FUTURE AI RESEARCH

ICSC
Centro Nazionale di Ricerca in HPC,
Big Data and Quantum Computing

QUICK INTRODUCTION TO - TRANSFORMERS -

S. Giagu - 5th ML_INFN Hackathon: Advanced Level

INFN Pisa - 14.11.2023

SAPIENZA
UNIVERSITÀ DI ROMA

INFN
Istituto Nazionale di Fisica Nucleare

ATTENTION AND TRANSFORMERS

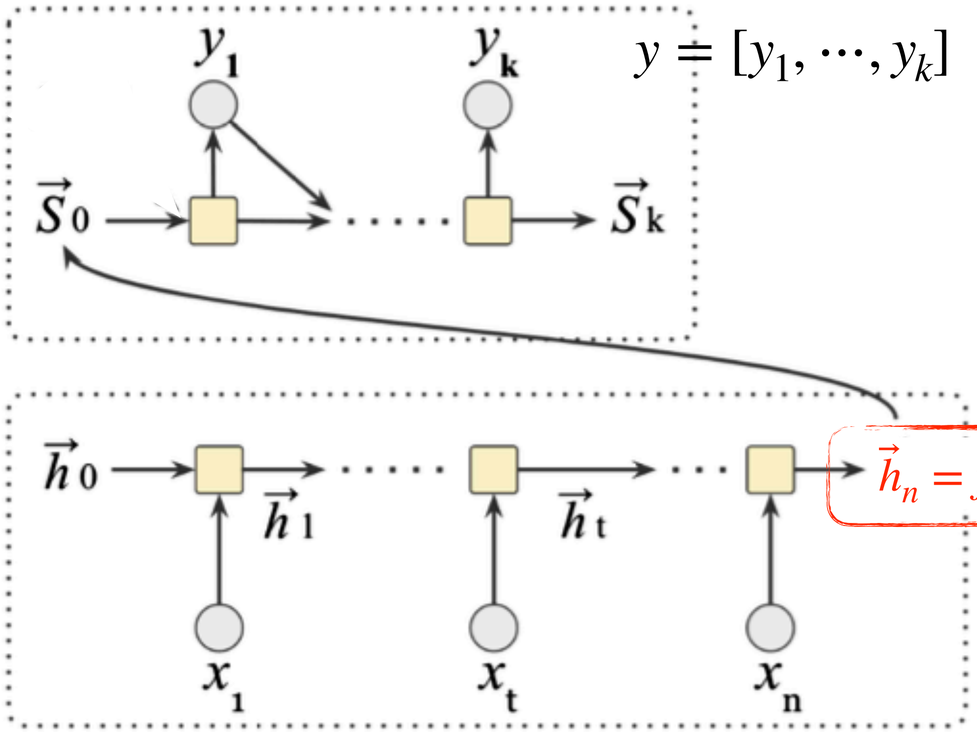
- The attention mechanism (Bhdanau et al, arXiv:1409.0473, 2014) has been proposed to help memorize long source sentences in neural machine translation tasks
  - ideally a context vector used to store the learned internal representation of the input should contains information from the entire input, but this scales poorly with the input size: either the size of the context vector grows making the problem computationally hard, or one must accept a degradation in performance
  - attention idea: replace the static elements of the context vector with shortcuts between the context vector and the entire source input, with shortcuts weights that dynamically adapt for each output element
  - intuition: even if the model may need to draw upon information from the entire input, however some parts of it will be more relevant than others for the specific task. The attention mechanism provides a way to identify such parts …

- Transformers (Vaswani et al, arXiv:1706.03762, 2017) are recent DL architectures based on the attention mechanism that have gradually replaced RNNs in mainstream NLP tasks, and compete/surpass (when (pre)trained with large datasets) other neural architectures in vision or in time-domain related tasks
  - facilitate the learning of long range sequences
  - don't need recurrence → no gradient vanishing or explosion problems
  - needs fewer training steps and can be easily parallelised on GPUs → computationally efficient

EXAMPLE: CONTEXT VECTOR LENGHT ISSUE IN SEQ2SEQ

seq2seq models (used in machine translation tasks): transforms an input sequence to a new one (both of arbitrary lengths)

decoder



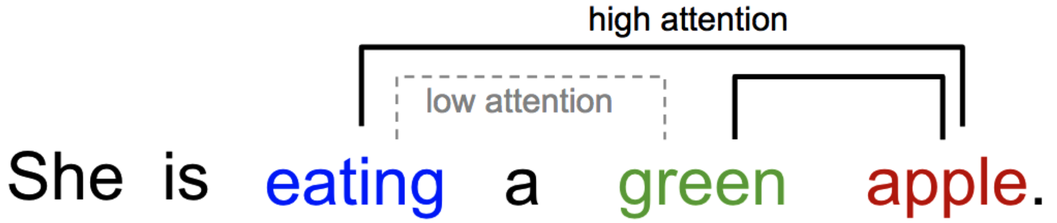
encoder $x = [x_1, \cdots, x_n]$

- encoder-decoder architecture:
  - encoder: processes the input sequence and compresses the information into a context vector of a fixed length: $\vec{h}_N$ (summary of the meaning of the whole source sequence)
  - decoder: initialised with the context vector to emit the transformed output $p(y_i \mid \{y_1, \cdots, y_{i-1}\}) = g_w(y_{i-1}, S_{i-1})$

$\vec{h}_n = f_w(h_{n-1}, x_n) = [v_0, v_1, \cdots, v_d]$

becomes ineffective for long sequence, unless implemented in complex StackedRNN architectures that are hard/impossible to train in an acceptable time

ATTENTION



- intuitive idea:
  - attention is, to some extent, motivated by how we correlate words in one sentence or pay visual attention to different regions of an image
  - when we see “eating”, we expect to encounter a food word very soon. The color term describes the food, but probably not so much with “eating” directly
- the attention mechanism forms a representation of the entire input, but different parts of it are weighted differently according to the task at hand. By making the weights a learnable component, the network can learn to attend only to the relevant parts of the input

context vector for the output i: $C_i = \sum_j \alpha_{ij} h_j$

a suitable alignment function:
ex. inner product: $\text{alignment}(s_i, h_j) = s_i^T h_j$

$$\alpha_{ij} = \text{softmax}(\text{alignment}(y_i, x_j)) = \frac{\exp[\text{alignment}(s_{i-1}, h_j)]}{\sum_k \exp[\text{alignment}(s_{i-1}, h_k)]}$$

alignment weight between output y_i and the input x_j

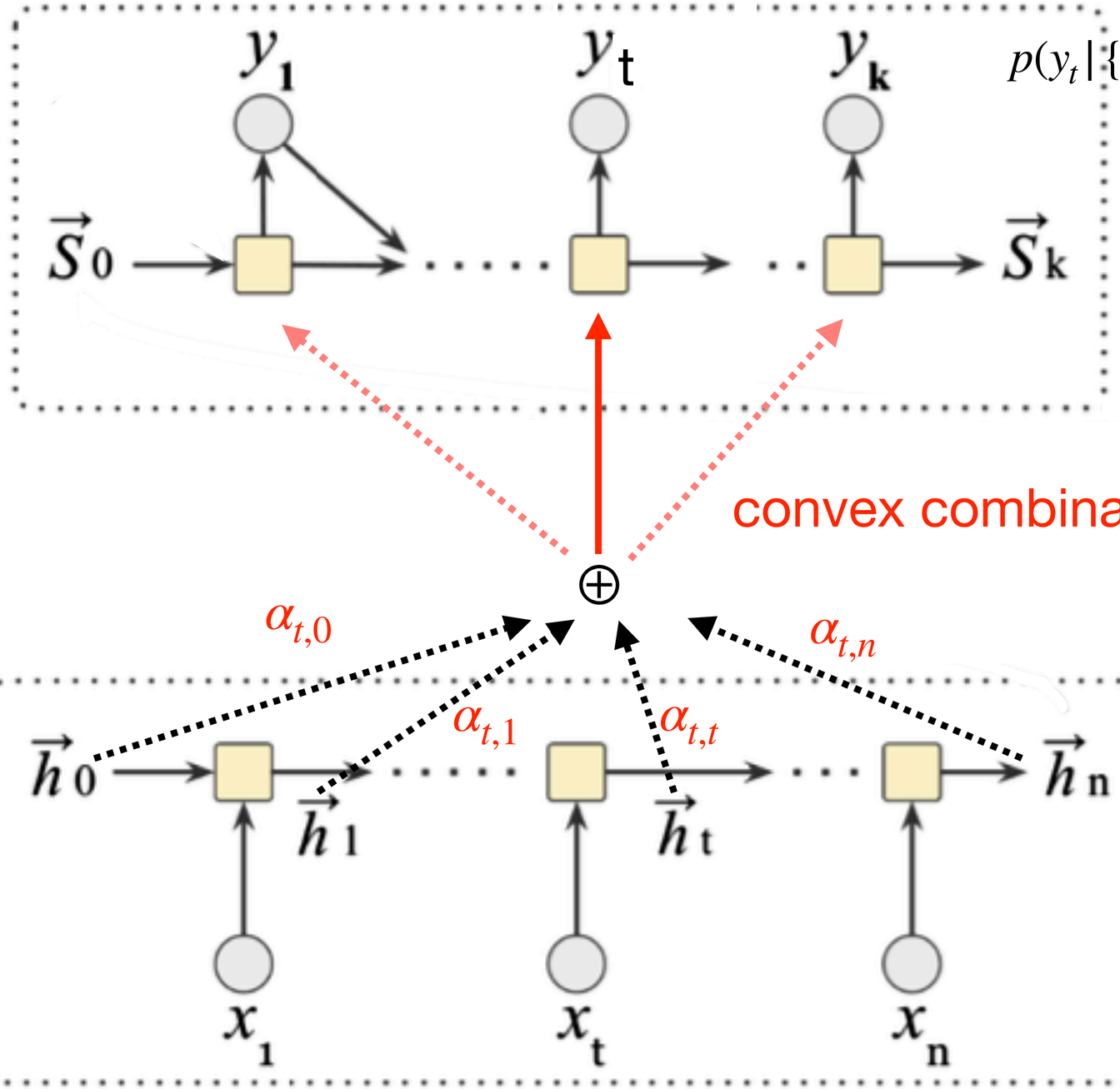
guarantees a convex combination $\alpha_{ij} \geq 0; \sum_j \alpha_{ij} = 1$

