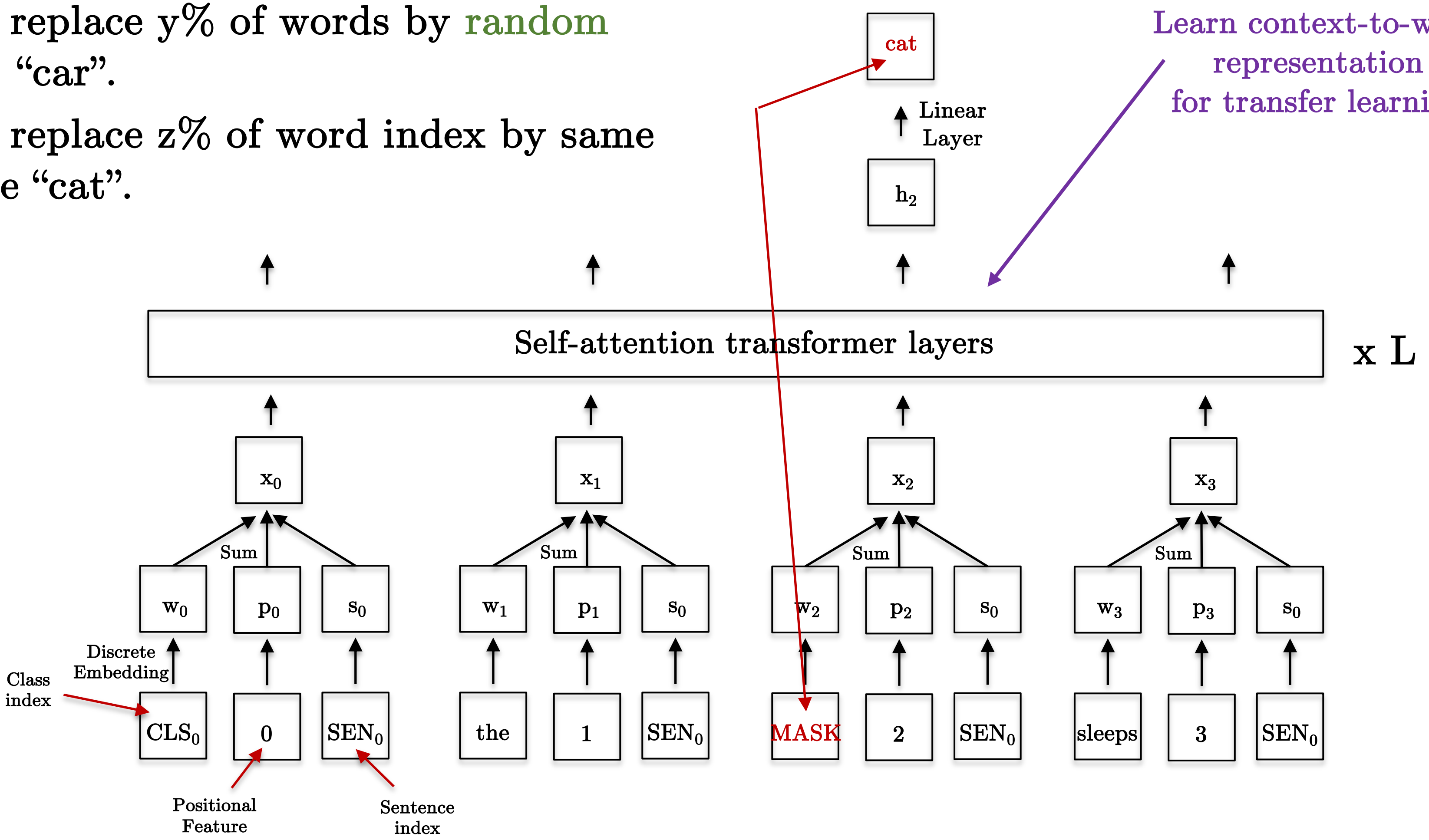
BERT

- Bi-directional Encoder Representation for Transformers (Devlin-et-al Google Brain 2019)
- Use positional encoding, class index and sentence index.
- Trained with two levels of hierarchical context :
  - Local dependencies with word prediction.
  - Global dependencies with sequence prediction.



