

QUICK INTRODUCTION TO TRANSFORMERS

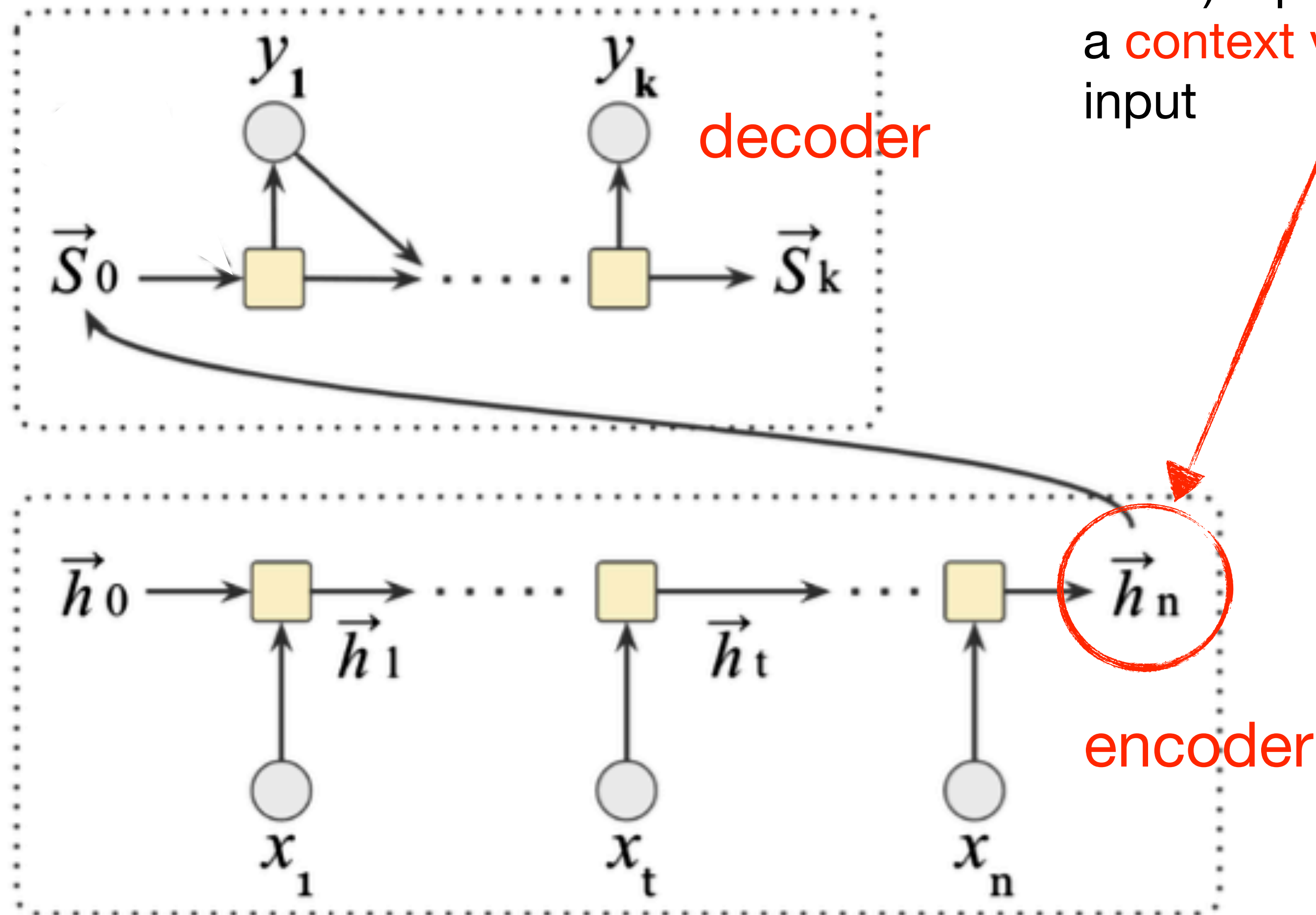
S. Giagu - 3rd ML_INFN Hackathon: Advanced Level

ATTENTION MECHANISM

- Several examples in DNN in which the whole task to be learned is subdivided in sub-tasks which apply to local elements of the input:
 - RNN example: to translate each element of a sequence
 - CNN example: to classify each pixel in a image segmentation (eg classification pixel by pixel)
 - GNN example: to predict the target for each node in a graph
- to solve the sub-task in all these cases the model learns to form an **internal representation of the input** that acts as **context** for the sub-task
- ideally this **context vector** should contains information from the entire input, however:
 - fixed length vectors scales quite poorly with input size (number of subtasks grow), like in the analysis of long sequences
 - either the size of the context vector grows or it will not have the capacity to represent all the relevant information leading to a degradation in performance
- this **clash with computational resources**
- **idea:** 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. The **attention mechanism** provides a way to identify such parts ...

REMINDER: SEQ-TO-SEQ LSTM/GRU

Encoder-Decoder RNNs



seq2seq models (used for example in machine translation tasks) represents a first example of such attempt to create a **context vector** (the cell state of the LSTM/GRU) from the input

use of LSTM or GRU cells allows to “memorise” relevant terms that are far from the current element in the sequence and that are crucial to solve the task

limitation: **LSTM/GRU** becomes ineffective for very long sequence, unless implemented in complex StackedRNN architectures that are impossible to train in an acceptable time due to the recursive (i.e. non parallelizable) intrinsic structure of a RNN

ATTENTION

- **intuitive idea:** one forms a representation for the entire input, but **different parts of the input are weighted differently** according to the task at hand. By making **the weights a learnable component**, the network can learn to attend to the relevant parts of the input
- **example:** in a NLP translation task, attention will work by **aligning each words in the output sequence** (translated text) with **some words in the input sequence that give context to the translation**. These words are not necessarily aligned in the original order, they can aligned in different order to take into account that words order may be for example different in different languages ...

context vector for the i-th output word
(or sequence element in general)

$$C_i = \sum_j \alpha_{ij} h_j$$

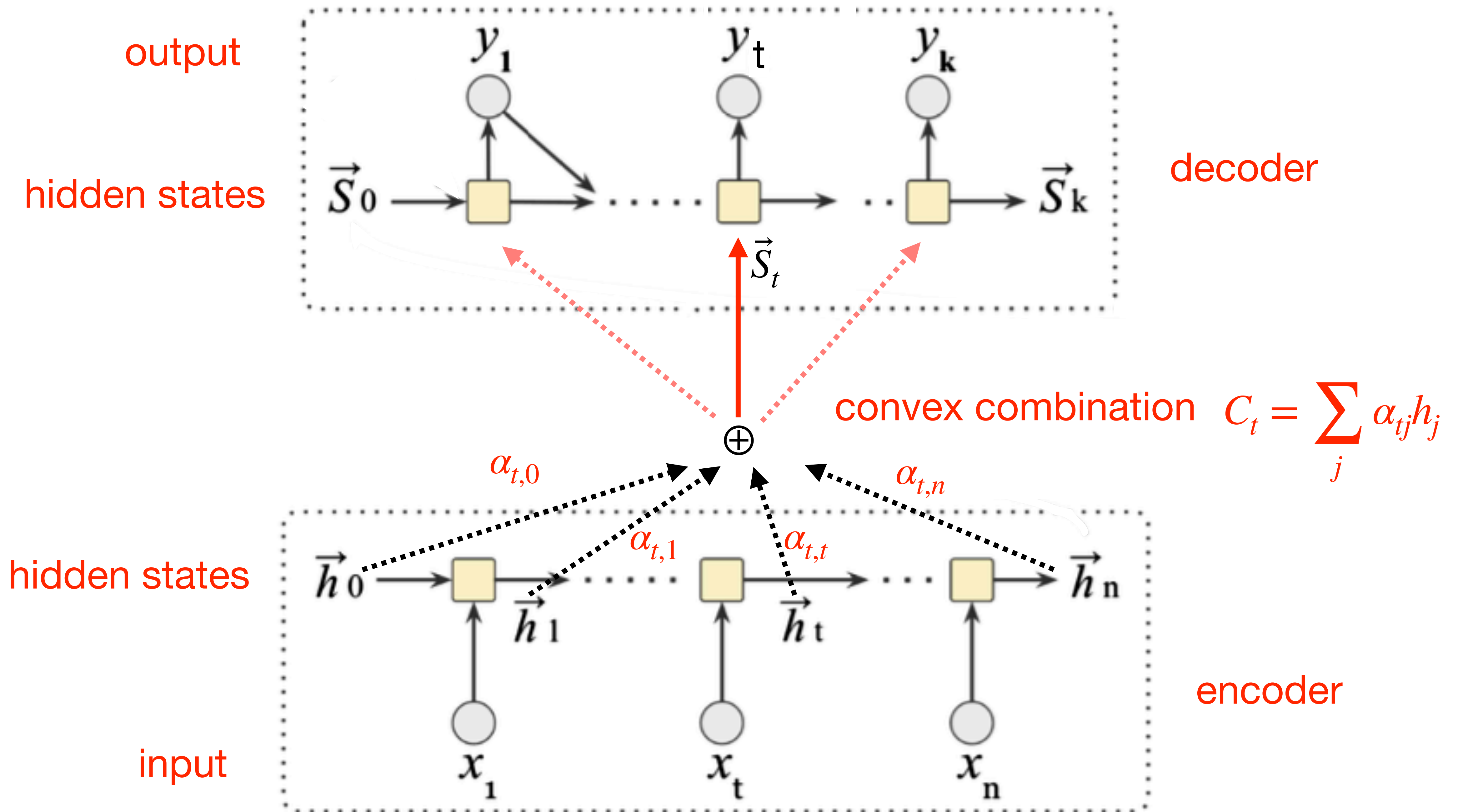
alignment weight between
input encoding h_j and output
encoding s_i

$$\alpha_{ij} = \text{softmax}(\text{alignment}(s_i, h_j)) =$$

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

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

$$= \frac{\exp[\text{alignment}(s_i, h_j)]}{\sum_k \exp[\text{alignment}(s_i, h_k)]}$$