output

hidden states

$p(y_t|\{y_1, \cdots, y_{t-1}\}) = g_w(y_{t-1}, s_{t-1}, C_t)$

decoder



convex combination $C_t = \sum_j \alpha_{tj} h_j$

hidden states

encoder

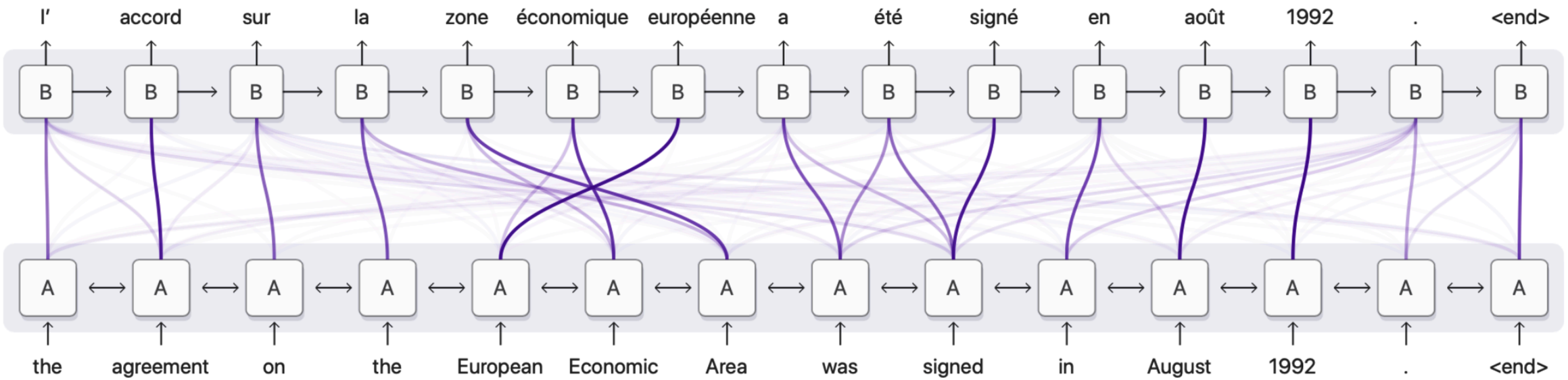
input

RNNSearch: BIDIRECTIONAL RNN WITH ATTENTION

- attention idea implemented for the first time in a model (RNNSearch: D. Bahdanau, K. Cho and Y. Bengio, ICLR 2015) which made a breakthrough in machine translation by combining a bi-directional RNN with an additive attention mechanism

$$\text{alignment}(y_i, x_j) = U \tanh(W s_{i-1} + \tilde{W} h_j + b_i)$$

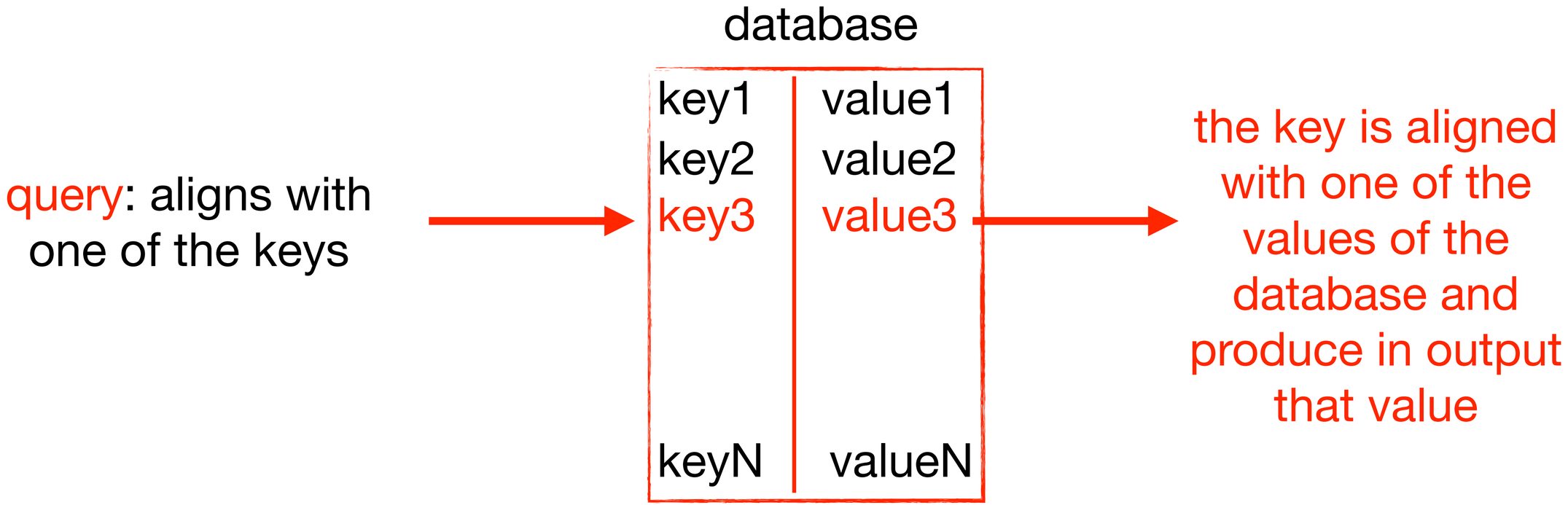
$U, W, \tilde{W}, b_i$
learnable weights



attention weights in a seq-to-seq problem of translation from ENG to FR

ATTENTION MECHANISM AS A DB RETRIEVAL TECHNIQUE

- the attention mechanism can be also described in a different way, as a technique that mimics the retrieval in a database of a value **v** based on a query **q** and on a key **k**
- in a database retrieval process the query is used to identify a key that allows to retrieve a given value associated to that key:



- the dotted attention mechanism mimics this via a neural network architecture:

$$\text{attention}(q, \mathbf{k}, \mathbf{v}) = \sum_i \text{similarity}(q \cdot k_i) \times v_i$$

a way to measure how similar ("aligned') are q and k_i

the value associated to the key k_i

- in a traditional db the query returns one value, and this corresponds to use a similarity function that produce a one-hot encoding [0,0,0,…,**1**,0,…,0] that effectively return just one value v_k
- the dotted attention generalise that by using a distribution, e.g. weights ∈ [0,1] that sum up to 1

different possibilities for the similarity measure

$$\lambda_i = f(q, k_i) = \begin{cases} W_q q + W_k k_i & \text{additive (as in the RNNSearch)} \\ q^T k_i & \text{dot-product attention} \\ \dfrac{q^T k_i}{\sqrt{d_k}} & \text{scaled dot-product attention} \\ (W_q q^T W_k k_i)/\sqrt{d_k} & \text{general att.} \end{cases}$$

similarity measures

much more efficient than additive similarity

project q and k on new spaces (to be in the same similarity-space with the key) via a learnable transformation

$$\Rightarrow \alpha_i = \frac{e^{\lambda_i}}{\sum_j e^{\lambda_j}} \Rightarrow \text{Attention scores}(v) = \sum_i \alpha_i v_i$$

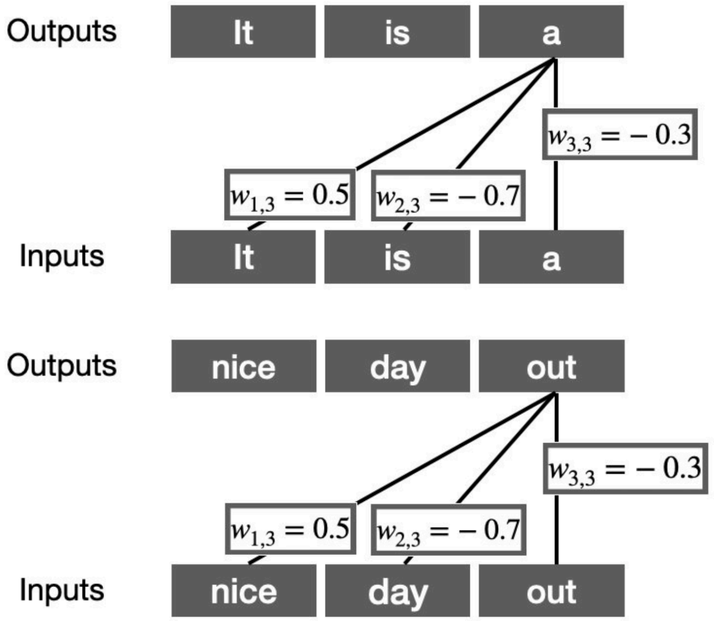
- **Cross Attention:** allows to compare each output with a context vector that takes into account all the input elements
  - query i: hidden representation vector for the i-th output element: **s_i**
  - key j: hidden representation vector for the j-th input element: **h_j**
  - value j: again the hidden representation vector for the j-th input element: **h_j**