Training with hidden words

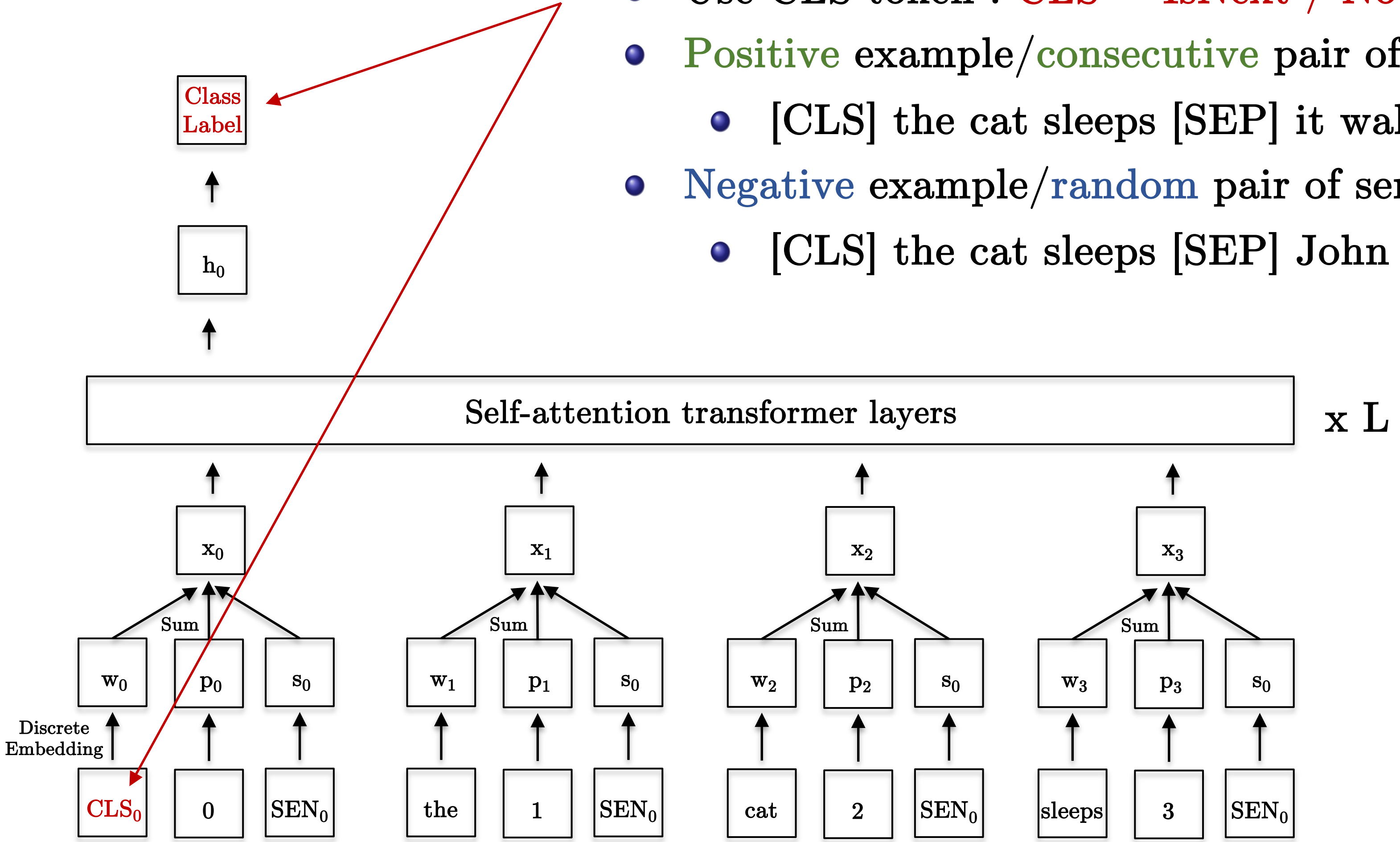
- Train by predicting words
  - Randomly replace x% of words by token MASK.
  - Randomly replace y% of words by random words, s.a. "car".
  - Randomly replace z% of word index by same words, here "cat".