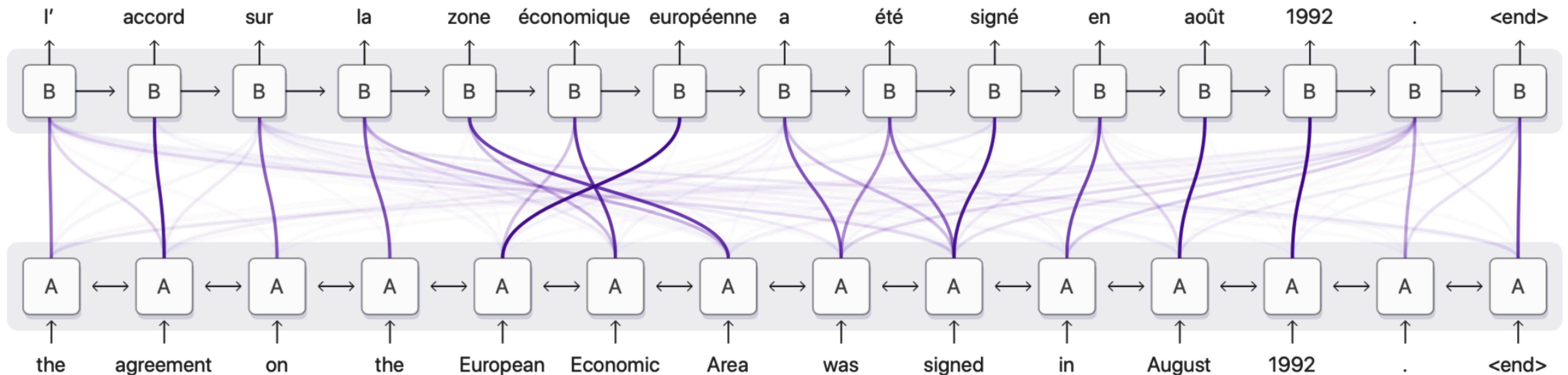


RNNSearch: BIDIRECTIONAL RNN WITH ATTENTION

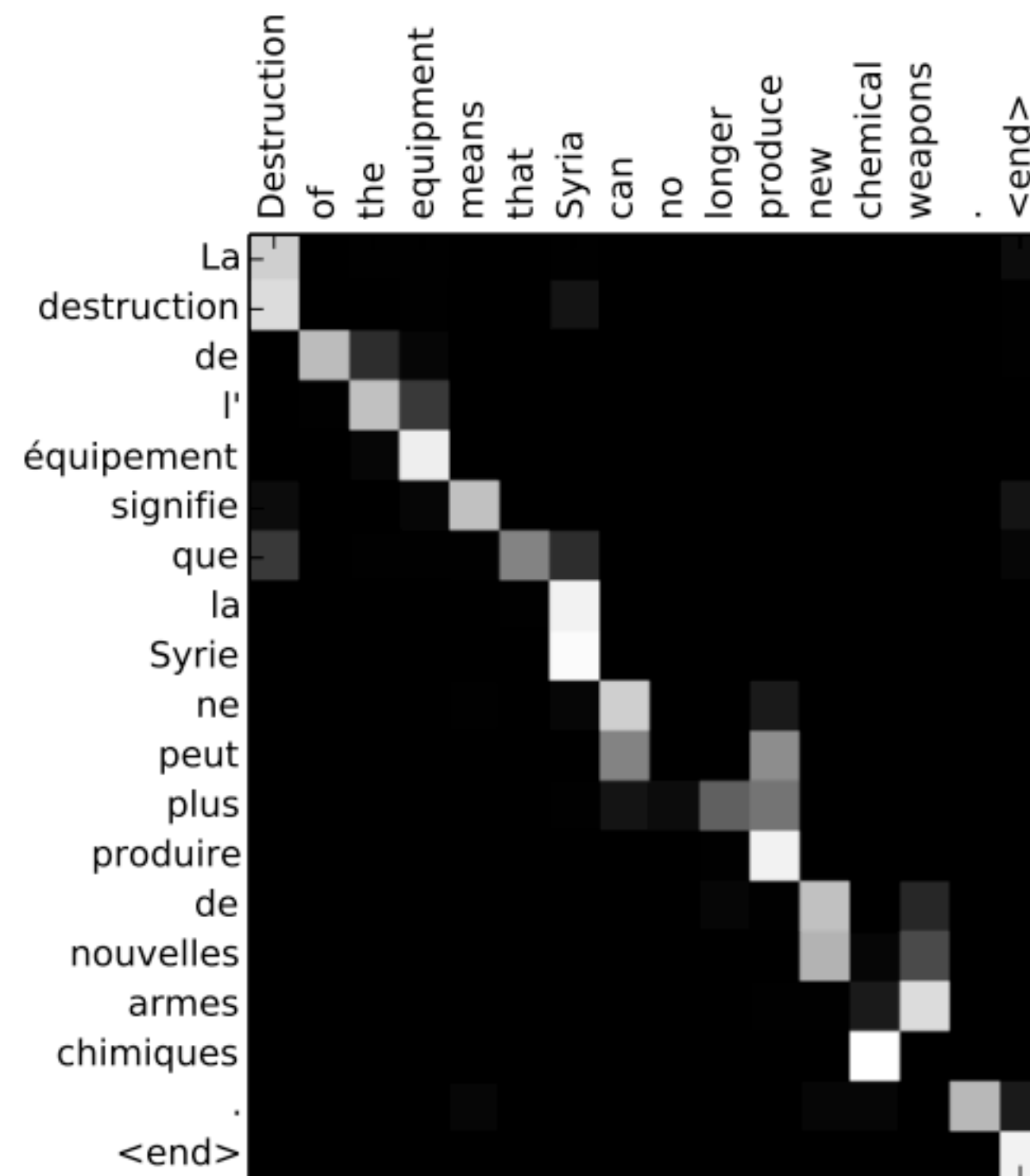
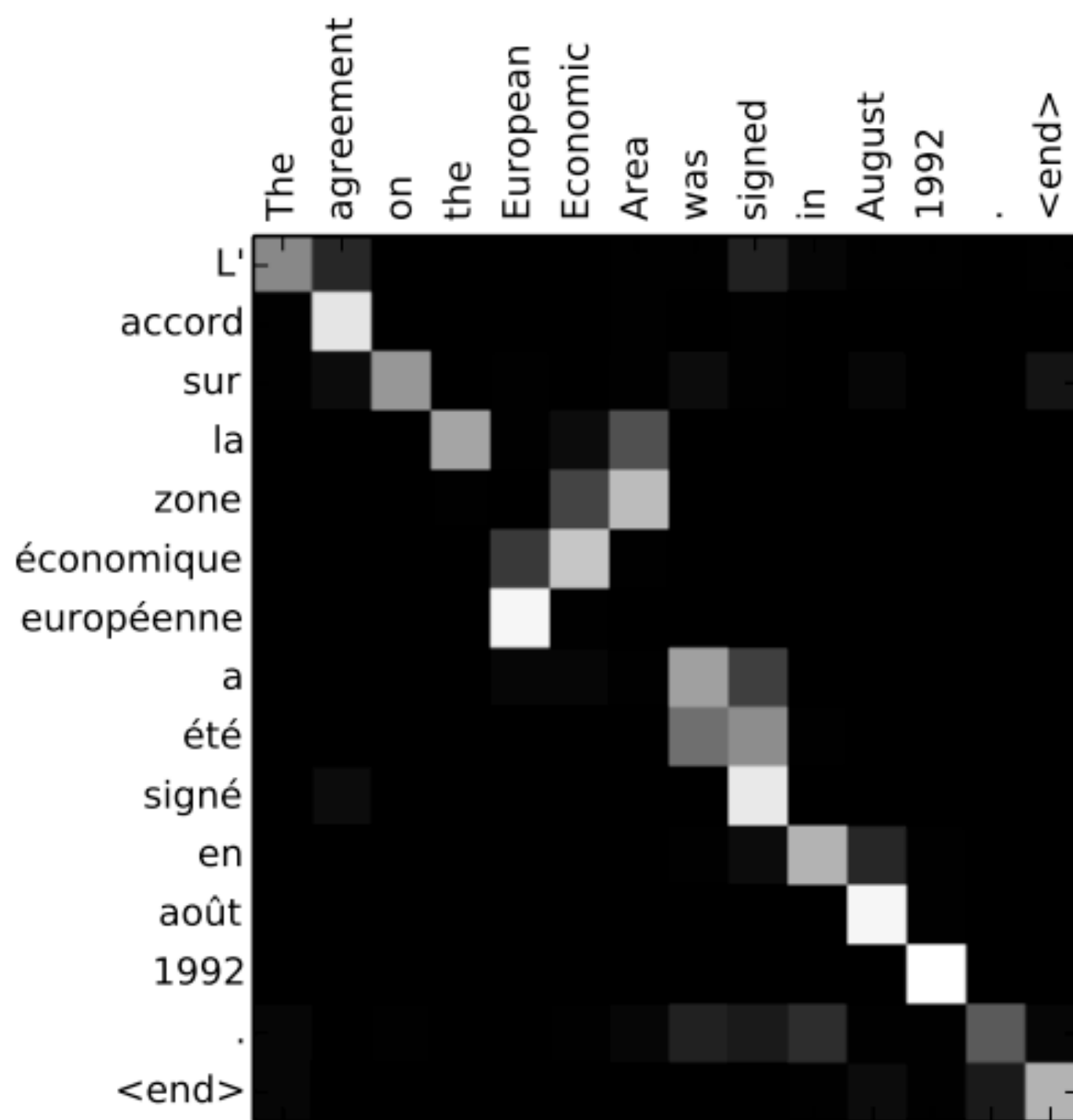
- attention idea implemented for the first time in a model (RNNSearch: D. Bahdanau, K. Cho and Y. Bengio, ICLR 2015) which made a breakthrough in machine translation by combining a bi-directional RNN with an **additive attention mechanism**

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

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

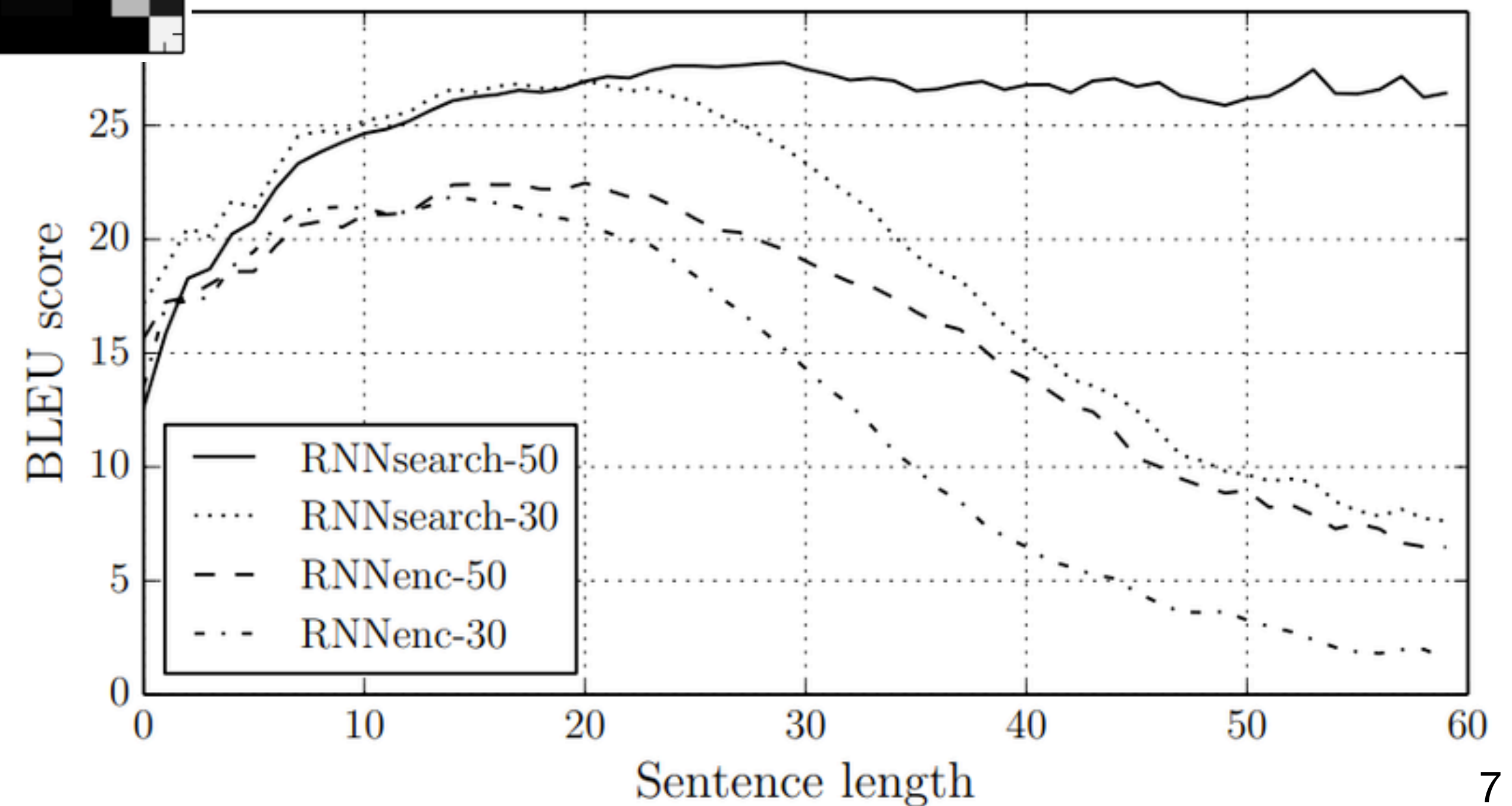


attention weights in a seq-to-seq problem of translation from ENG to FR



alignment matrices

BLEU score: percentage of translated words that appear in the ground truth



RNNSearch: D. Bahdanau, K. Cho and Y. Bengio, ICLR 2015

DIGRESSION: ATTENTION AND NOTION OF SPARSITY OF INTERACTIONS

- the attention mechanism is a **generalisation of the assumption of locality used in CNN** with the concept of sparsity of interactions
- this can be intuitively understood by considering the k-NN algorithm:

$$g(x) = \frac{1}{k} \sum_{i \in k\text{-nn}(x)} y_i$$

in a regression task returns the average of the values of the closest k-points according to a defined distance $d(x, x_i)$

- **$g(x)$ is considered “sparse” as only depends on k points of the entire dataset**
- the **attention makes the operation of selection the k-nn points differentiable** and useable by summing over all points and weighting them with the distance d :

$$g(x) = \sum_i d(x, x_i) y_i = \sum_i e^{-\beta \|x - x_i\|_2} y_i \quad \text{Nadaraya-Watson kernel estimator}$$

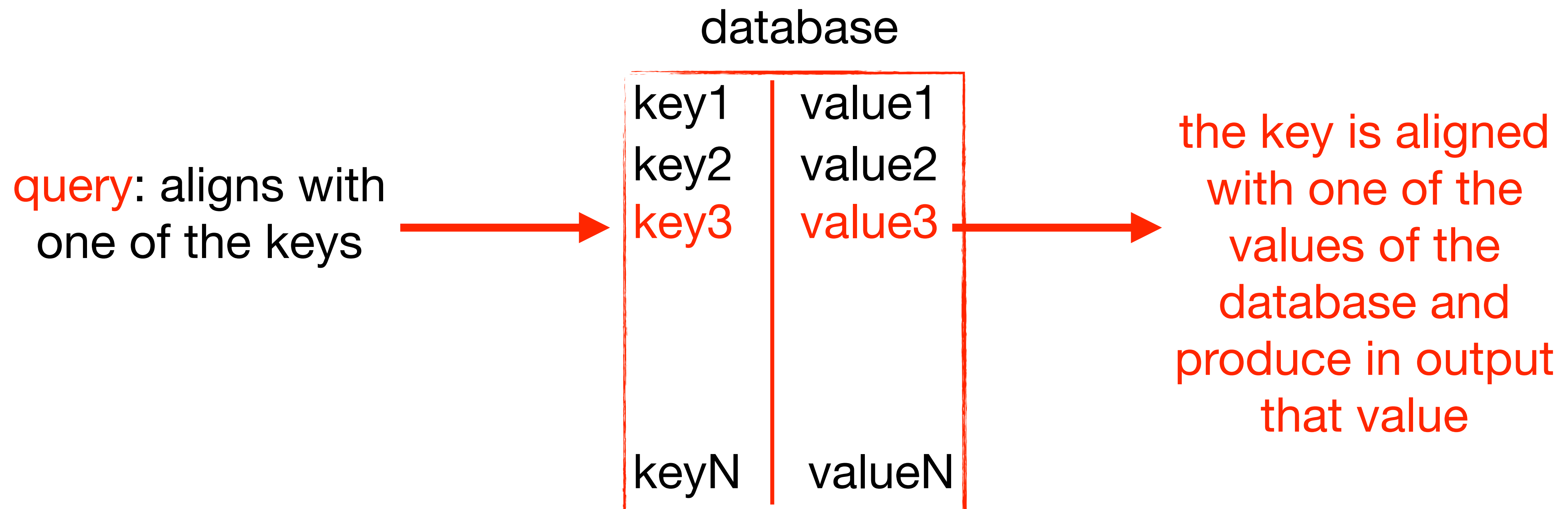
the attention mechanism makes this estimator a convex sum using the softmax