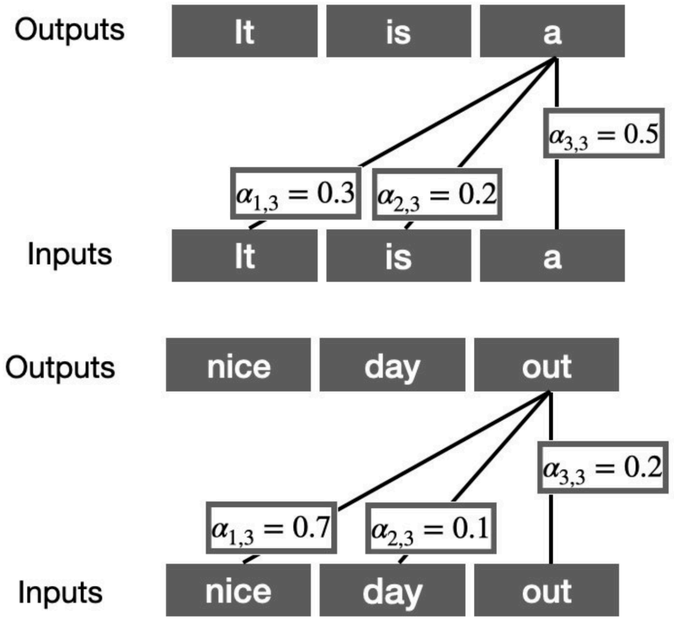
- **Self Attention:** relates different positions of a single sequence in order to compute a representation of the same sequence)
  - query i: hidden representation vector for the i-th input element: **h_i**
  - key j: hidden representation vector for the j-th input element: **h_j**
  - value j: again the hidden representation vector for the j-th input element: **h_j**

ATTENTION SCORES VS WEIGHTS IN FULLY CONNECTED LAYERS

- attention mechanisms allows to attend to different parts of a sequence, this sounds very similar to a fully connected layer in a MLP, but with an important advantage:
  - in a fully-connected layers weight are fixed after training, so are static wrt the input
  - in the attention instead, the weights are dynamic and input dependent (the computation involves the comparison of each input element to each other)
  - this allow a neural network to selectively weight the importance of different input features

Dense layer weights

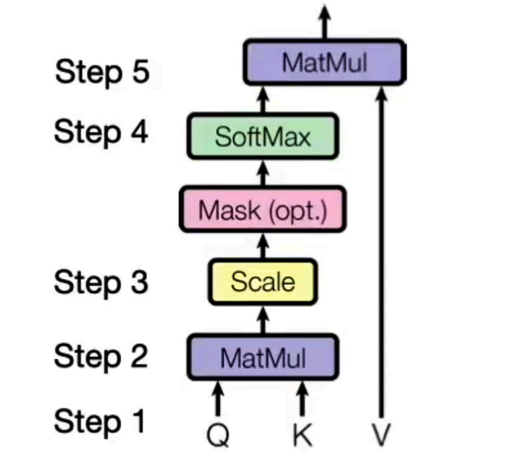




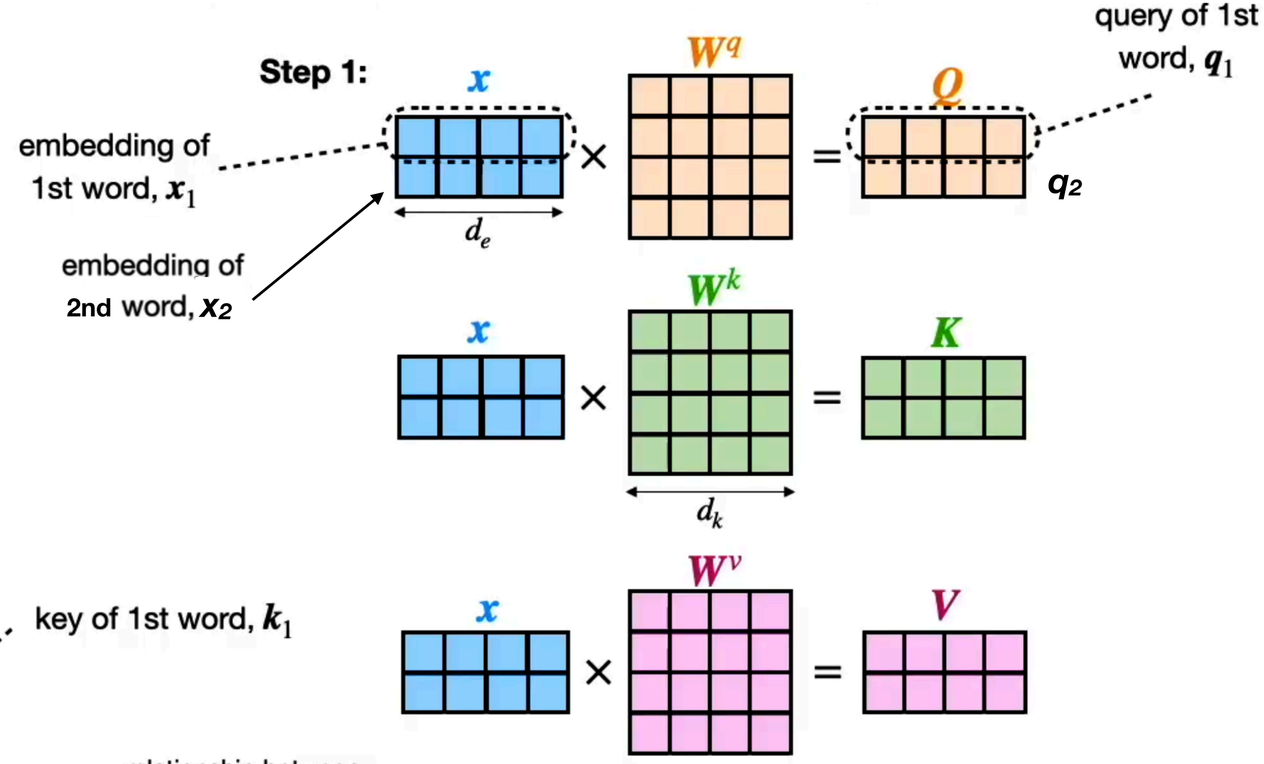
Attention scores

credit: S. Raschka

SCALED DOT-PRODUCT ATTENTION VISUALIZED



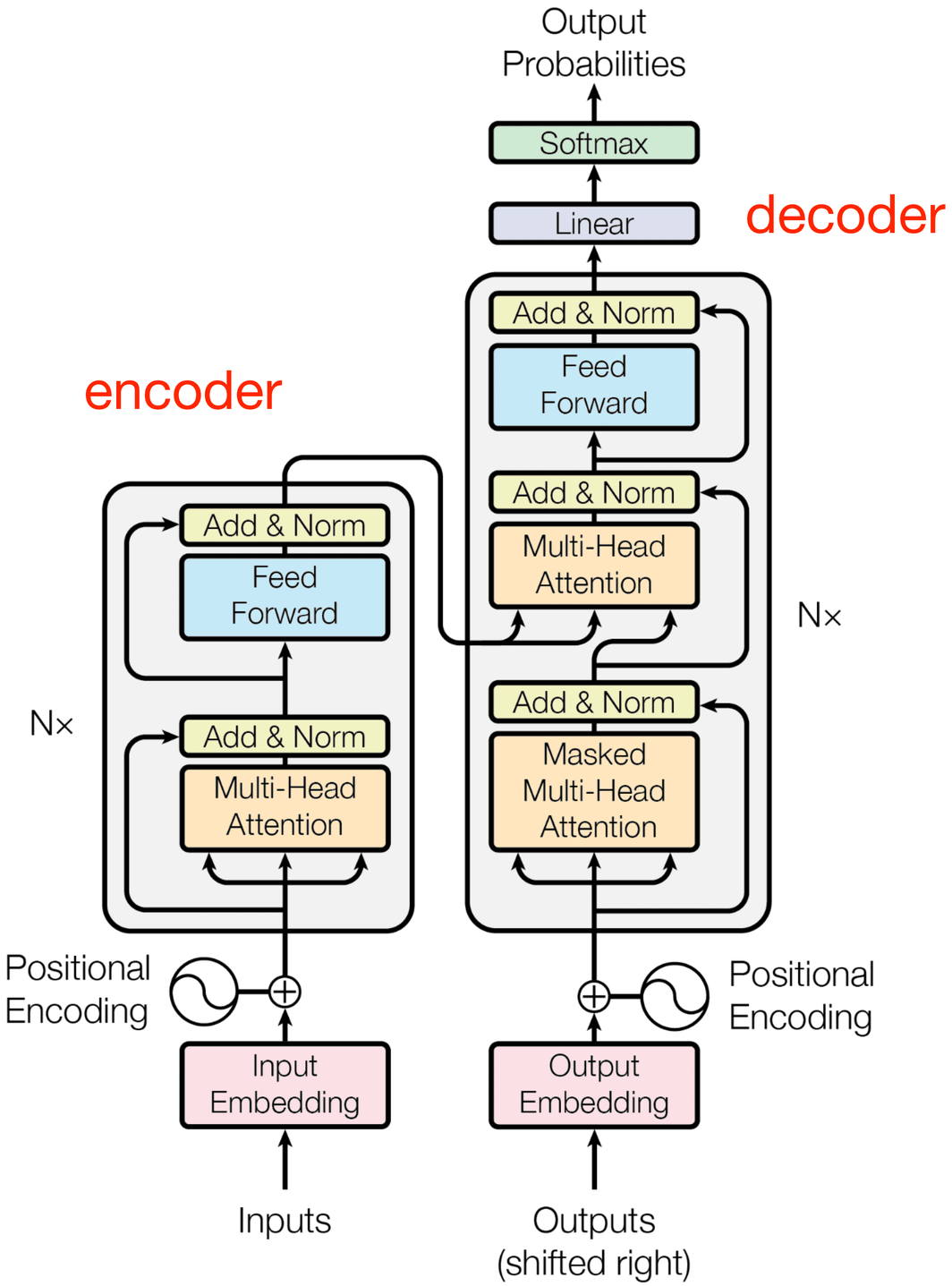
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.



credit: S. Raschka

TRANSFORMER ARCHITECTURE

- A. Vaswani et al. “Attention is All You Need” (2017) arXiv:1706.03762
- Encoder-decoder architecture for sequence analysis fully based on attention w/o recurrence
- today has substantially replaced any other DNN model in NLP tasks