Learn context-to-word representation for transfer learning

Training with sentence prediction

- Train by predicting the next sentence
  - Use CLS token : CLS = IsNext / NotNext
  - Positive example/consecutive pair of sentences :
    - [CLS] the cat sleeps [SEP] it wakes up
  - Negative example/random pair of sentences :
    - [CLS] the cat sleeps [SEP] John drives fast

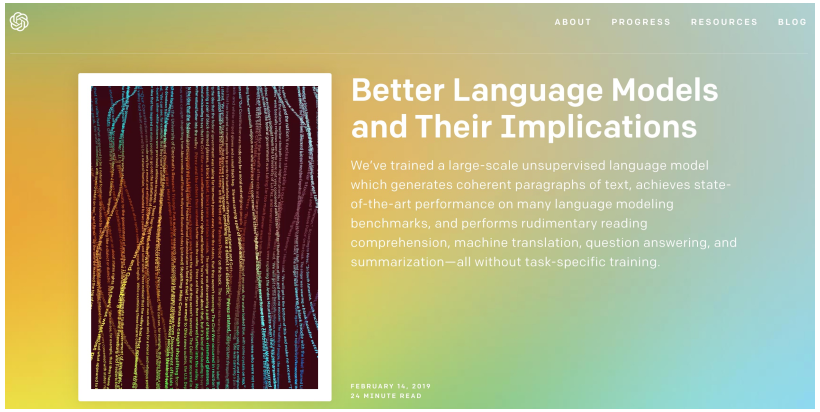


Training

- BERT base
  - 12 Transformers layers
  - 768 hidden features
  - 12 Attention heads
  - 110M parameters
- BERT large
  - 340M parameters
- Special tokenization of words with only 30K tokens.
- Dataset of 3B words
- Training took 256 TPU days (Oct 2018)
- Fine-tune on sentence classification, named-entity recognition (word classification), Q&A, etc.

GPT-2

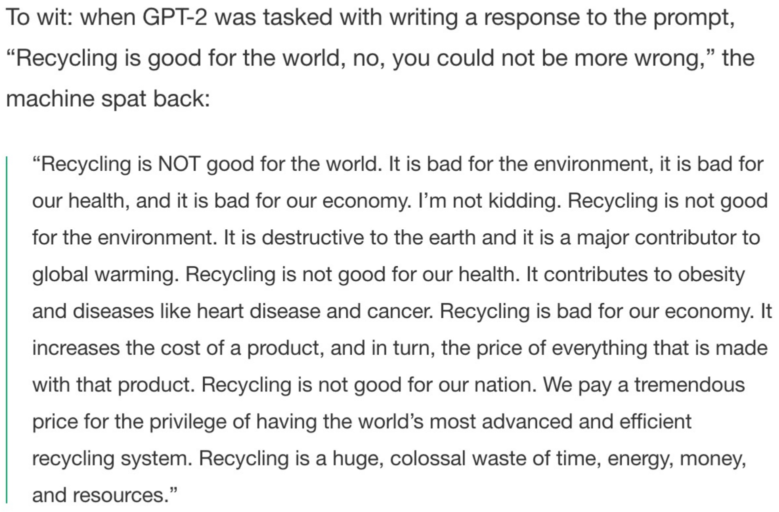
- Improving Language Understanding by Generative Pre-Training (GPT) (Radford-et-al OpenAI 2018)
  - Pre-trained on 8M webpages, WebText 40GB.
  - SOTA on 7 NLP tasks without fine-tuning, simply by zero-shot learning !
  - 1.5B parameters
  - 2048 GPUs