SCALED DOT-PRODUCT ATTENTION AND TRANSFORMERS

- **Transformers** are recent DNN architectures based on the attention mechanism that have gradually replaced RNNs in mainstream NLP tasks, and that can also often compete/surpass (**when trained with very large datasets**) CNNs and RNNs in vision and in time-domain related tasks
- Compared to RNN, Transformers:
 - facilitate the learning of long range sequences
 - don't need recurrence:
 - **no gradient vanishing or explosion** problems
 - typically **need fewer training steps** (contrarily to RNNs that due to recurrence when unrolled are very deep networks), and **can be easily parallelised on GPUs** (while recurrence is intrinsically serial)
- The core ingredient of Transformers is the so called **multi head (self) attention layers based on scaled dot-product alignment**

ATTENTION MECHANISM AS A DB RETRIEVAL TECHNIQUE

- the attention mechanism can be also seen in a different way, as a technique that mimics the retrieval in a database of a value \mathbf{v} based on a query \mathbf{q} and on a key \mathbf{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:



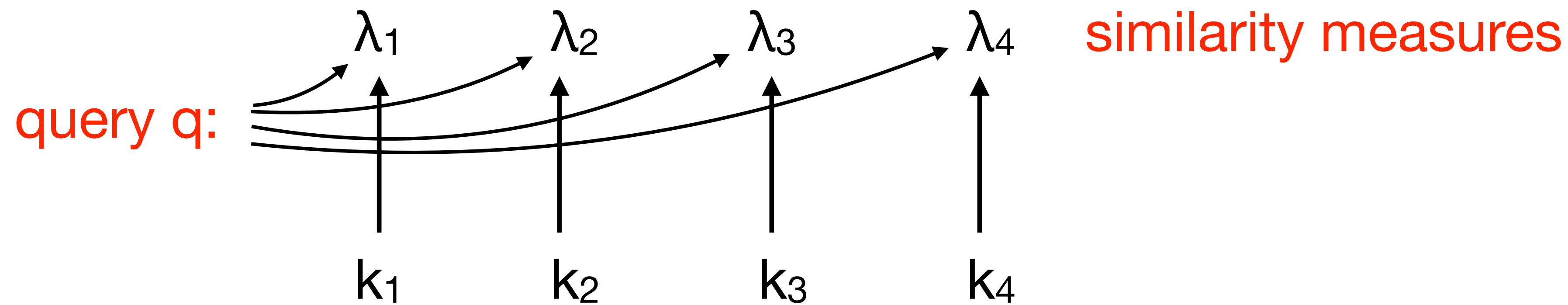
- the **dotted attention mechanism** mimics this via a neural network architecture:

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

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

the value associated to the
key k_i

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



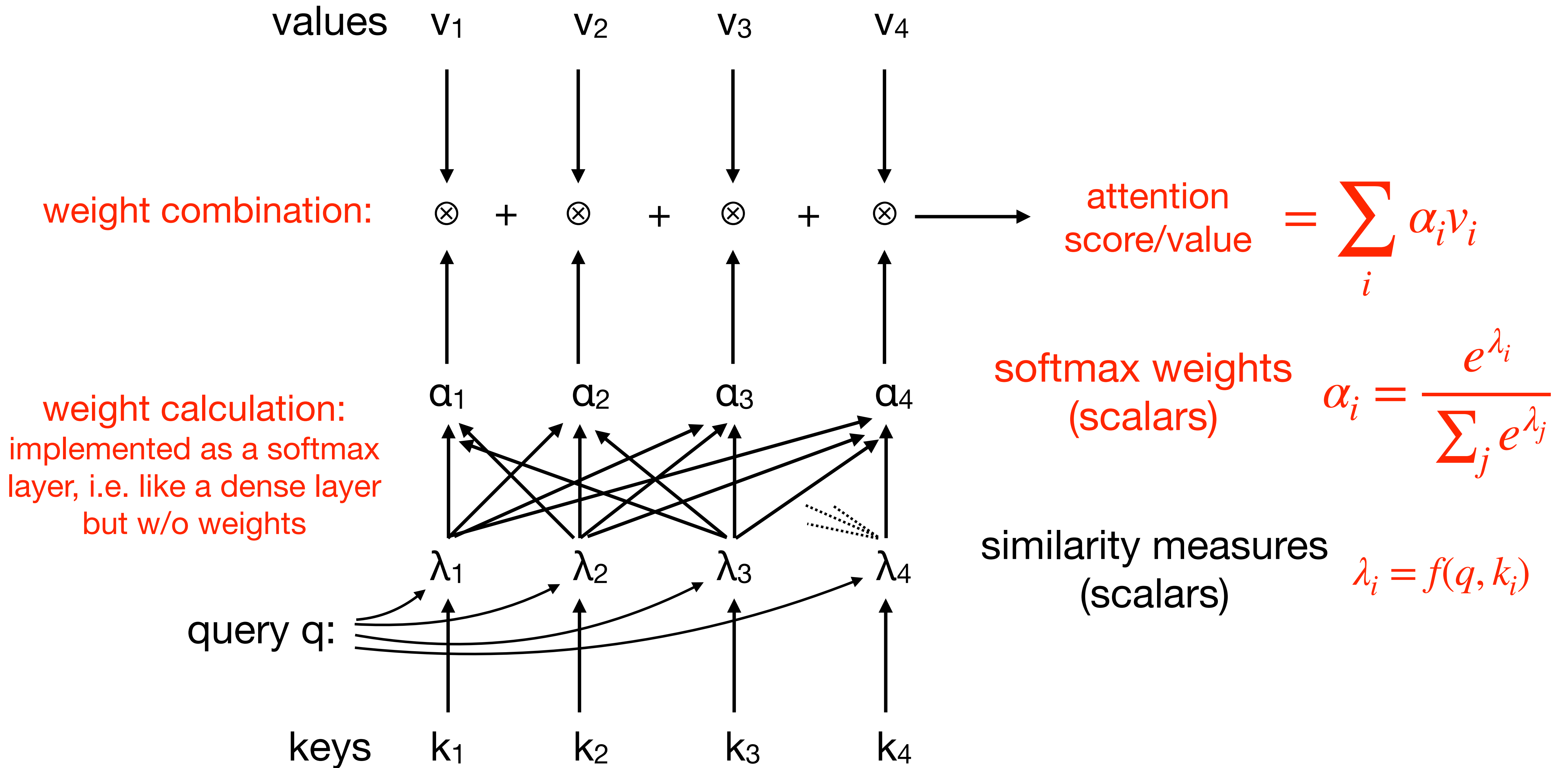
several possibilities for the similarity measure

$$\lambda_i = f(q, k_i) = \begin{cases} q^T k_i & \text{dot product} \\ \frac{q^T k_i}{\sqrt{d}} & \text{scaled dot product} \\ W_q q^T W_k k_i & \text{general (scaled) dot product} \\ W_q q + W_k k_i & \text{additive similarity (as in the RNNSearch)} \end{cases}$$

much more efficient than additive similarity

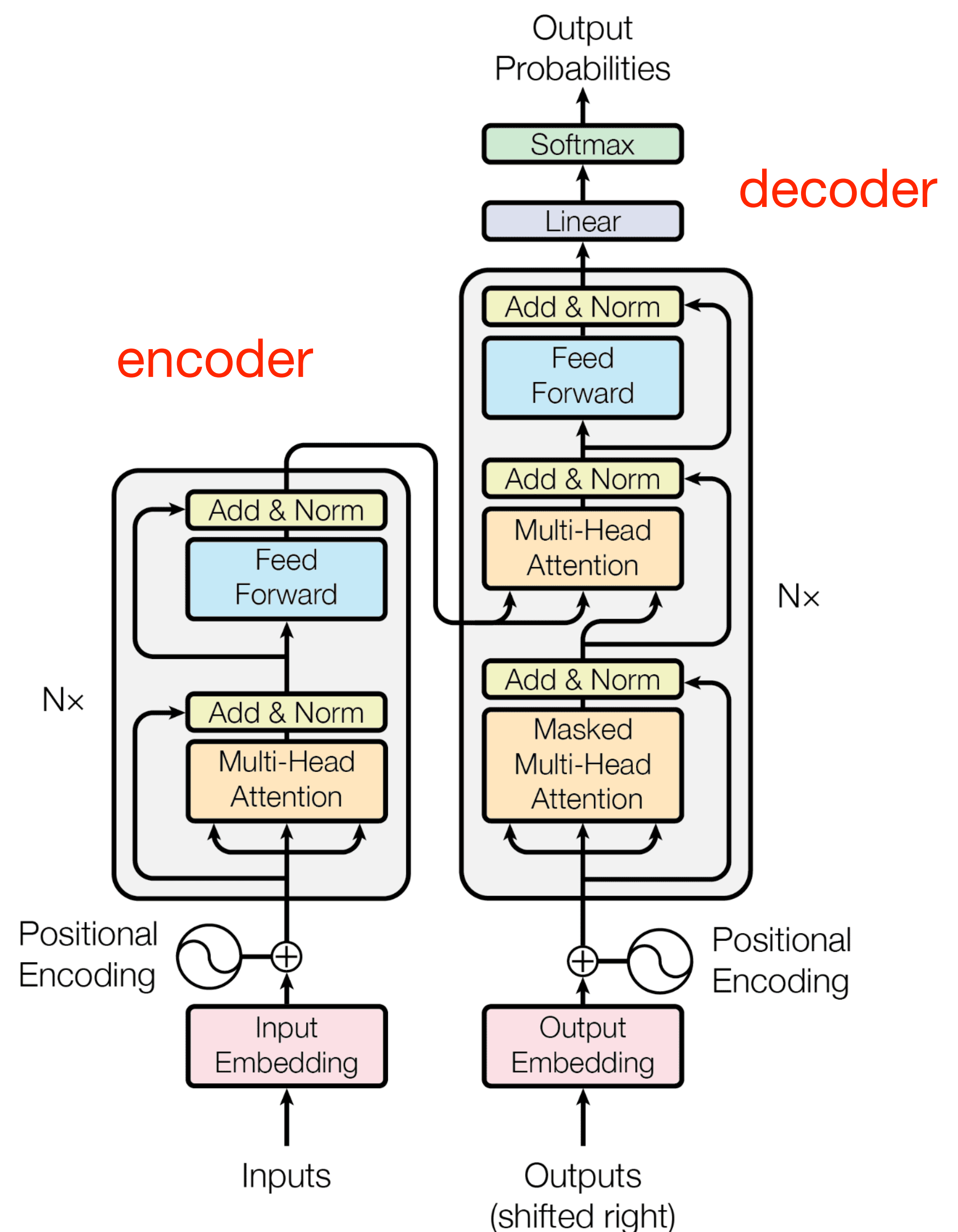
project the query on a new space (for example to be in the same space from the point of view of the similarity as the key) via a learnable transformation

- as example in a machine translation task we may have:
 - query i** : hidden representation vector for the i -th output word: \mathbf{s}_i
 - key j** : hidden representation vector for the j -th input word: \mathbf{h}_j
 - value j** : again the hidden representation vector for the j -th input word: \mathbf{h}_j
- this way the attention allows to compare each output word with a context vector that takes into account all the input words