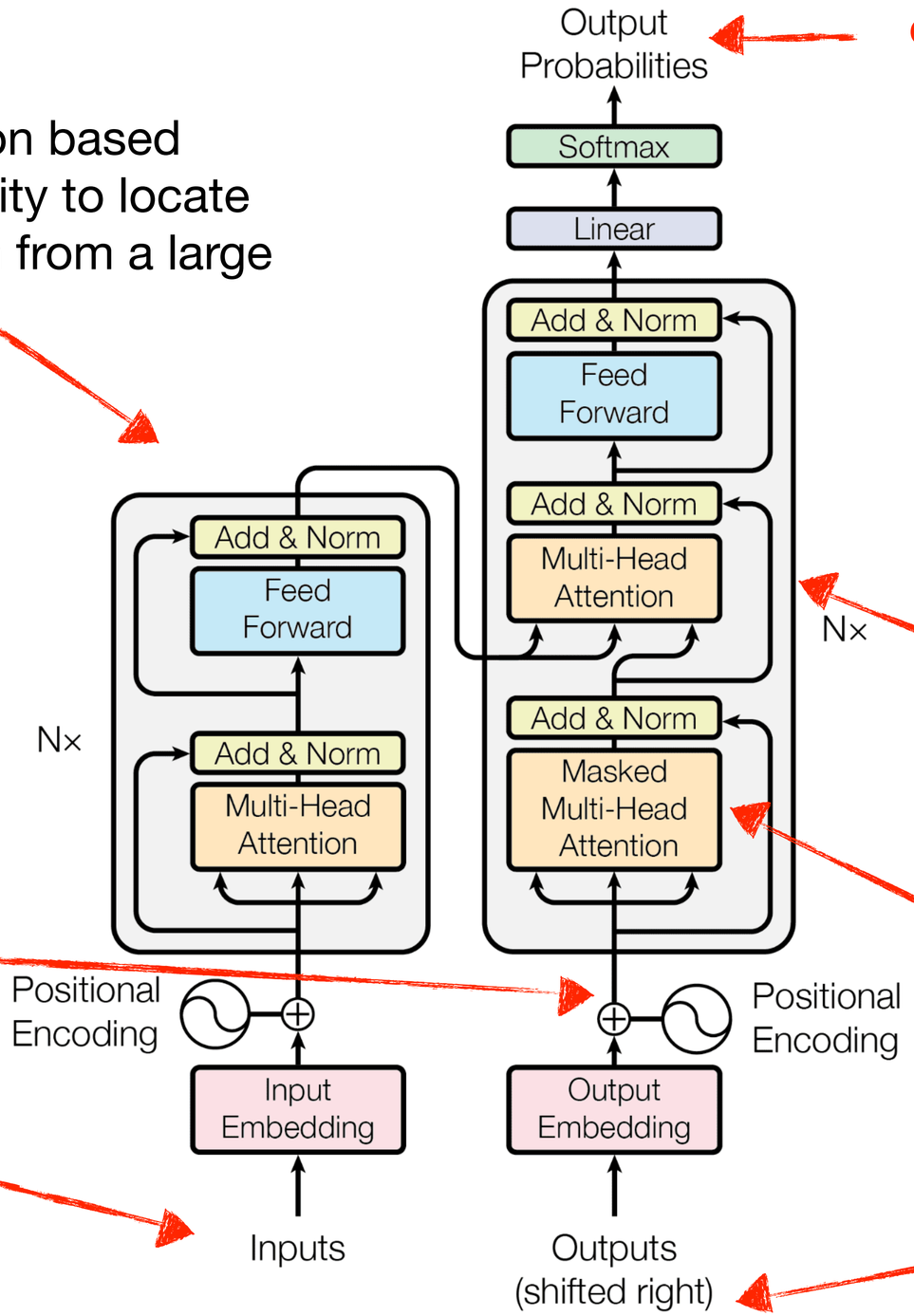
encoder:

generates a self-attention based representation with capability to locate specific piece of information from a large context

stack of N modules, each one made by a **multi-head self-attention** layer and a **point-wise dense feed-forward net**

positional encoding: allows the sentence not to be treated as a bag of words

input is the entire sequence of words (not one by one like in a RNN)



output: probability over the words/ tokens dictionary (the higher prob. word/token is chosen

decoder:

looks at the correlations from the output words and between them and the encoded input to produce the translated text

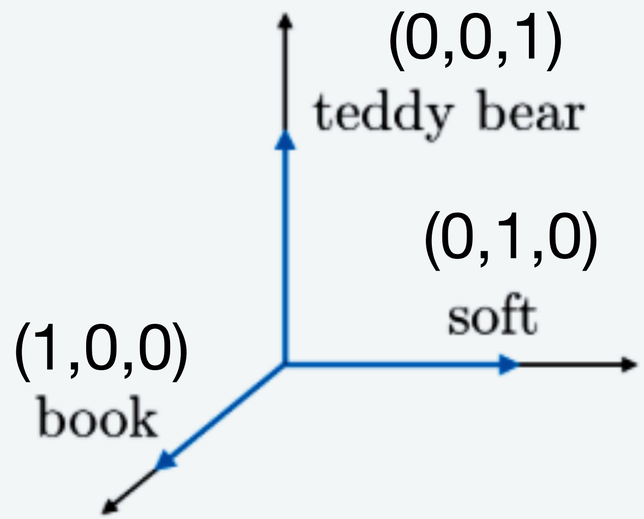
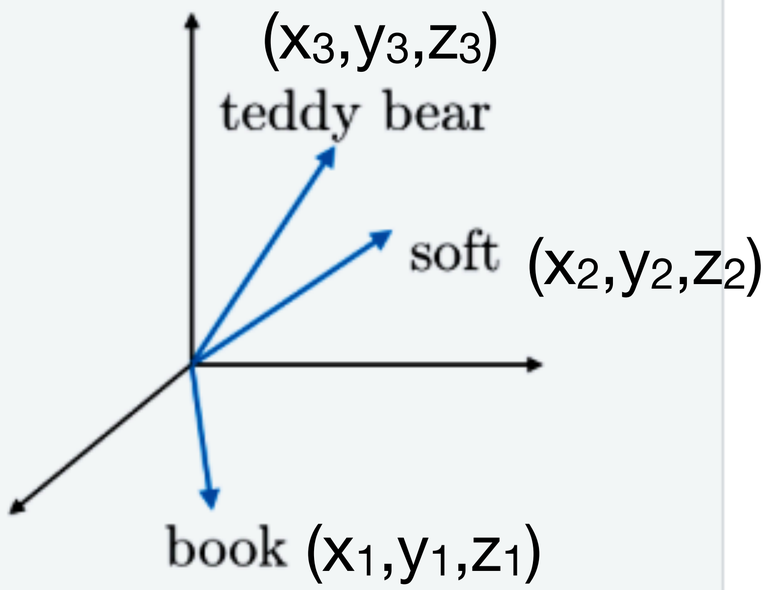
x-**attention** layer that combines output words embedding with input words embeddings

masked self attention layer that combines output words with **previous** output words (teacher forcing)

WORD EMBEDDING (I.E. LEARN REPRESENTATIONS OF WORDS)

to be understood by a NN a text must be vectorized + represented effectively
two main techniques typically used:

example:
a 3 words
dictionary

1-hot representation	Word embedding
	
• simple • naive approach, no similarity information	• more complex, more powerful • takes into account words similarity • dense vectors can be learned with a NN

doesn't scale well with the dictionary dimension …

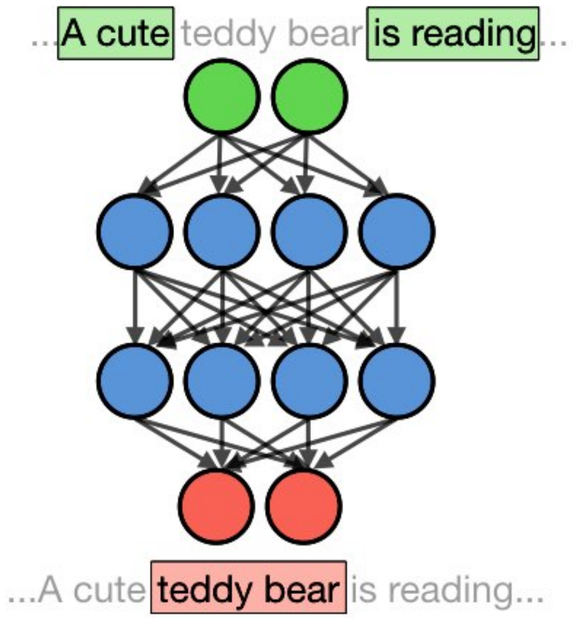
scale well with dimensionality

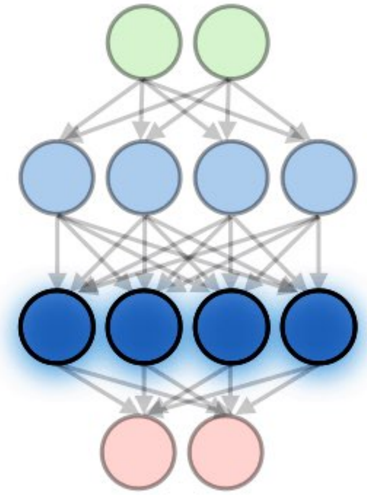
WORD EMBEDDING: WORD2VEC METHOD

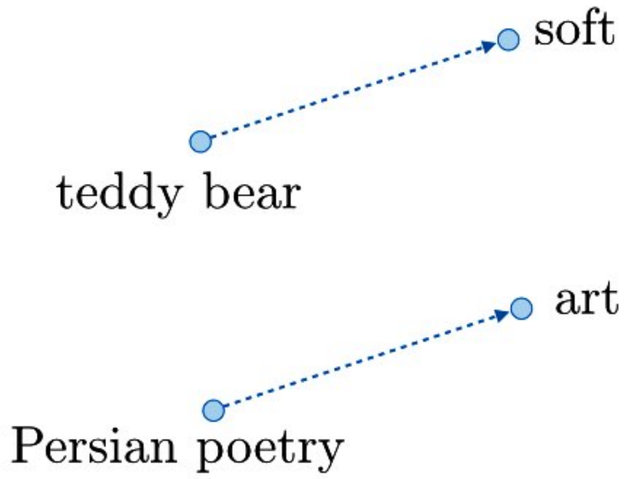
Word2Vec (Mikolov (Google) 2013) is one of the most popular word embedding techniques

a DNN is trained to associate a vector of real numbers to each word, so that words with similar or related meanings in some way are associated with "similar" vectors