Better Language Models and Their Implications

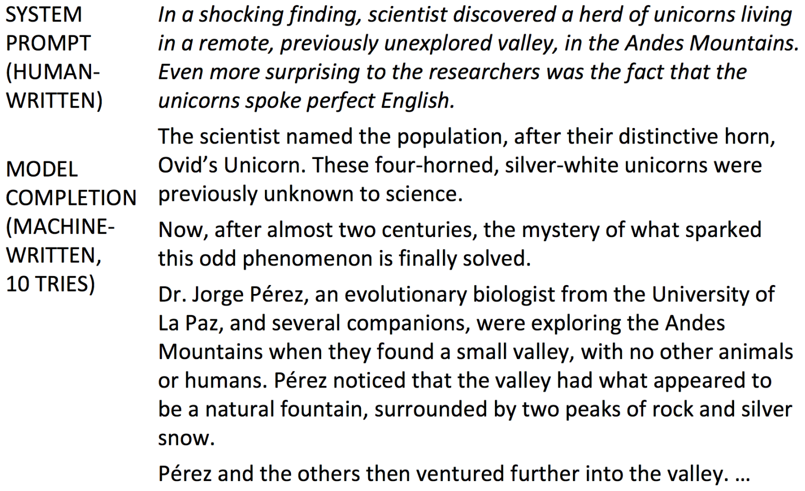
We've trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state-of-the-art performance on many language modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization—all without task-specific training.

FEBRUARY 14, 2019
24 MINUTE READ



To wit: when GPT-2 was tasked with writing a response to the prompt, “Recycling is good for the world, no, you could not be more wrong,” the machine spat back:

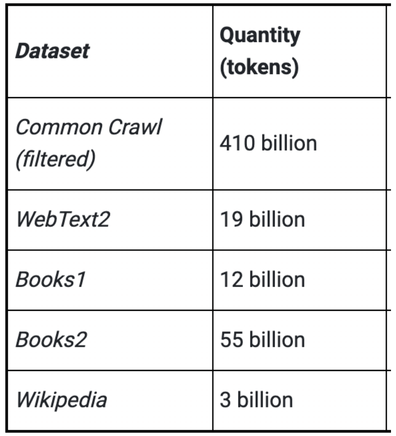
> “Recycling is NOT good for the world. It is bad for the environment, it is bad for our health, and it is bad for our economy. I’m not kidding. Recycling is not good for the environment. It is destructive to the earth and it is a major contributor to global warming. Recycling is not good for our health. It contributes to obesity and diseases like heart disease and cancer. Recycling is bad for our economy. It increases the cost of a product, and in turn, the price of everything that is made with that product. Recycling is not good for our nation. We pay a tremendous price for the privilege of having the world’s most advanced and efficient recycling system. Recycling is a huge, colossal waste of time, energy, money, and resources.”



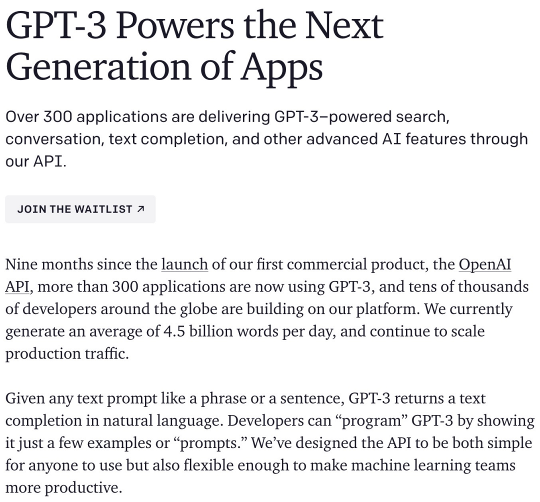
SYSTEM PROMPT (HUMAN-WRITTEN)	*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)	The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow. Pérez and the others then ventured further into the valley. …

GPT-3

- Introduced in Brown-et-al OpenAI 2020
- Pre-trained on multiple datasets.
- 175B parameters
- “The supercomputer developed for OpenAI is a single system with more than 285,000 CPU cores, 10,000 GPUs and 400 gigabits per second of network connectivity for each GPU server”
- US$12 Million to train