TRANSFORMER ARCHITECTURE

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



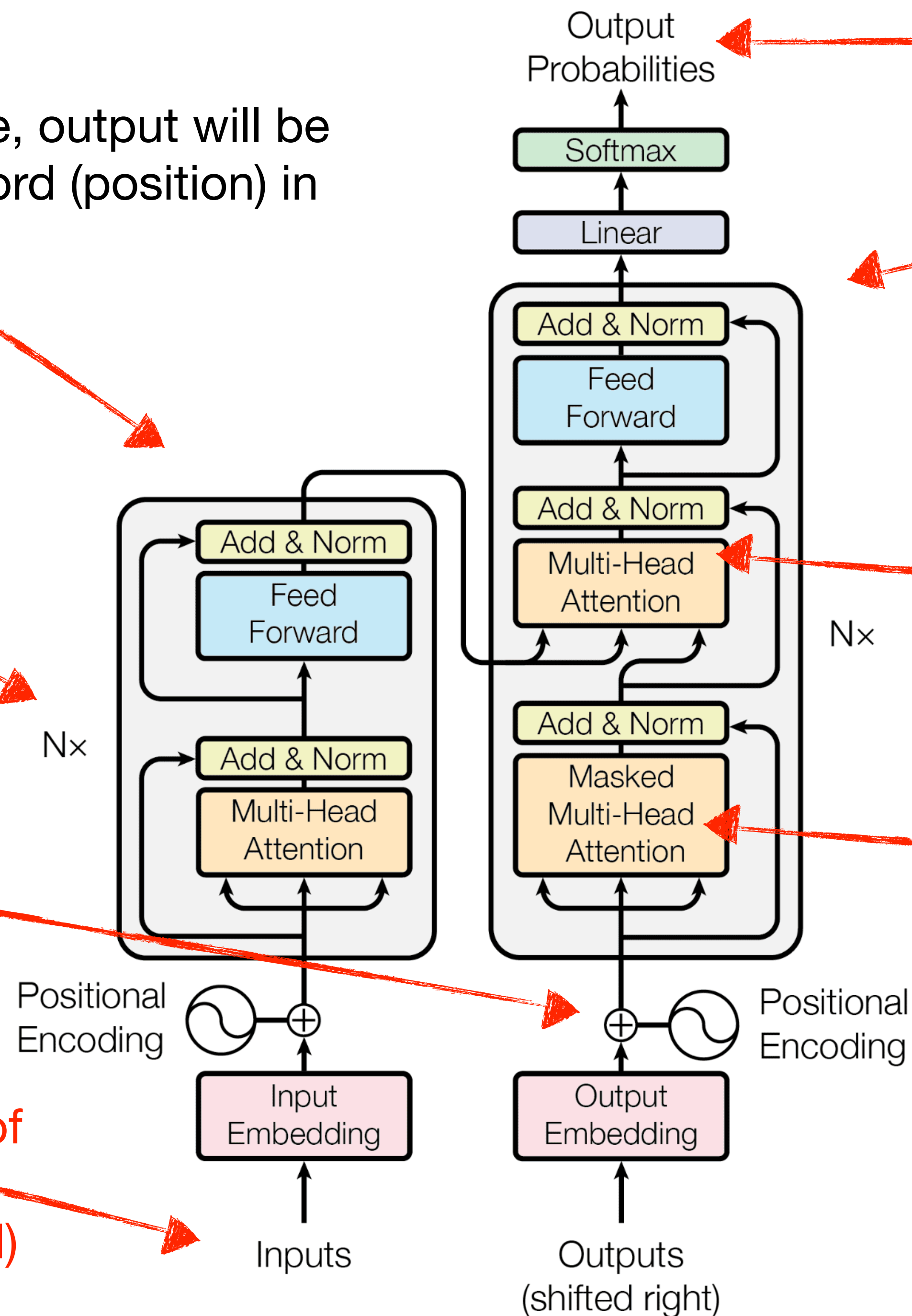
encoder:

encode the input sequence, output will be an embedding for each word (position) in the input

there are N of these blocks organised as a stack

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: distribution over the words dictionary

decoder:

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

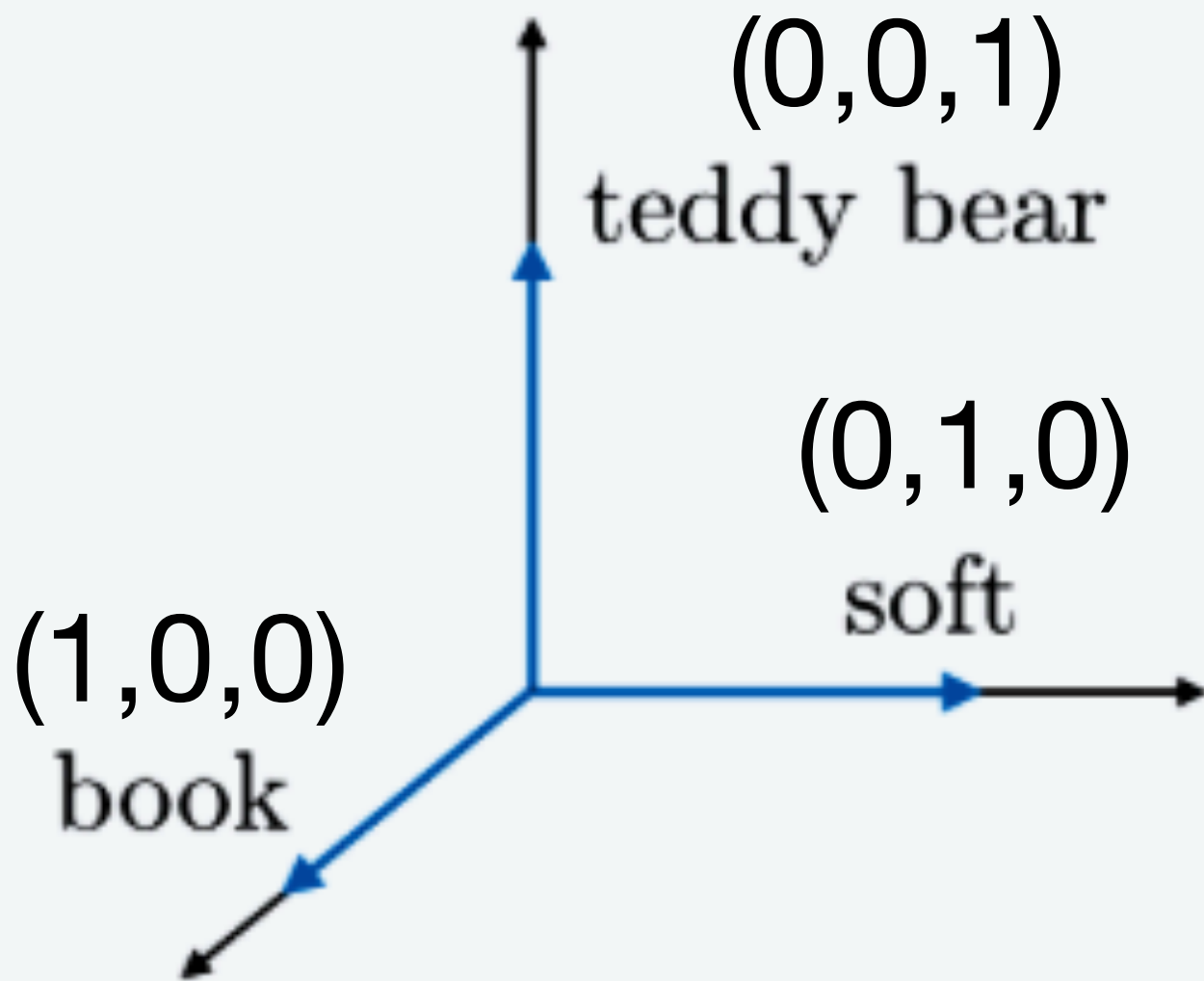
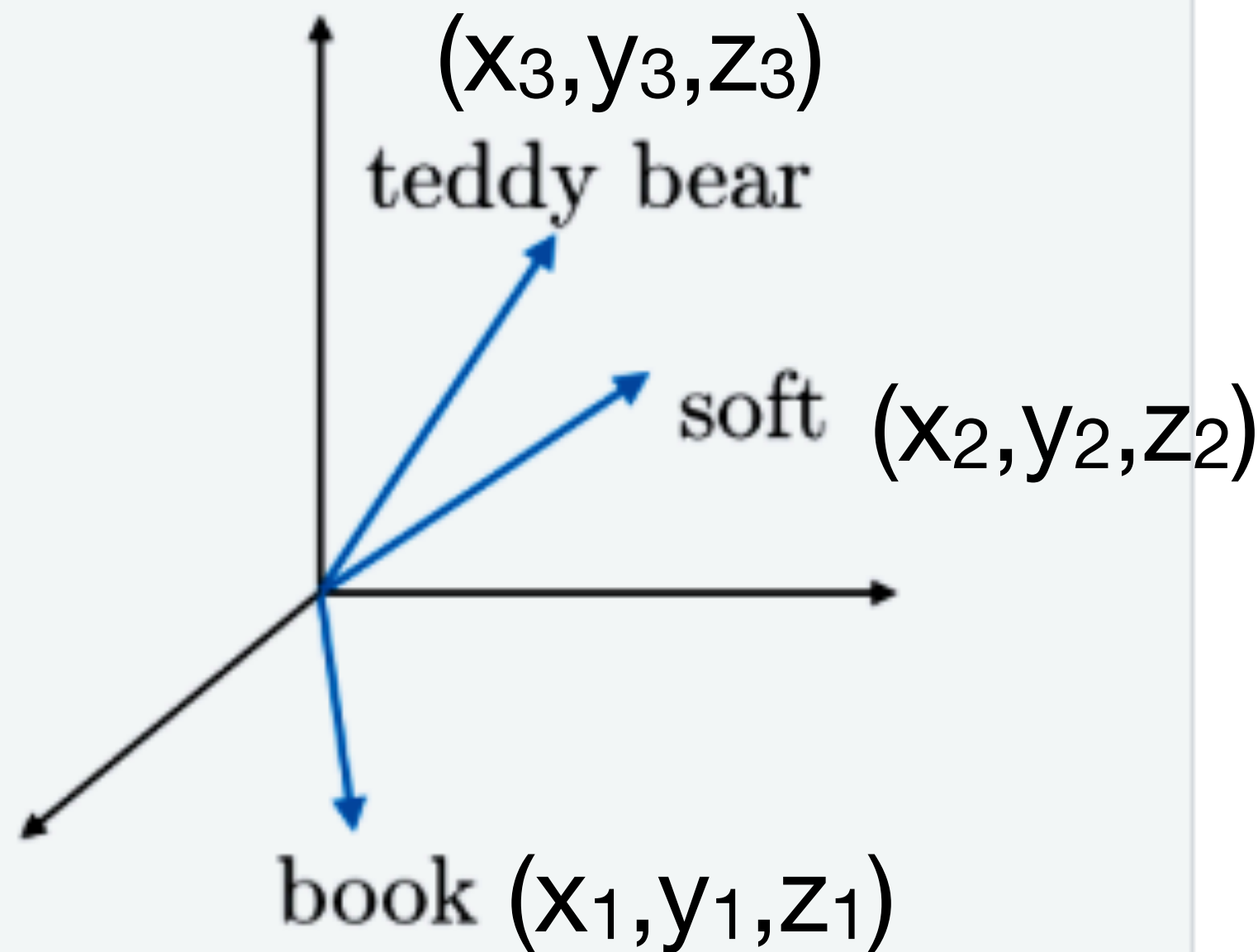
attention layer that combines output words embedding with input words embeddings

self attention layer that combines output words with **previous** output words (w/ 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
	
<ul style="list-style-type: none">• simple• naive approach, no similarity information	<ul style="list-style-type: none">• 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

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:
 - feed with a vector made by the embedding vectors of every words in the sentence
 - the MH attention compute the self attention between every position and every other position in the input vector, treating each word as a query and find some keys that corresponds to the other words on the sentence and make a weighted convex sum of the values (taken to be equal to the keys) to produce a better embedding that merge informations from pair of words
 - to increase the expressive power, in a way similar to the convolution filters in a CNN, multiple sets $i=1, \dots, h$ of keys, queries, and values are computed:

$$Q_i = XW_{q,i}$$

$$K_i = XW_{k,i}$$

$$V_i = XW_{v,i}$$

for each one a dot product attention is computed:

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

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

$$\text{MultiHead}(Q, K, V) = \text{concat}[h_1, h_2, \dots, h_h] W_0$$

- **NOTE:** in the Transformer there are N of these multi head attention blocks organised in stacks, the first one capture correlations between pair of words, second between pair of pair of words, and so on so eventually all the words in the sentence will be combined together ...

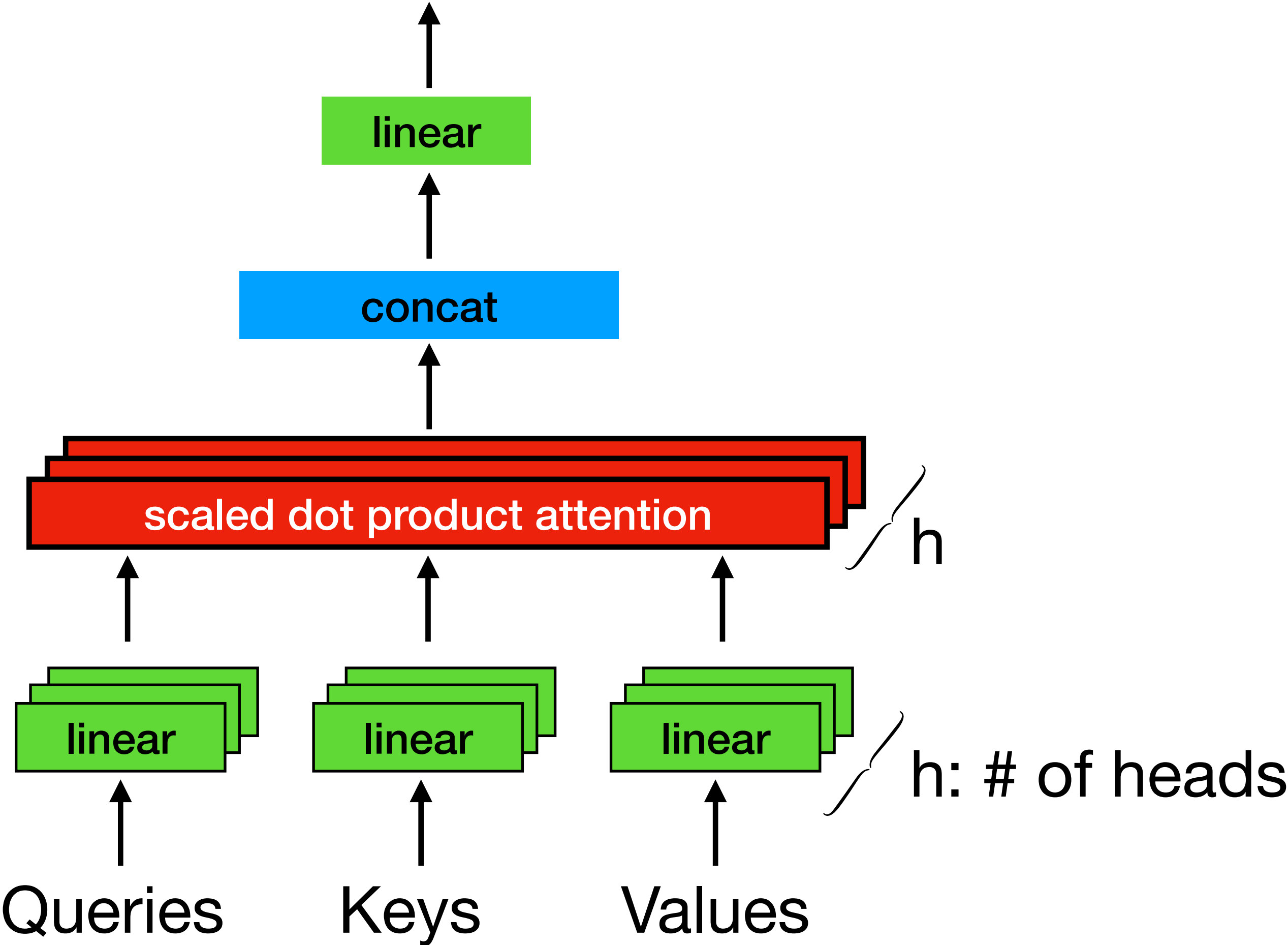
schema of a MH Attention layer

$\text{MultiHead}(Q, K, V) = \text{concat}[h_1, h_2, \dots, h_h]W_0$

$\text{concat}[h_1, h_2, \dots, h_h]$

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

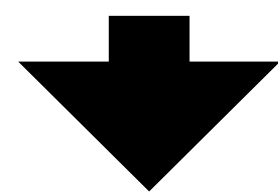
$Q_i = XW_{q,i}$
 $K_i = XW_{k,i}$
 $V_i = XW_{v,i}$