- example: take a sentence as input and try to predict the probabilities that a target word is associated with the other words of the analysed sentence
- the high-level representations learnt by the network are used as embedding for the target word







Train network on proxy task → Extract high-level representation → Compute word embeddings

POSITIONAL ENCODING

- because the self-attention operation is permutation invariant, positional encoding is used to provide order information to the model when needed (ex. text or time sequences)
- implemented by adding a positional vector E, with same dimension as the input embedding, directly to te input embedding: $X \rightarrow X + E$
- two positional encoding typically considered in vanilla transformers:
  - Sinuisoidal PE: in which small constant values are added to the embeddings (has shown to provide very good performance):

$X \in R^{L \times d} =$

	0	1	2	3	⋯	d
0	x(0,0)	x(0,1)	x(0,2)	x(0,3)		x(0,d)
1						
⋮						
L	x(L,0)					x(L,d)

$E \in R^{L \times d} =$

0	0	1	0	1		1
1	$\sin(\omega_0)$	$\cos(\omega_0)$	$\sin(\omega_1)$	$\cos(\omega_1)$		$\cos(\omega_{(d-1)/2})$
⋮						
L	$\sin(L\omega_0)$	$\cos(L\omega_0)$	$\sin(L\omega_1)$	$\cos(L\omega_1)$		$\cos(L\omega_{(d-1)/2})$

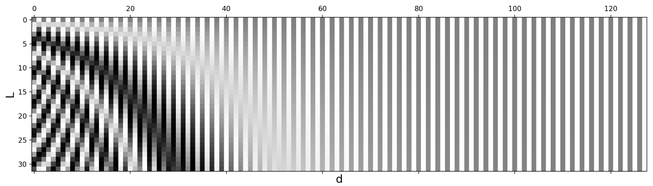
$$X \rightarrow X + E$$

word position in the input sequence embedding ∈ [0,L/2)

$$E(p,2i) = \sin(p \times \omega_i) = \sin\left(\frac{p}{10000^{2i/d}}\right)$$

$$E(pos,2i+1) = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

hidden dimension i ∈ [0,d/2)

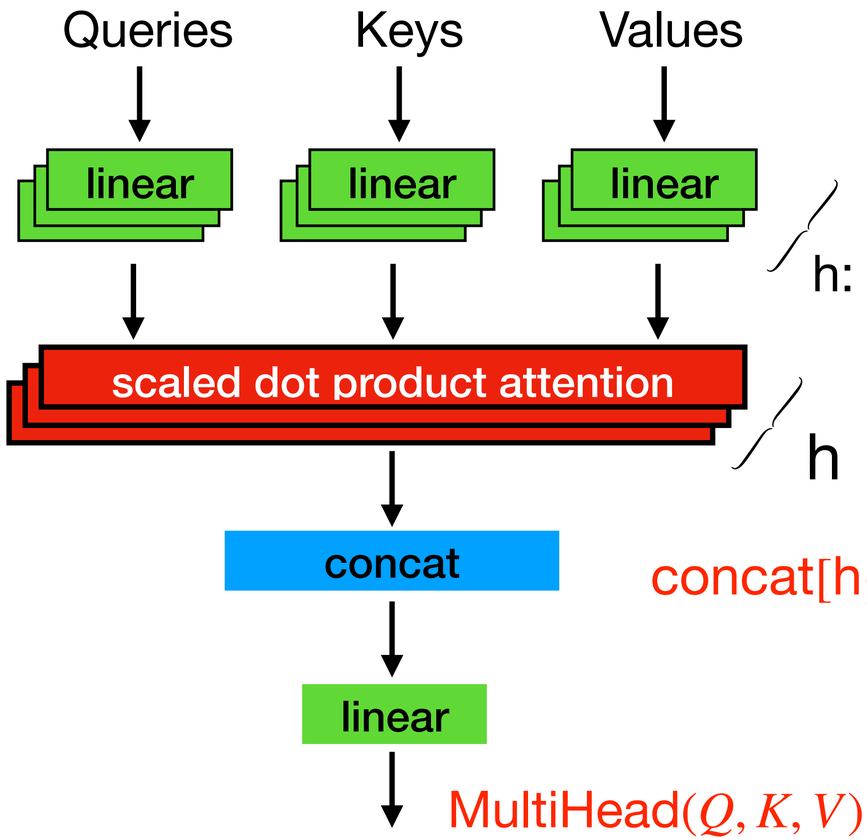


as a result, same words will have slightly different embeddings depending on where the occur in the sentence …

- Learnable PE: assigns each element of E with a learned vector which encodes its absolute position (ex. $E(p) = f_w[E_{sin}(p)]$)

MULTI HEAD (SELF) ATTENTION

- it is the core of the Transformer architecture, the structure is the same of the attention layer we have just discussed



to increase the expressive power, in a way similar to the convolution filters in a CNN, multiple sets {i=1,…,h} of keys, querys, and values are computed

$$Q_i = XW_{q,i} \qquad K_i = XW_{k,i} \qquad V_i = XW_{v,i}$$

$$\text{h}_i(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i$$

for each one a scaled dot-product attention is computed

$$\text{concat}[\text{h}_1, \text{h}_2, \ldots, \text{h}_h]$$

and finally all of them are concatenated before to apply a final projection

$$\text{MultiHead}(Q, K, V) = \text{concat}[\text{h}_1, \text{h}_2, \ldots, \text{h}_h]W_0$$

- NOTE: in the transformer N of these multi head attention blocks are organised in stacks, the first one capture correlations between pair of words/tokens, second between pair of pair of words/tokens, and so on so that eventually all the words in the sentence will be combined together …

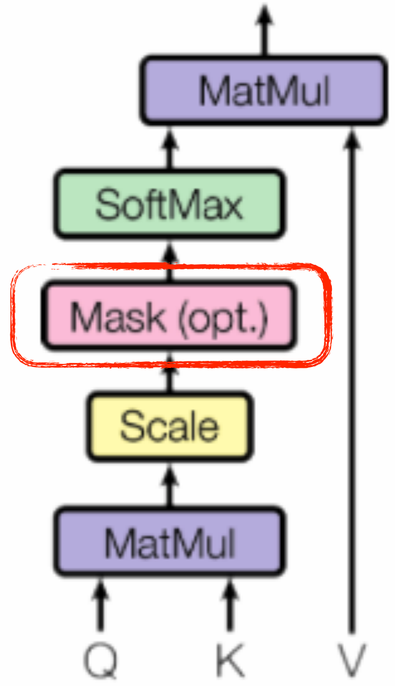
MASKED MH ATTENTION

- is a masked version of the MHA layer in which some values are masked to prevent them to be selected
- in the decoder the first MHA correlates output words with previous output words (a given output cannot depends on future outputs), so future outputs should be masked

$$\mathrm{H}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{Masked } \mathrm{H}(Q,K,V) = \mathrm{softmax}\left(\frac{Q_i K^T + M}{\sqrt{d}}\right)V$$

with M a mask matrix with zero's for unmasked elements and -∞ for masked elements (exp(-∞) = 0)



LAYER NORMALIZATION

- normalize values in each layer to have 0 mean and 1 variance to reduce covariate shifts (eg gradient correlations/dependences between each layer), making training much faster and stable

- for each hidden unit h substitute h with $\gamma(h-\mu)/\sigma$ with γ a “gain” hyper parameter that compensate for the fact that we are normalising:

$$\mu = \frac{1}{H}\sum_{i=1}^{H} h_i \qquad \sigma = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(h_i-\mu)^2}$$

- is very similar to a batch normalisation layer, with the difference that here the normalisation is done at the level of the layer (normalising across hidden units) while in BN it is done for each units normalising across batch elements. This is done in order to be insensitive to small batch sizes

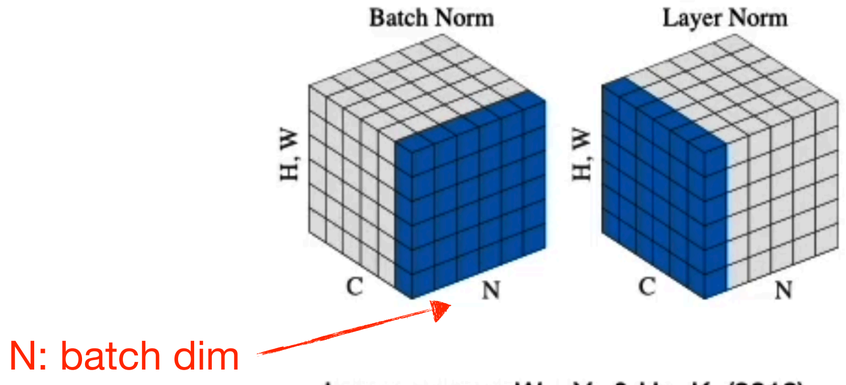


Image source: Wu, Y., & He, K. (2018).