Dataset	Quantity (tokens)
Common Crawl (filtered)	410 billion
WebText2	19 billion
Books1	12 billion
Books2	55 billion
Wikipedia	3 billion



GPT-3 Powers the Next Generation of Apps

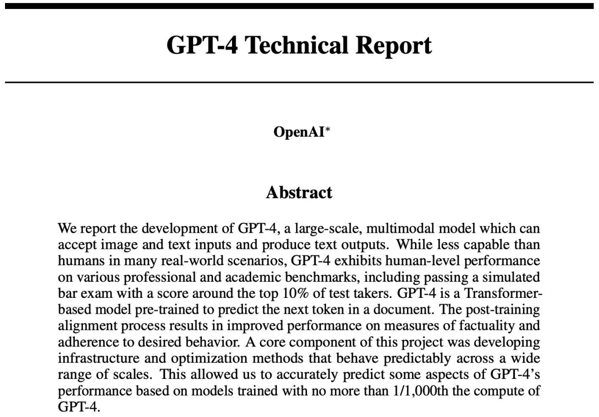
Over 300 applications are delivering GPT-3–powered search, conversation, text completion, and other advanced AI features through our API.

Nine months since the launch of our first commercial product, the OpenAI API, more than 300 applications are now using GPT-3, and tens of thousands of developers around the globe are building on our platform. We currently generate an average of 4.5 billion words per day, and continue to scale production traffic.

Given any text prompt like a phrase or a sentence, GPT-3 returns a text completion in natural language. Developers can “program” GPT-3 by showing it just a few examples or “prompts.” We’ve designed the API to be both simple for anyone to use but also flexible enough to make machine learning teams more productive.

GPT-4

- Introduced in Achiam-et-al OpenI 2023
- Pre-trained on dataset ?
- ? parameters
- Training time ?

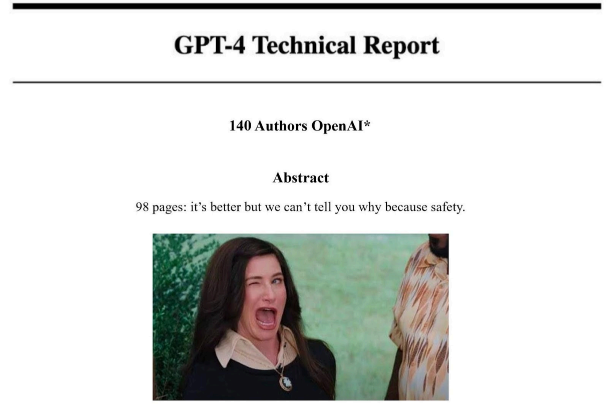


GPT-4 Technical Report

OpenAI*

Abstract

We report the development of GPT-4, a large-scale, multimodal model which can accept image and text inputs and produce text outputs. While less capable than humans in many real-world scenarios, GPT-4 exhibits human-level performance on various professional and academic benchmarks, including passing a simulated bar exam with a score around the top 10% of test takers. GPT-4 is a Transformer-based model pre-trained to predict the next token in a document. The post-training alignment process results in improved performance on measures of factuality and adherence to desired behavior. A core component of this project was developing infrastructure and optimization methods that behave predictably across a wide range of scales. This allowed us to accurately predict some aspects of GPT-4's performance based on models trained with no more than 1/1,000th the compute of GPT-4.



GPT-4 Technical Report

140 Authors OpenAI*

Abstract

98 pages: it's better but we can't tell you why because safety.