MASKED MH ATTENTION

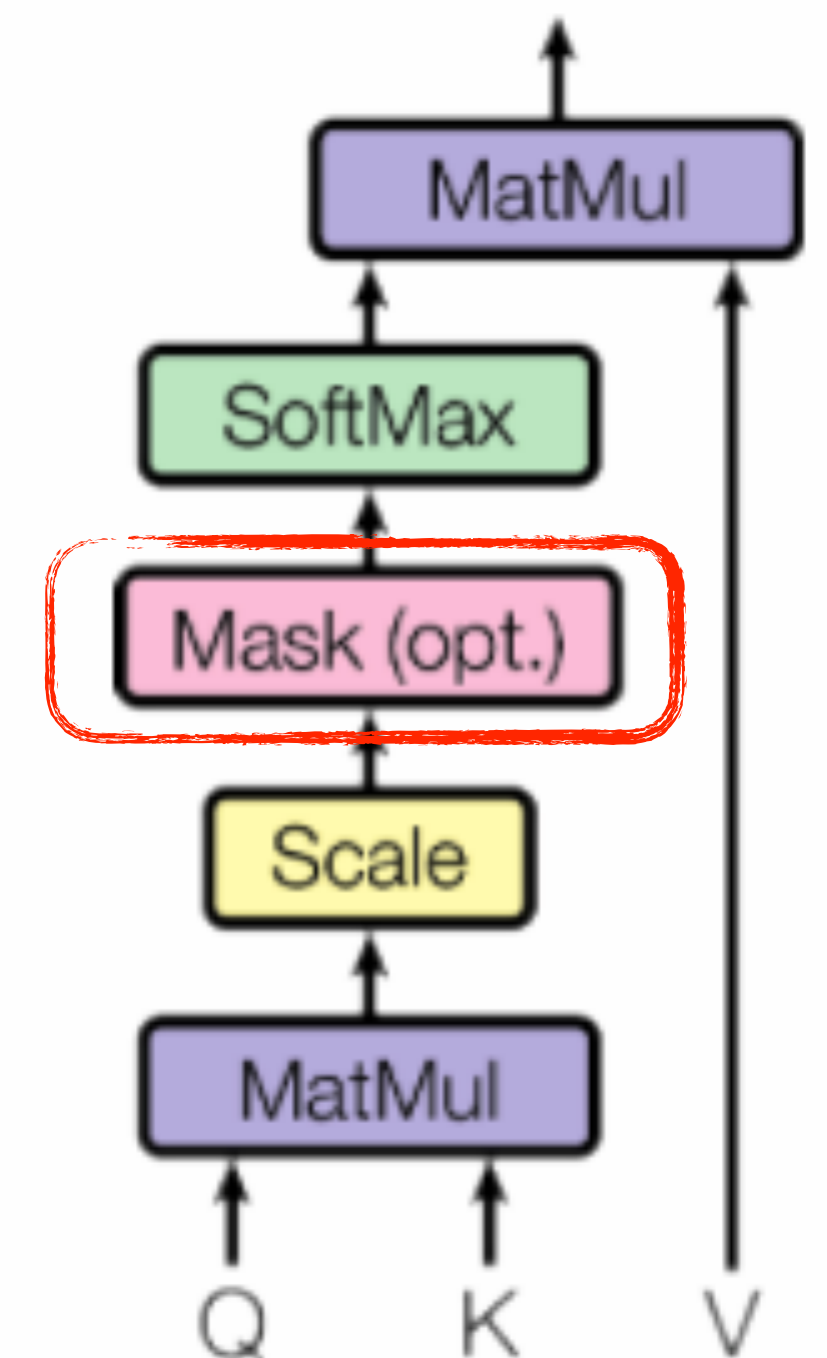
- is a masked version of the MHA layer in which **some values are masked to prevent them to be selected**
- in the decoder the first MHA combines output words with **previous** output words (a given output cannot depends on future outputs), so future outputs will be masked

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



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

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



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 faster
- 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:

$$\mu = \frac{1}{H} \sum_{i=1}^H h_i$$

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

- 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 is done for each units normalising across batch elements, so it is no sensitive to small batch sizes

POSITIONAL EMBEDDING

- is used in both encoder and decoder modules just right after the input
- allows the words in the sentence not to be treated as a bag of words, e.g. takes into account the position in the sentence of each word
- this is needed as the attention mechanism is equivariant to the ordering of the elements (e.g. $MH(PX) = P MH(X)$ with P permutation matrix) and this is not what we want for an input that is a sequence in which order is important (PE can be omitted in case we want to retain permutation equivariance in the transformer model)
- implemented with a trick:
 - use a vector that embed the position and add or concatenate this to the word embedding vector: $MH(Pconc(X,E)) \neq P MH(conc(X,E))$
 - empirically using as positional embedding vector a position integer or a one-hot encoding of the position has been shown to not perform always well, instead very good performances have been obtained using a **sinusoidal embedding**:

word position in the input embedding

for each position (scalar) a vector is produced E:

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

i: from 0 to d-1

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

d = dimension of the embedding vector

and added to the embedding ($X \rightarrow X+E$) instead of concatenating it to reduce the number of parameters (this is debatable)

PERFORMANCES

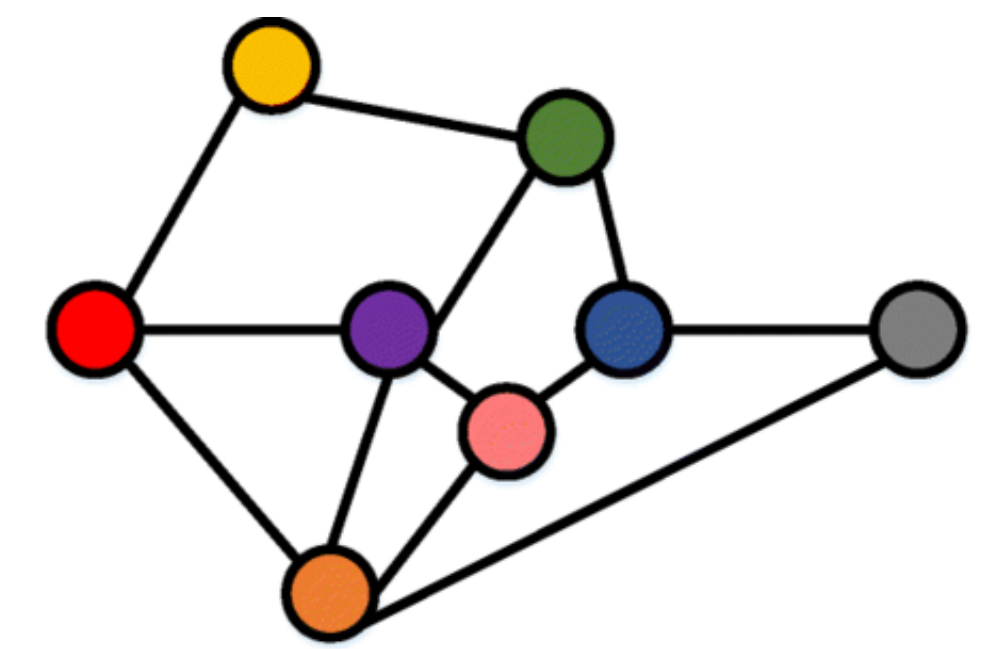
- original transformer

BLEU score: percentage of translated words that appear in the ground truth

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	largest of the two EN-DE/EN-FR
Transformer (big)	28.4	41.8	$2.3 \cdot 10^{19}$	

65 Mpar
213 Mpar

TRANSFORMERS AND GNN



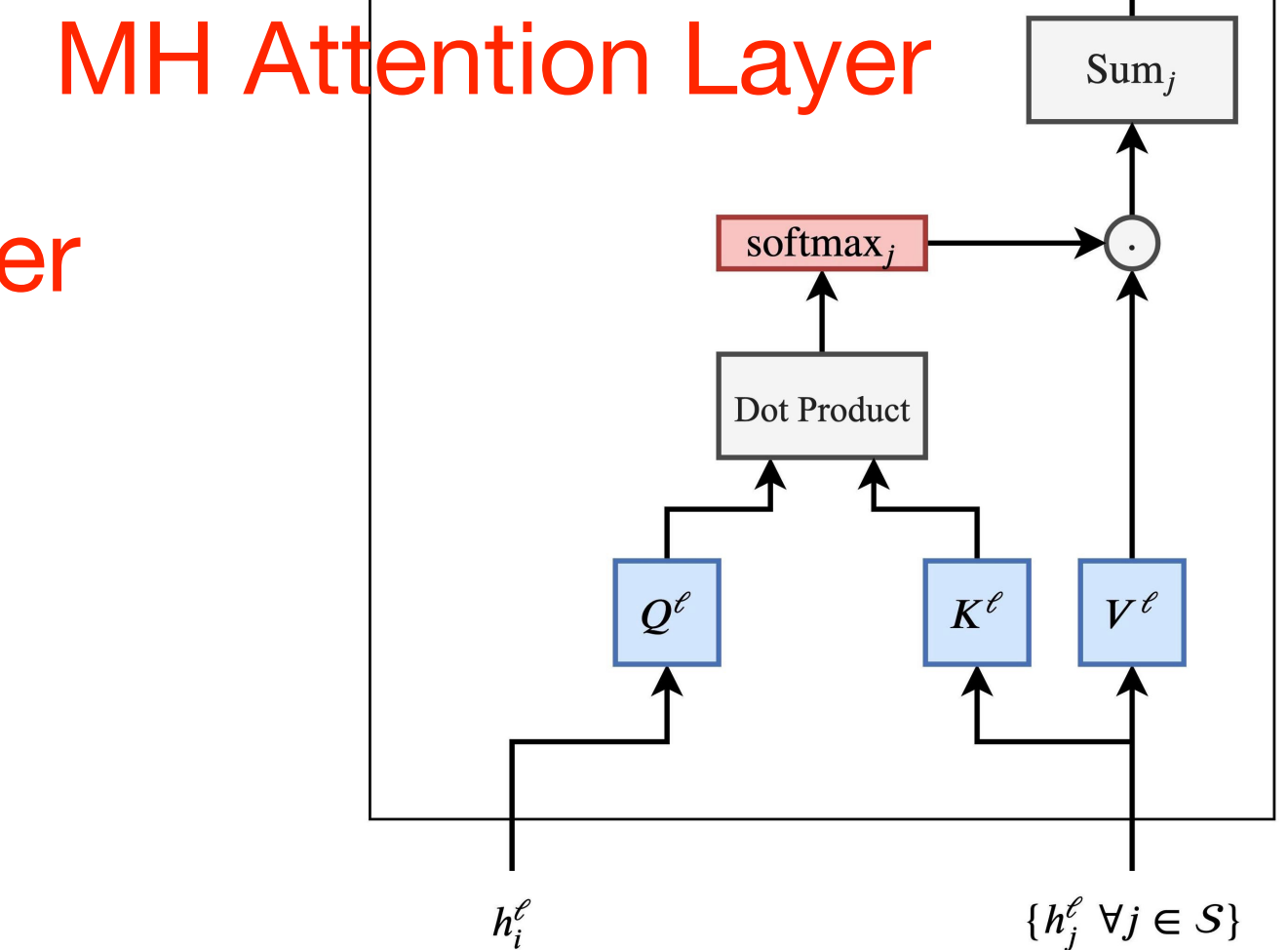
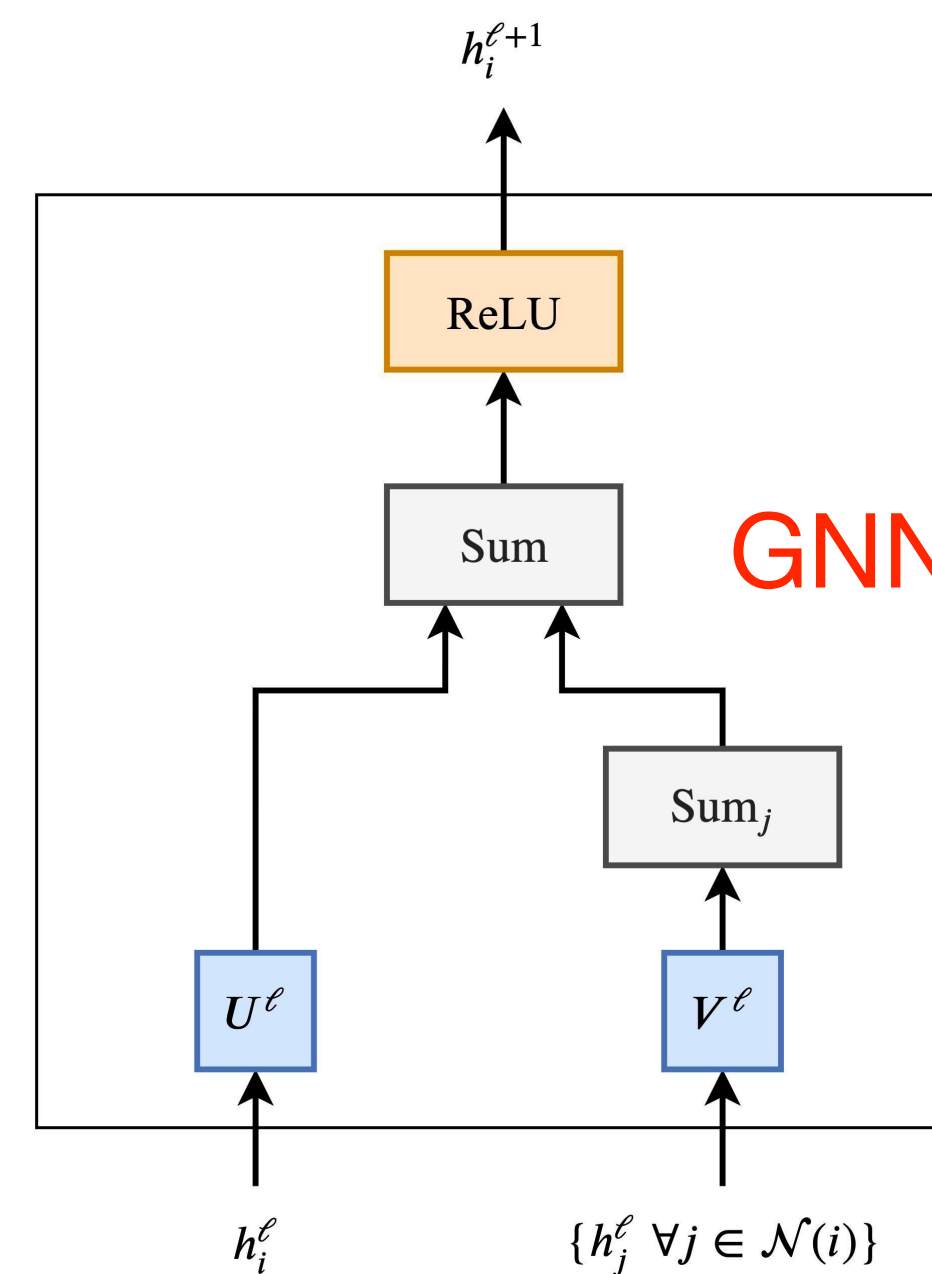
- there is a strong link between Graph Neural Networks and Transformers
- in the most simple form a GNN update the hidden feature of a given node, by message passing, i.e. by a non linear transformation of the node feature added to an aggregation of the features of the neighbouring nodes:

$$j \in N(i) \quad h_i^{t+1} = \phi(U^t h_i^t + \sum_{j \in N(i)} (V^t h_j^t))$$

non linear function (i.e. ReLU, σ , ...)

learnable weight matrices

- the sum over the neighbours node can be replaced by a permutation invariant aggregation function (ex. mean, max, ...), or with more powerful aggregators, like an **attention mechanism**

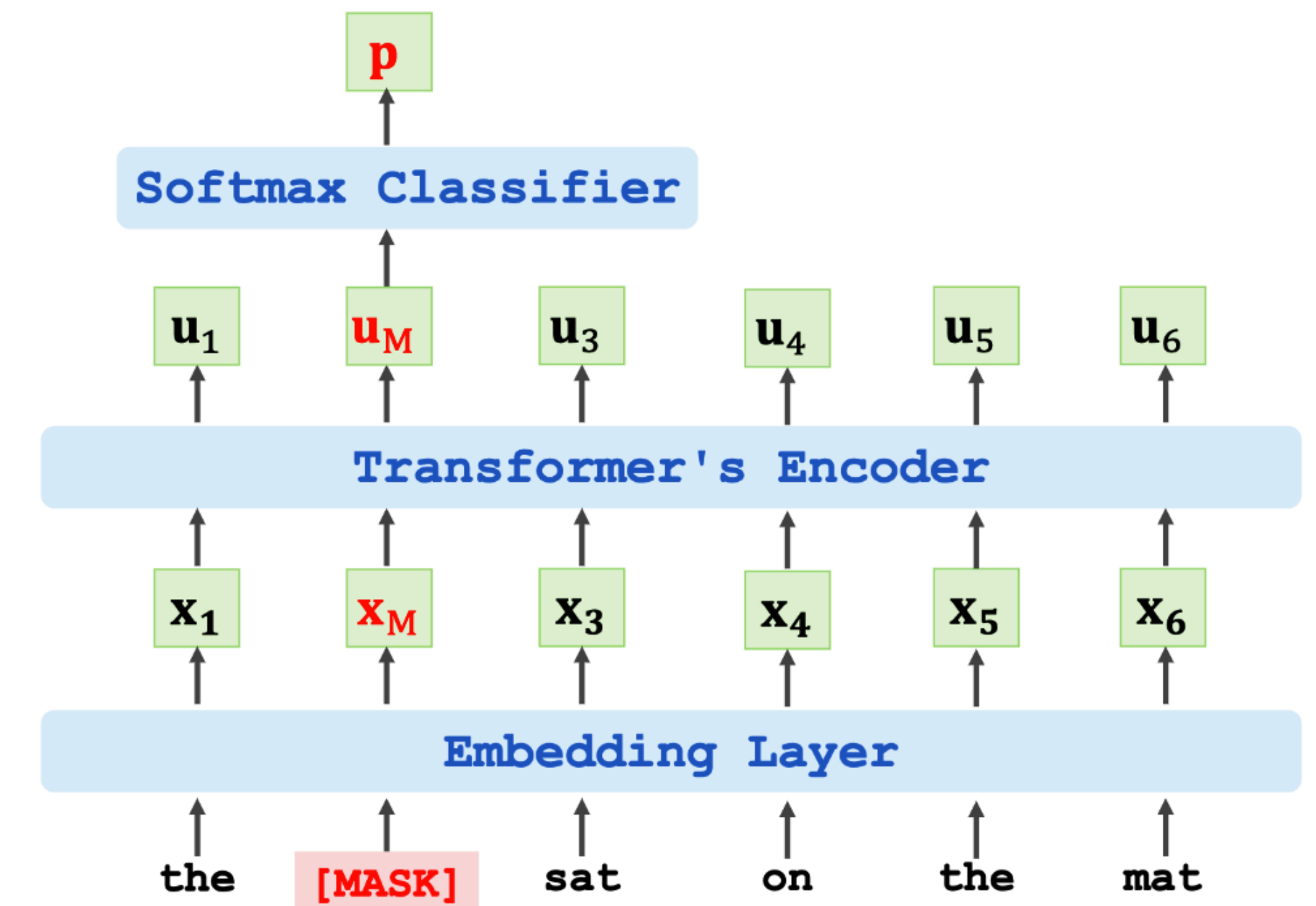


- replacing the summation over the neighbours j with the attention mechanism, i.e., with a weighted sum, we'd get the Graph Attention Network (GAT), adding normalisation and an MLP we get something formally equivalent to a graph transformer
- a Transformer is a GNN with a multi-head attention as aggregation function
- while a GNN aggregate features from their local neighbourhood nodes $j \in N(i)$, transformers treat the whole input sequence as the local neighbourhood, aggregating features from each element of the sequence at each layer

TRANSFORMERS EVOLUTIONS

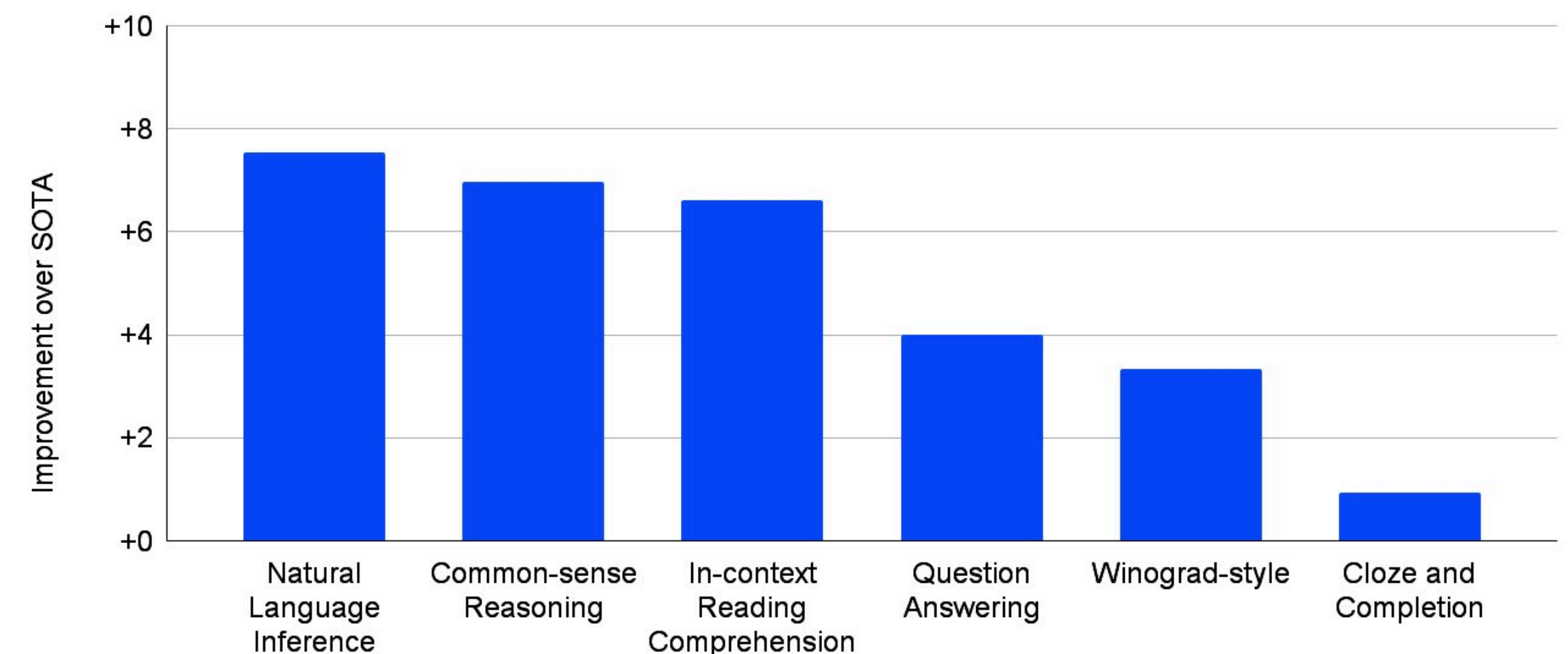
- the original transformer has spawn series of evolutions that today dominate NLP
- GPT (2018)/GPT2/GPT3 (GPT3: 175B paramaters): unsupervised multitask learned based on a decoder transformer (no encoder) that predicts next words based on previous words by computing $P(x_t|x_1, \dots, x_{t-1})$
- BERT (2019), 340M parameters: unsupervised bi-directional encoder transformer that predicts a missing word based on surrounding words by computing $P(x_t|x_1, \dots, x_{t-1}, x_{t+1}, \dots, x_N)$ or the next sentence based on a previous sentence

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



Google - PaLM (Pathways Language Model): [arXiv:2204.02311](https://arxiv.org/abs/2204.02311)

- 540B parameters
- single model that can generalize across different domains and tasks
- highly efficient training across multiple TPUs pods
- tested on hundreds of language understanding and generation tasks: achieves state-of-the-art few-shot performance across most tasks, by significant margins in many cases



Prompt

```
// Translate from C to Python
int add_one ( int x ){
    int m = 1;
    while ( x & m ) {
        x = x ^ m;
        m <<= 1;
    }
    x = x ^ m;
    return x; }
```

Model Response

```
def add_one(x: int):
    m = 1
    while (x & m):
        x = (x ^ m)
        m <<= 1
    x = (x ^ m)
    return x
```

Prompt

```
def exercise5():
    """Marie ordered one chicken meal that
    costs $12, 5 packs of milk that cost $3
    each, 4 apples that cost $1.50 each, and
    some boxes of pizza. Marie paid a total of
    $50. How many boxes of pizza did Marie
    order if box costs $8.50?"""
```

Model Response

```
total_paid = 50
cost_of_pizza_per_box = 8.50
cost_of_meal = 12
cost_of_milk_per_pack = 3
cost_of_apples_per_apple = 1.50
packages_of_milk = 5

return float((total_paid - cost_of_meal -
cost_of_milk_per_pack * packages_of_milk -
cost_of_apples_per_apple * 4)/
cost_of_pizza_per_box)
```

some PaLM examples ...

prompt

Q: Tracy used a piece of wire 4 feet long to support tomato plants in the garden. The wire was cut into pieces 6 inches long. How many pieces did she obtain?
A:

62b model

She cut the wire into 6 inch pieces. This means she got $4 * 6 = 24$ pieces. The answer is 24. ✗

540b model

The wire was 4 feet long. This means it was $4 * 12 = 48$ inches long. It was cut into pieces 6 inches long. This means she obtained $48 / 6 = 8$ pieces. The answer is 8. ✓

prompt

Q: Tom's ship can travel at 10 miles per hour. He is sailing from 1 to 4 PM. He then travels back at a rate of 6 mph. How long does it take him to get back?
A:

62b model

He travels from 1 to 4 PM. That means he travels for 3 hours. He travels back at 6 mph. That means he travels for $3 * 6 = 18$ miles. The answer is 18. ✗

540b model

He travels at 10 mph from 1 to 4 PM. This means he travels 3 hours. 3 hours at 10 mph means he travels $3 * 10 = 30$ miles. He then travels back at 6 mph. This means he travels 6 miles per hour. He has to travel 30 miles, so it takes him $30 / 6 = 5$ hours. The answer is 5. ✓

Explaining a joke

Prompt

Explain this joke:

Joke: Did you see that Google just hired an eloquent whale for their TPU team? It showed them how to communicate between two different pods!