TRANSFORMERS EVOLUTION IN LLMs

- the original transformer has spawn series of evolutions that today dominate LLMs
- three directions explored:

ENCODER-DECODER

aka

sequence-to-sequence models

TASKS

- Translation
- Summarization
- Generative question-answering

EXAMPLES

BART (META), T5, UL@, …

ENCODER ONLY

aka

auto-encoding models

TASKS

- Sentence classification
- Named entity recognition
- Extractive question-answering
- Masked language modeling

EXAMPLES

BERT (META), RoBERTa, distiBERT, …

DECODER ONLY

aka

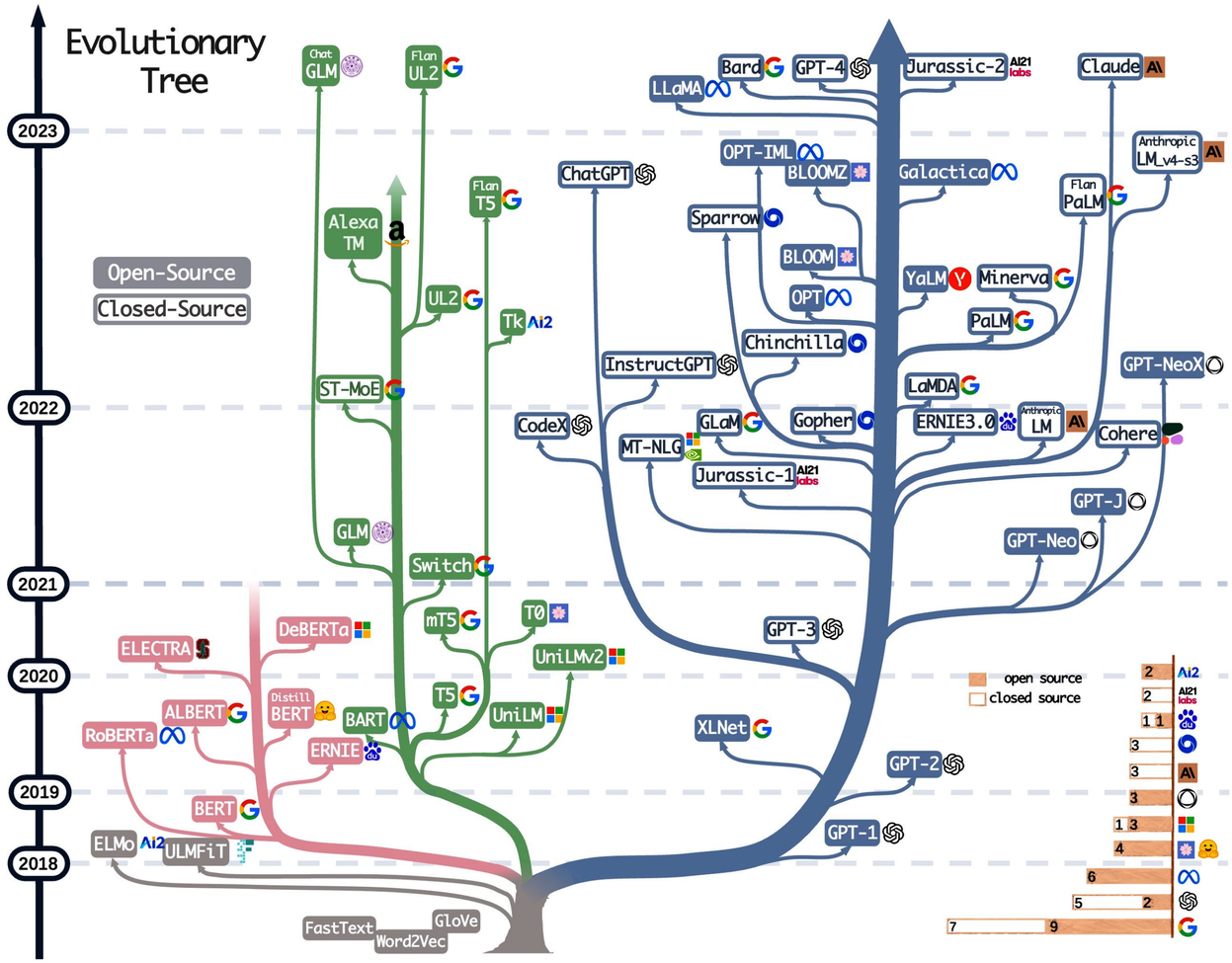
auto-regressive models

TASKS

- Text generation
- Causal language modeling

EXAMPLES

GPT3.5/Chat-GPT, GPT4, LLaMA (META), LaMDA (Google), Galactica, Bard (Google)…



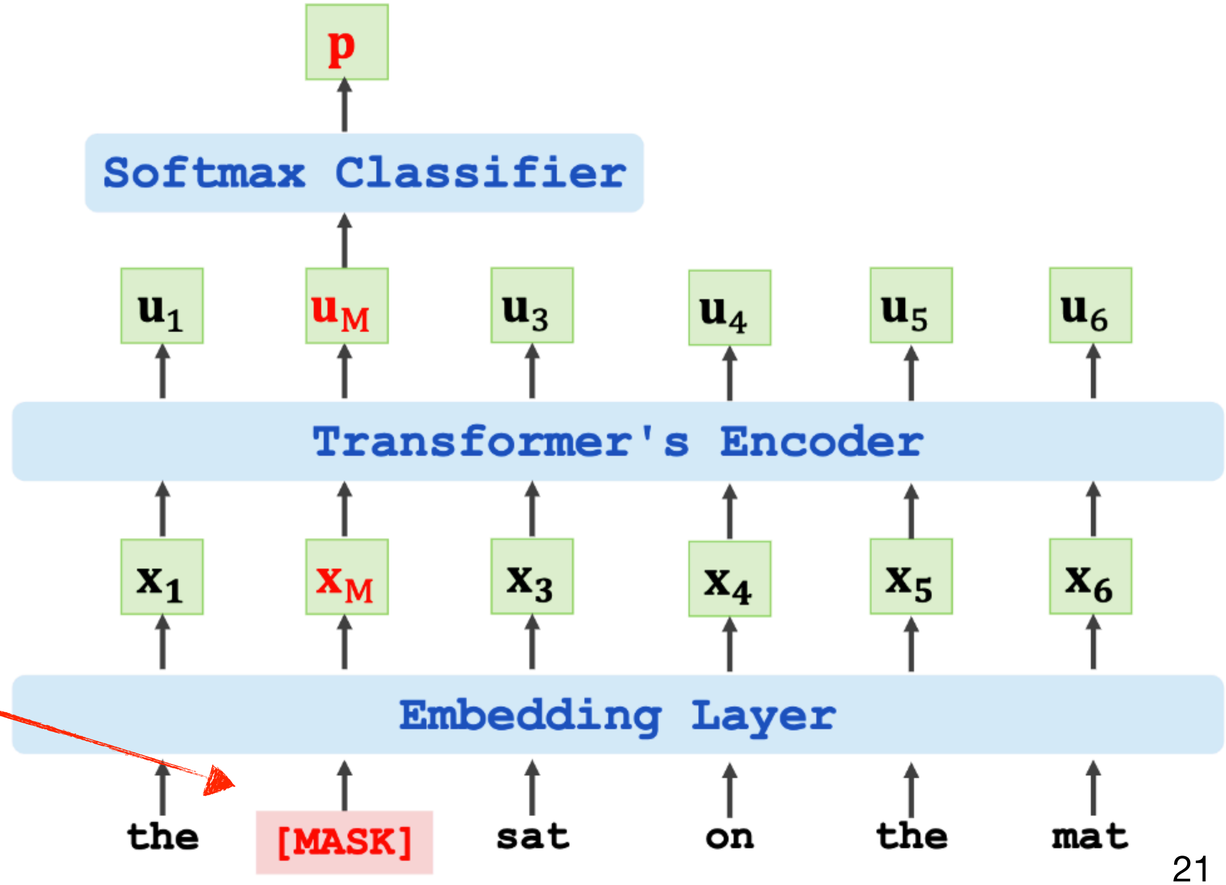
https://github.com/Mooler0410/LLMsPracticalGuide

BERT

- BERT (2019), 340M parameters: unsupervised bi-directional encoder transformer that predicts a missing word based on surrounding words by computing $P(x_t|x_1,\ldots,x_{t-1},x_{t+1},\ldots,x_N)$ or the next sentence based on a previous sentence

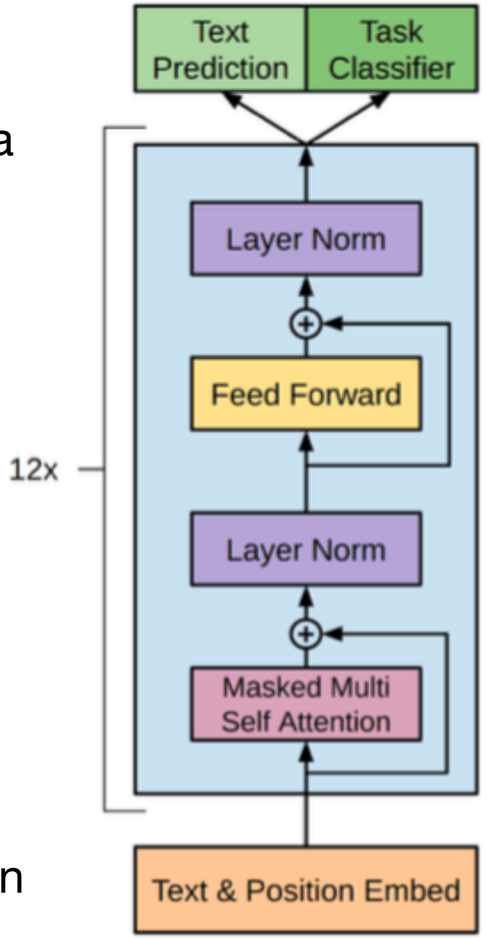
BERT: can be considered as a bidirectional transformer:

randomly mask one or multiple words in the sentence, the associated context is passed to a softmax classifier that produce the probability of classifying the correct missing word



GPT

- GPT is an unidirectional, autoregressive model, approach at the core of most influential LLMs
- it works by predicting the next token given a sequence of tokens (so outputs one token at a time at difference of BART)
- GPT3/3.5/4: latest incarnations: trained on a huge Internet text datasets (~600 GB for GPT3), and scaled up in terms of parameters wrt older version (GPT4: 8x220b pars = 1.76t pars)
- most impressive feature: it's a meta-learner, eg it has learned to learn, you can ask it in natural language to perform a new task and it "understands" what it has to do, mimicking how humans would (of course in much more rudimental way)
- very popular variant: ChatGPT, smaller version specifically designed for chatbot applications (eg. to generate natural-sounding responses in conversations. Trained on a large dataset of conversational text, that allows it to insert appropriate context-specific responses in conversations, making it more effective at maintaining a coherent conversation
  - one of the reasons behind ChatGPT's impressive performance is its training technique: reinforcement learning from human feedback (RLHF)