Outline

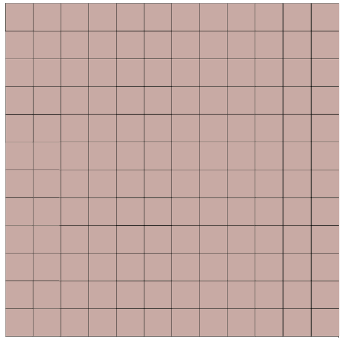
- Language Models
- Memory Networks
- Transformers
- Language Model Transformers
- Sequence-To-Sequence Transformers
- Transfer Learning
- Conclusion

Conclusion

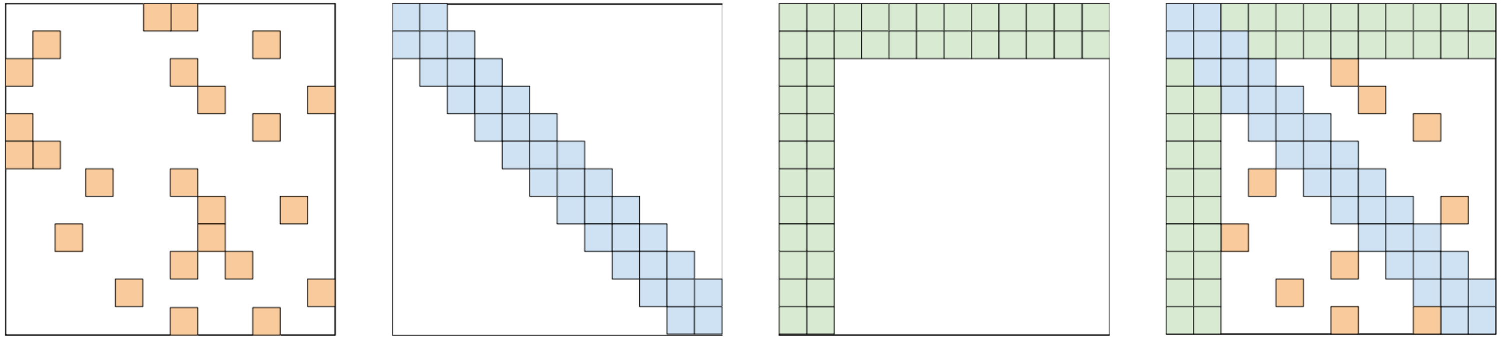
- Human attention mechanism allows to focus biological resources on a small set of important things (visual, sound, cognitive signals) to make decisions.
- ANNs are a generic/universal architecture to process any unstructured datasets, a.k.a. sets.
- Attention is “eating” deep learning.
- Transformers for Computer Vision with Visual Transformers (Dosovitskiy-et-al Google Brain 2021).
- Transformers for Graphs with Graph Transformers.
- Issue with long sequences because complexity is (L^2d).

Reducing complexity

- Long sequence issue with $O(L^2d)$, n being sequence length and d hidden dimension.
  - Sparse transformers s.a. BigBird (Zaheer-et-al Google Brain 2021).