Model Response

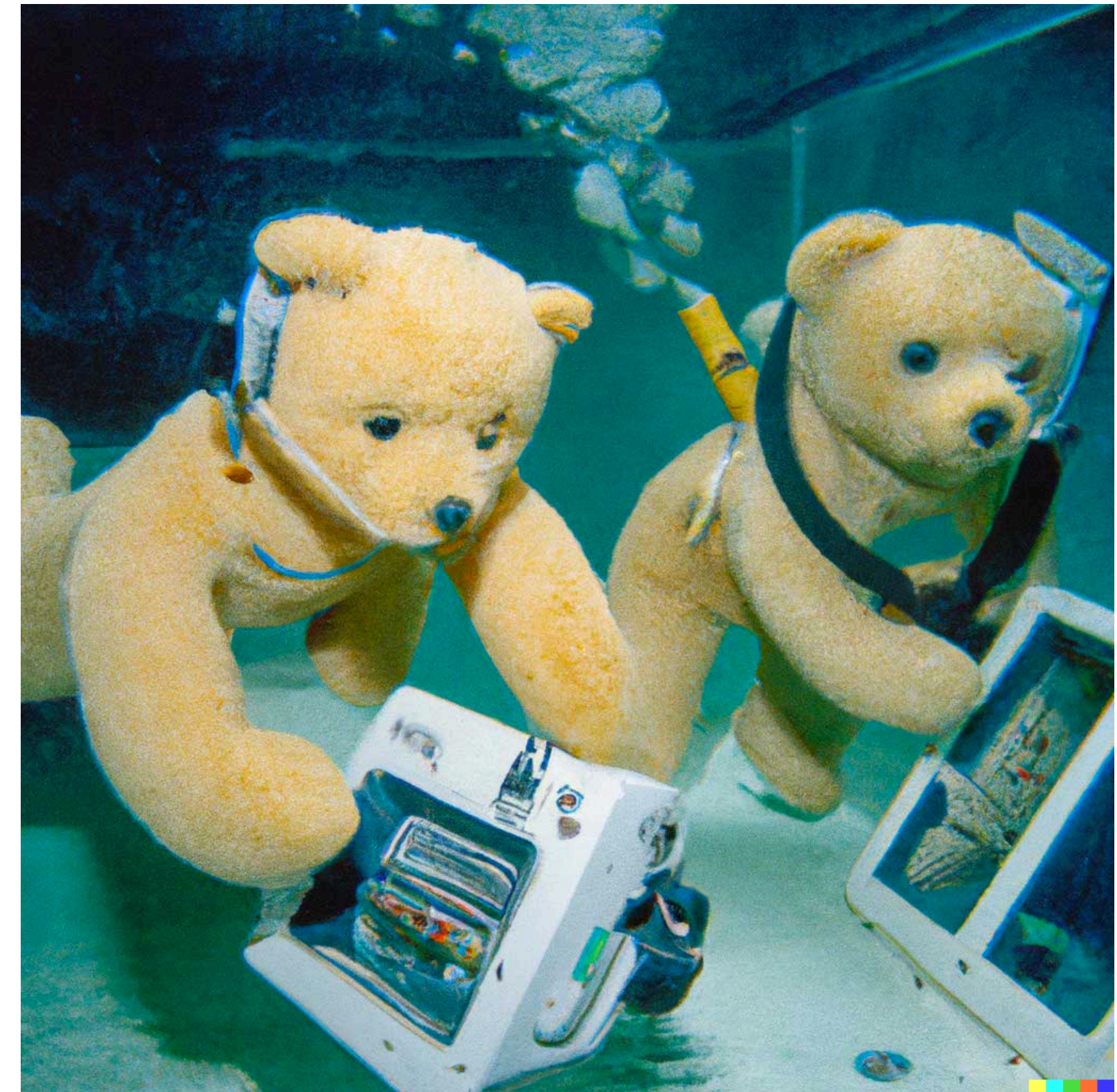
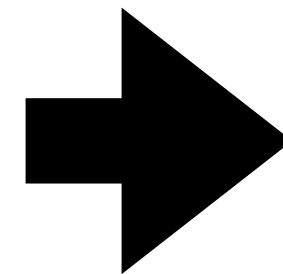
Prediction: TPUs are a type of computer chip that Google uses for deep learning. A "pod" is a group of TPUs. A "pod" is also a group of whales. The joke is that the whale is able to communicate between two groups of whales, but the speaker is pretending that the whale is able to communicate between two groups of TPUs.

OpenAI - DALL·E 2: <https://openai.com/dall-e-2/>

- scaled down (3.5B parameters) evolution of the first version DALL·E (12B pars) released in 2021 which in turn is a scaled down evolution from GPT3 (175B pars)
- generative transformer (combines transformers with diffusion models) able to create new realistic images and art from based on a text description. It can combine concepts, attributes, and styles

TEXT DESCRIPTION

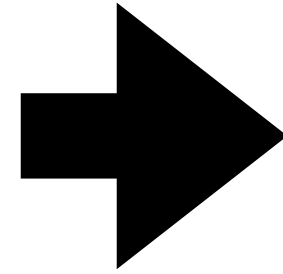
An astronaut Teddy bears A bowl of soup



TEXT DESCRIPTION

An astronaut Teddy bears A bowl of
soup

riding a horse lounging in a tropical
resort in space playing basketball
with cats in space



“A rabbit detective sitting on a park bench and
reading a newspaper in a victorian setting”



“macro 35mm film photography of a large family of
mice wearing hats cozy by the fireplace”



Meta - Galactica: <http://galactica.org/>

- transformer model (125B parameters) trained on a large corpus comprising more than 360 millions in-context citations and over 50 millions of unique references normalized across a diverse set of sources. Enables to suggest citations and help discover related papers
- on technical knowledge probes such as LaTeX equations, outperforms the latest GPT-3 and PaLM 540B on MATH.

Input:

Question: Translate the following Math formula:

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s}$$

into plain English.

Answer:

GALACTICA Output:

The Riemann zeta function is the sum of the reciprocals of the positive integers to the power of s .

Answer: The Riemann zeta function is the sum of the reciprocals of the positive integers to the power of s .

Input:

A paper that suggested to initialize model weights of linear layers with n inputs and m outputs using the following formula:

```
numpy.random.normal(0, numpy.sqrt(2 / n))
```

GALACTICA Suggestions:

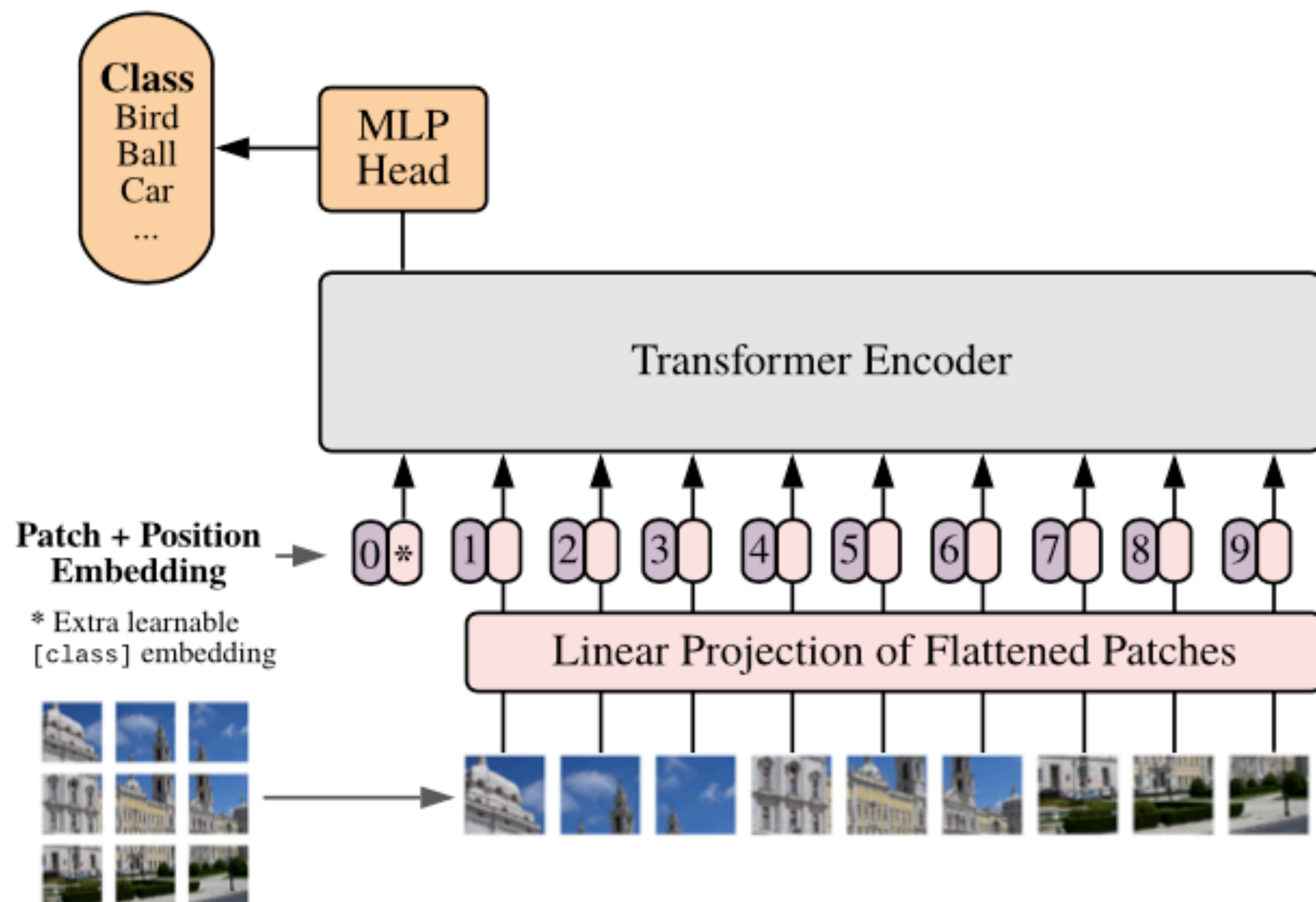
Delving Deep into Rectifiers: Surpassing Human-Level Performance on {ImageNet} Classification

He et al., 2015

VISION TRANSFORMERS

- the very same philosophy of the Transformer architecture can be applied to vision, signal analysis, point-cloud analysis, etc. tasks
- Vision Transformer (ViT) has been proposed in 2021 by A. Dosovitsky et al. in [arXiv:2010.11929](https://arxiv.org/abs/2010.11929)
 - is based on the same original Transformer architecture of Vaswani et al.
 - has shown to be able to surpass SOTA CNN architectures ResNet, but only if the dataset needed to re-train the model is large enough (large enough means $> 100\text{M}$ images)
- 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

Vision Transformer (ViT)



Transformer Encoder

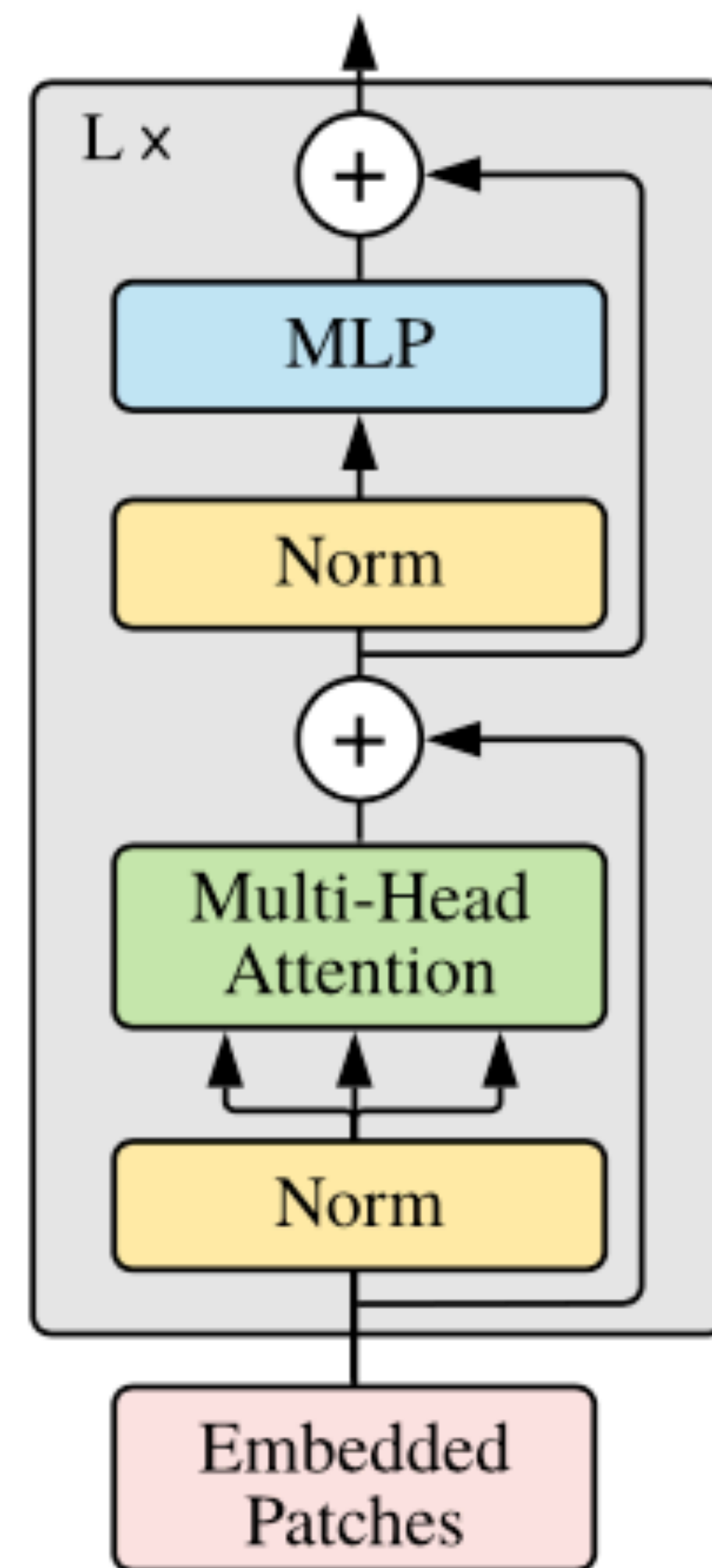
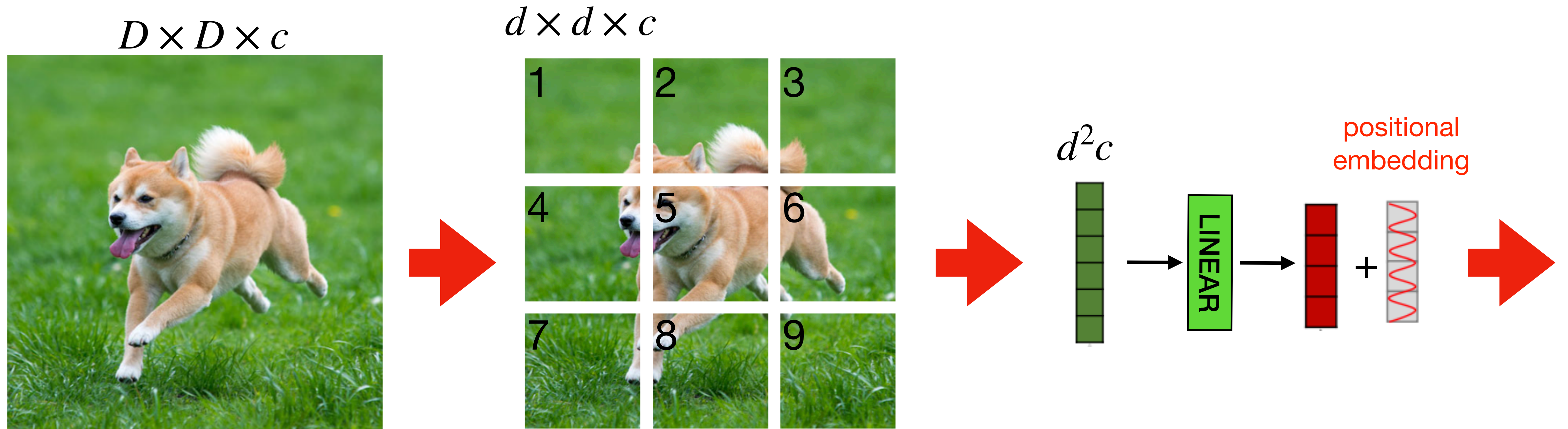


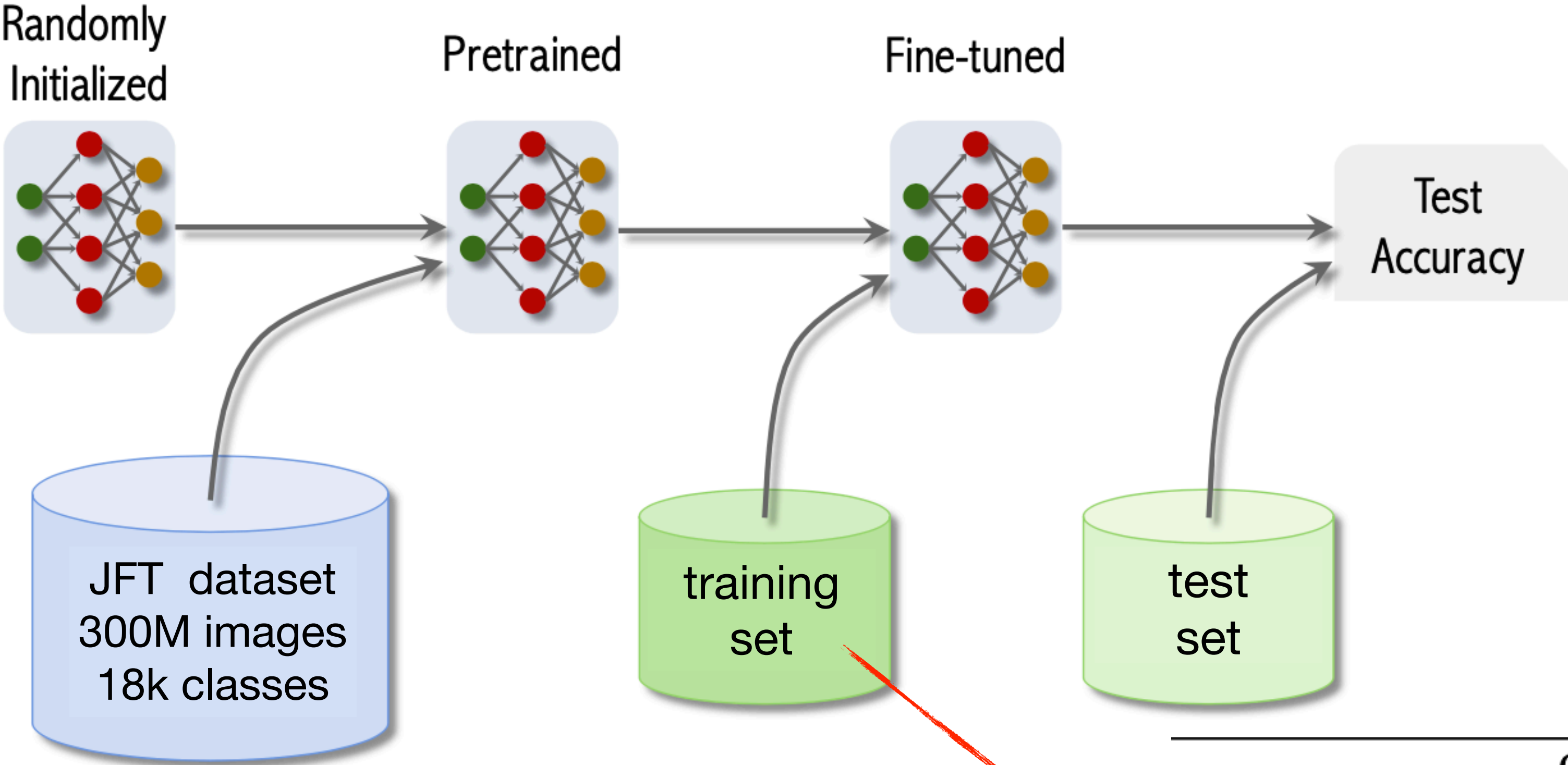
IMAGE PATCHING AND VECTORISATION

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



- NOTE: 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 two-dimensional 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

TRAINING AND PERFORMANCE



pertained on
JFT

pertained on
imagenet-21k
14M images
21k classes

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)
ImageNet	88.55 ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02
ImageNet ReaL	90.72 ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54
CIFAR-10	99.50 ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06
CIFAR-100	94.55 ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08
Oxford-IIIT Pets	97.56 ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23
Oxford Flowers-102	99.68 ± 0.02	99.74 ± 0.00	99.61 ± 0.02	99.63 ± 0.03
VTAB (19 tasks)	77.63 ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k

AN EXAMPLE OF PYTORCH IMPLEMENTATION OF VIT

- <https://github.com/lucidrains/vit-pytorch>

```
class Attention(nn.Module):
    def __init__(self, dim, heads = 8, dim_head = 64, dropout = 0.):
        super().__init__()
        inner_dim = dim_head *  heads
        project_out = not (heads == 1 and dim_head == dim)

        self.heads = heads
        self.scale = dim_head ** -0.5

        self.attend = nn.Softmax(dim = -1)
        self.dropout = nn.Dropout(dropout)

        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias = False)

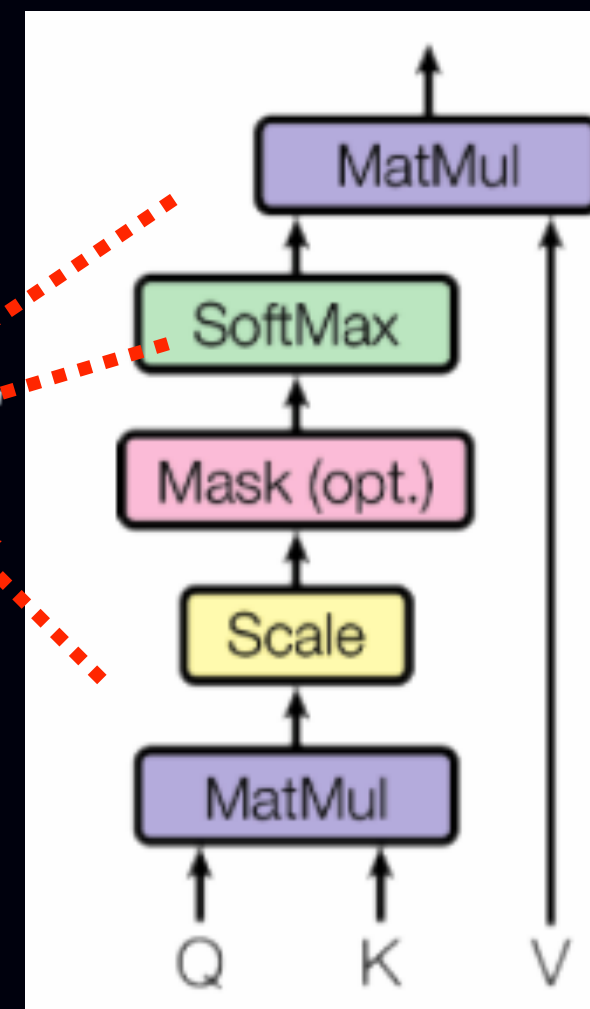
        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, dim),
            nn.Dropout(dropout)
        ) if project_out else nn.Identity()

    def forward(self, x):
        qkv = self.to_qkv(x).chunk(3, dim = -1)
        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', t), qkv)

        dots = torch.matmul(q, k.transpose(-1, -2)) * self.scale

        attn = self.attend(dots)
        attn = self.dropout(attn)

        out = torch.matmul(attn, v)
        out = rearrange(out, 'b h n d -> b n (h d)')
        return self.to_out(out)
```



```
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim, dropout = 0.):
        super().__init__()
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                PreNorm(dim, Attention(dim, heads = heads, dim_head = dim_head, dropout = dropout)),
                PreNorm(dim, FeedForward(dim, mlp_dim, dropout = dropout))
            ]))

    def forward(self, x):
        for attn, ff in self.layers:
            x = attn(x) + x
            x = ff(x) + x
        return x

class ViT(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes):
        super().__init__()
        image_height, image_width = pair(image_size)
        patch_height, patch_width = pair(patch_size)

    def forward(self, img):
        x = self.to_patch_embedding(img)
        b, n, _ = x.shape

        cls_tokens = repeat(self.cls_token, '1 n d -> b n d', b = b)
        x = torch.cat((cls_tokens, x), dim=1)
        x += self.pos_embedding[:, :(n + 1)]
        x = self.dropout(x)

        x = self.transformer(x)

        x = x.mean(dim = 1) if self.pool == 'mean' else x[:, 0]

        x = self.to_latent(x)
        return self.mlp_head(x)
```