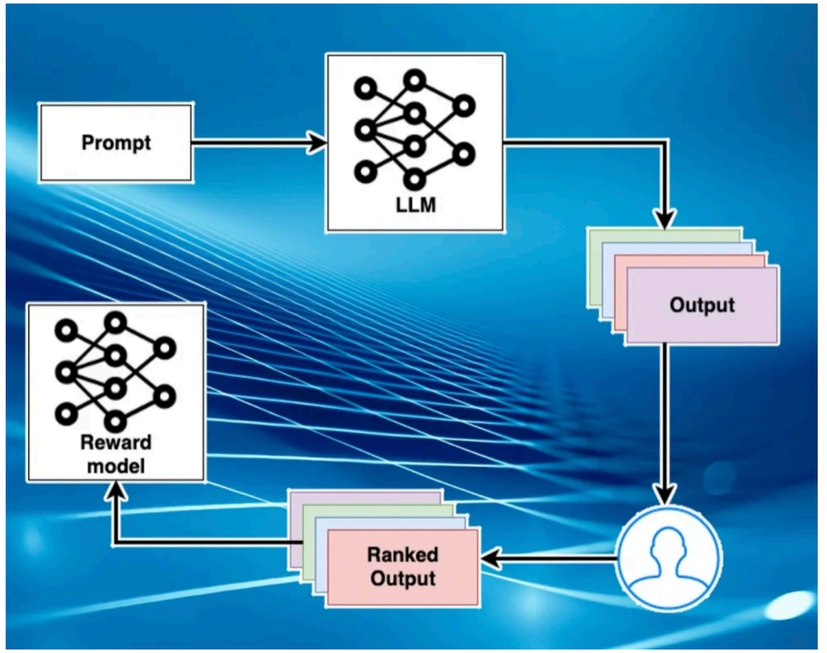
RLHF

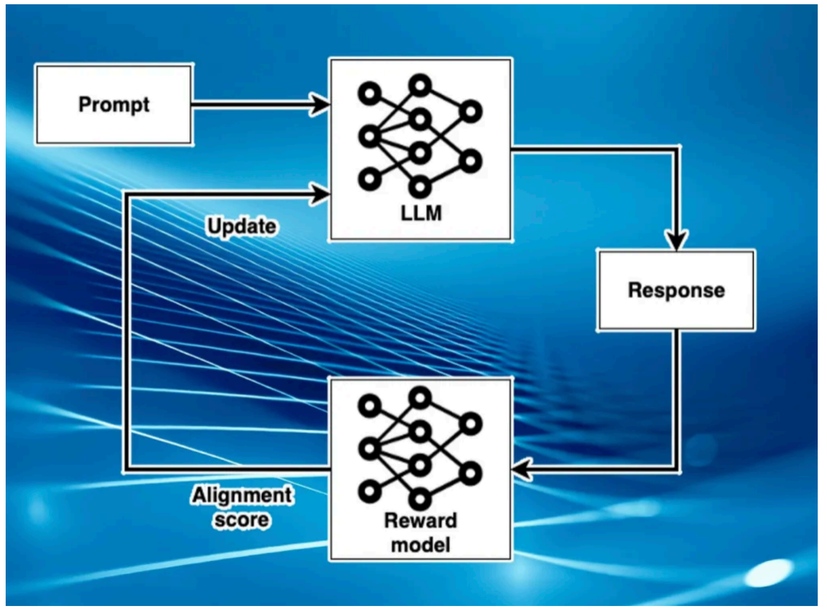
- reinforcement learning from human feedback consists in three phases:

  1. pre-train the LLM model with self-supervision (ex. fine-tuning of GPT3.5)

  2. create a reward model for the RL system: train a second model (based on a LLM model) that takes in the text generated by the main model and produces a quality score (labels are produced by humans)

  3. create a reinforcement learning loop. A copy of the main LLM becomes the RL agent. In each training episode, the LLM takes several prompts from a training dataset and generates text. Its output is then passed to the reward model, which provides a score that evaluates its alignment with human preferences. The LLM is then updated to create outputs that score higher on the reward model





VISION TRANSFORMER

- the very same philosophy of the transformer architecture can be applied to vision, signal analysis, point-cloud analysis, etc. tasks
- Vision Transformer (ViT): proposed in 2021 by A. Dosovitsky et al. in arXiv:2010.11929
- Simple idea:
  - split the images into patches
  - vectorise the patches into flat vectors
  - add positional encodings vectors to preserve patch positions in the original image
  - feed the embedding to a transformer encoder tailored for a classification task

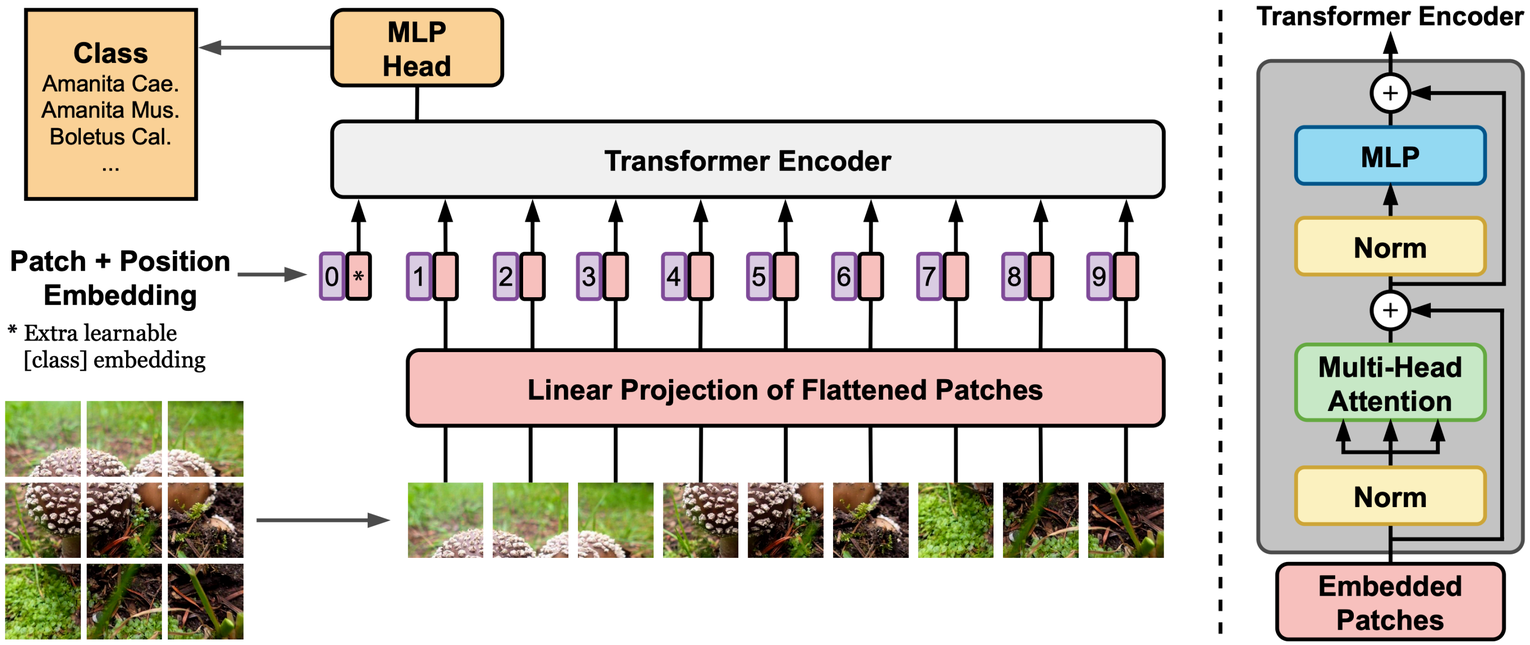
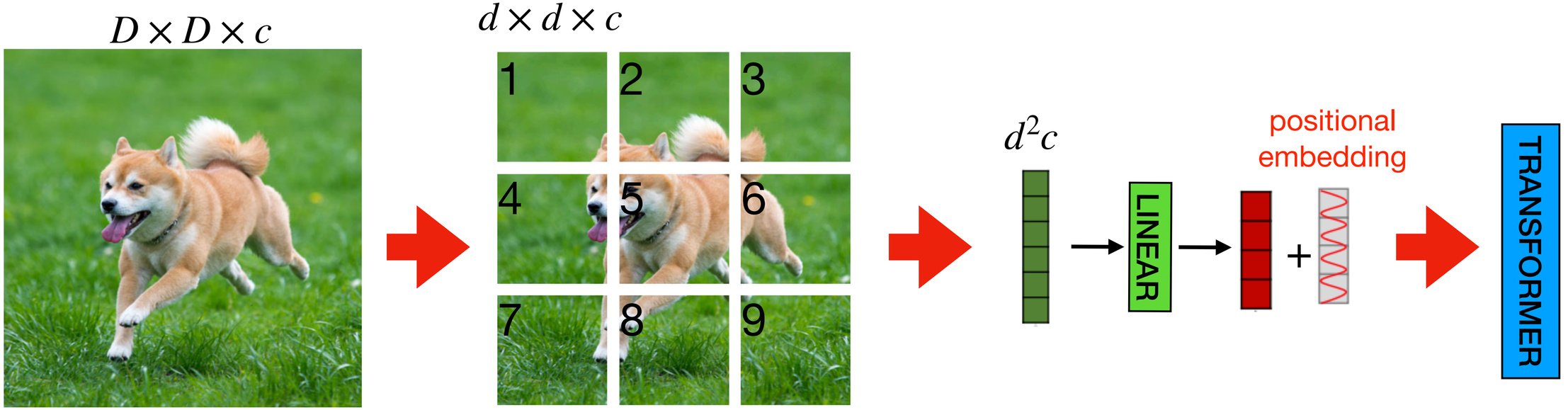