Original
Transformers

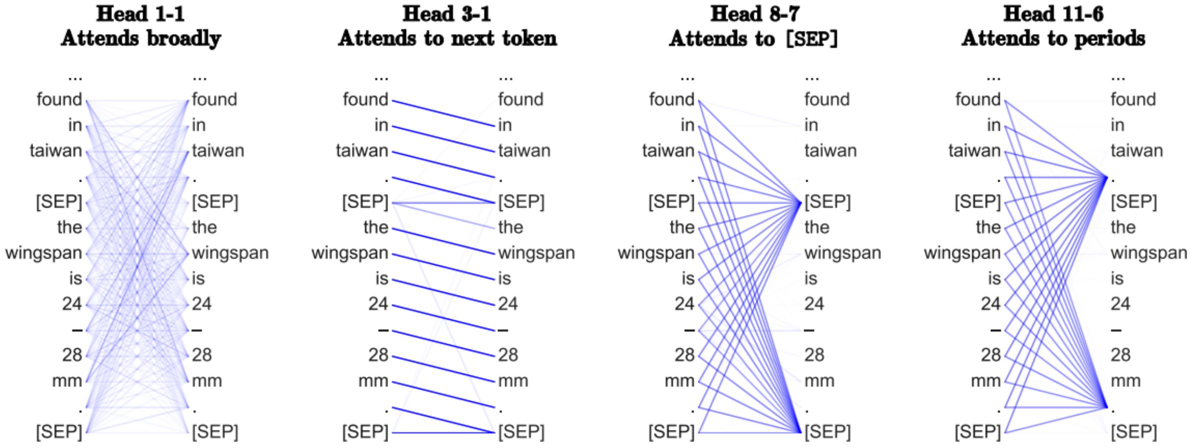


(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Structured
Transformers

Interpretability

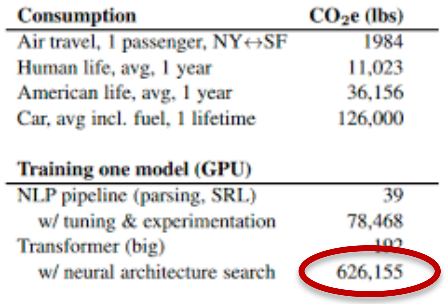
- What does BERT look at? (Clark-et-al, 2019) An Analysis of Transformer's attention heads.



$$\mathrm{Softmax}_{\mathrm{row}}\left(\frac{QK^T}{\sqrt{d}}\right) \in \mathbb{R}^{L\times L}$$

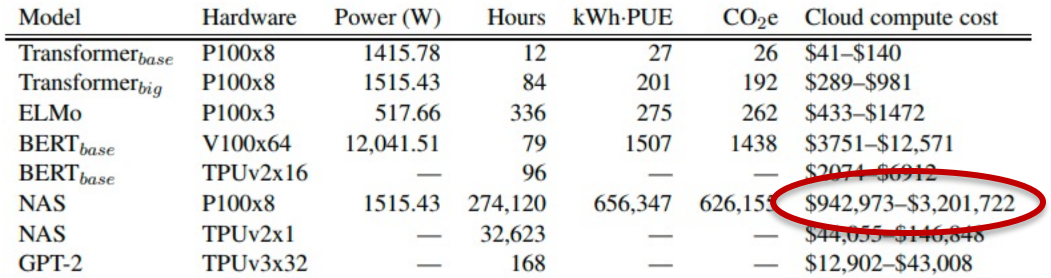
Open-source

- Training large-scale models is costly in terms of time, money, CO_2 :



Consumption	CO_2e (lbs)
Air travel, 1 passenger, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experimentation	78,468
Transformer (big)	192
w/ neural architecture search	626,155

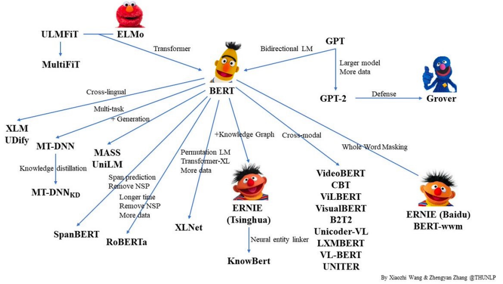
Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]



Model	Hardware	Power (W)	Hours	kWh·PUE	CO_2e	Cloud compute cost
$\text{Transformer}_{base}$	P100x8	1415.78	12	27	26	\$41–\$140
Transformer_{big}	P100x8	1515.43	84	201	192	\$289–\$981
ELMo	P100x3	517.66	336	275	262	\$433–\$1472
BERT_{base}	V100x64	12,041.51	79	1507	1438	\$3751–\$12,571
BERT_{base}	TPUv2x16	—	96	—	—	\$2074–\$6912
NAS	P100x8	1515.43	274,120	656,347	626,155	\$942,973–\$3,201,722
NAS	TPUv2x1	—	32,623	—	—	\$44,055–\$146,848
GPT-2	TPUv3x32	—	168	—	—	\$12,902–\$43,008

Table 3: Estimated cost of training a model in terms of CO_2 emissions (lbs) and cloud compute cost (USD).[7] Power and carbon footprint are omitted for TPUs due to lack of public information on power draw for this hardware.

- Available pre-trained/post-trained models :
  - PyTorch (Llama), TensorFlow (Gemma)
  - Hugging Face, https://huggingface.co



By Xiaozhi Wang & Zhengyan Zhang @THUNLP

Questions?