IMAGE PATCHING AND VECTORISATION

- patches can overlap or not (the original paper uses not overlapping patches)



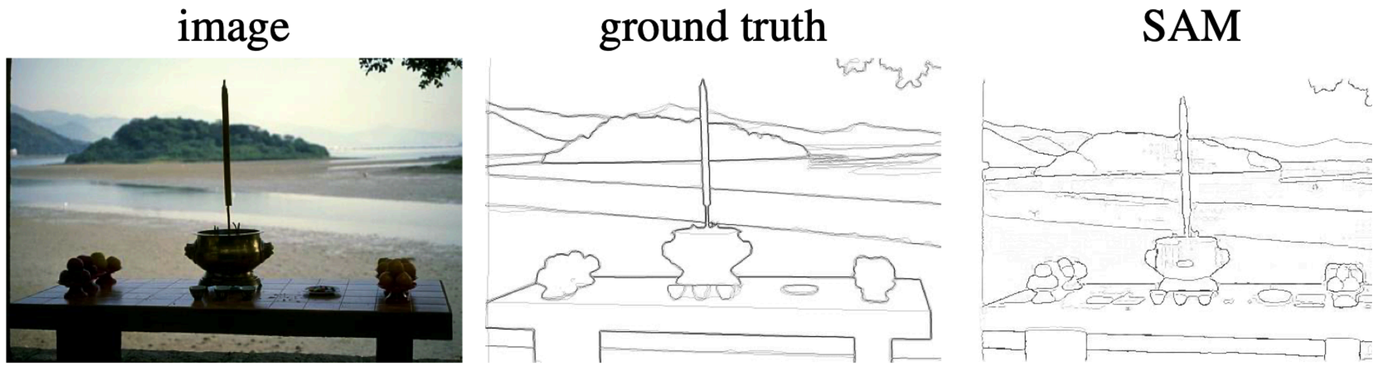
- ViT has much less image-specific inductive bias than CNNs
  - in CNNs, locality, two-dimensional neighbourhood structure, and translation equivariance, are baked into each layer throughout the whole model
  - in ViT, only MLP layers are local and translationally equivariant, while the self-attention layers are global. The 2D neighborhood structure is only used when cutting the image into patches while the position embeddings is only 1D and the 2D spatial relations between the patches have to be learned. Consequently, ViTs require more data for pretraining to acquire useful "priors" from the training data

SEGMENTATION WITH ViT

- SAM: Segment Anything by Meta (arXiv:2304.02643)

a promptable segmentation system with zero-shot generalisation to unfamiliar objects and images (eg w/o fine-tuning/retraining)

- image encoder (based on pre-trained ViT) + prompt encoder (Transformer) + mask decoder (Transformer) system

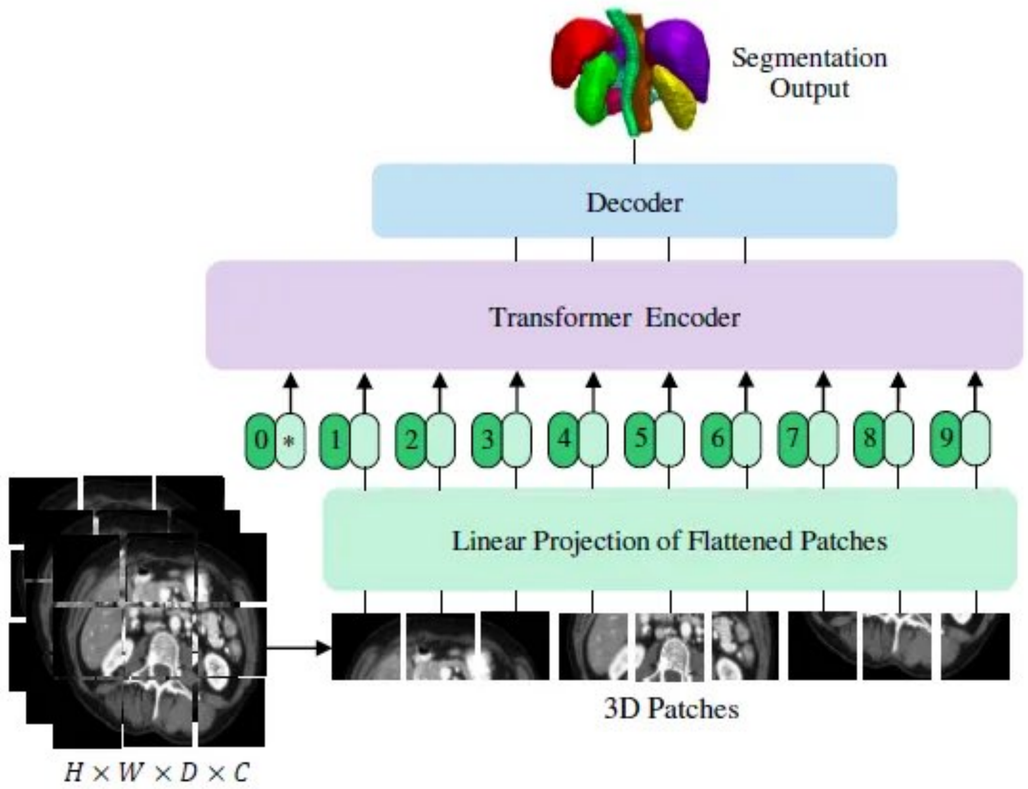


- UNetr (and Swin UNetr): algorithm by NVIDIA

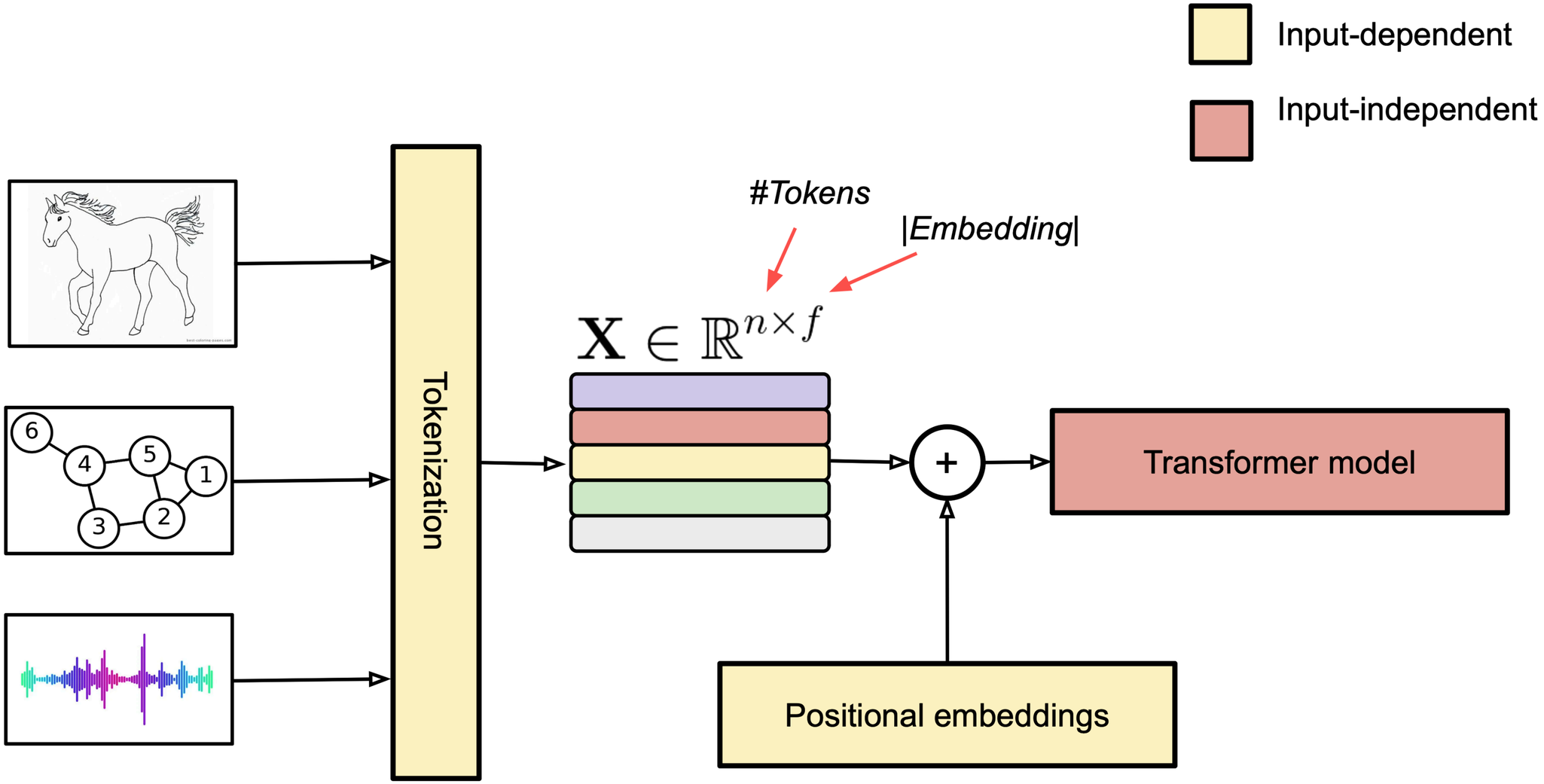
  (H. Hatamizadeh et al.: UNetr)

it is basically a mix between a 3D ViT and an UNet architecture

- the transformer encoder learns sequence representations of the input volume
- the encoder is directly connected to a decoder via skip connections at different resolutions like a U-Net to compute the final semantic segmentation output



MULTI-MODALITY TRANSFORMERS



credits: S. Scardapane

KEEP IN TOUCH …

SCAN ME

This activity is partially supported by PNRR MUR project PE0000013-FAIR and by ICSC – Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing, funded by European Union – NextGenerationEU

Future Artificial Intelligence Research

ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing