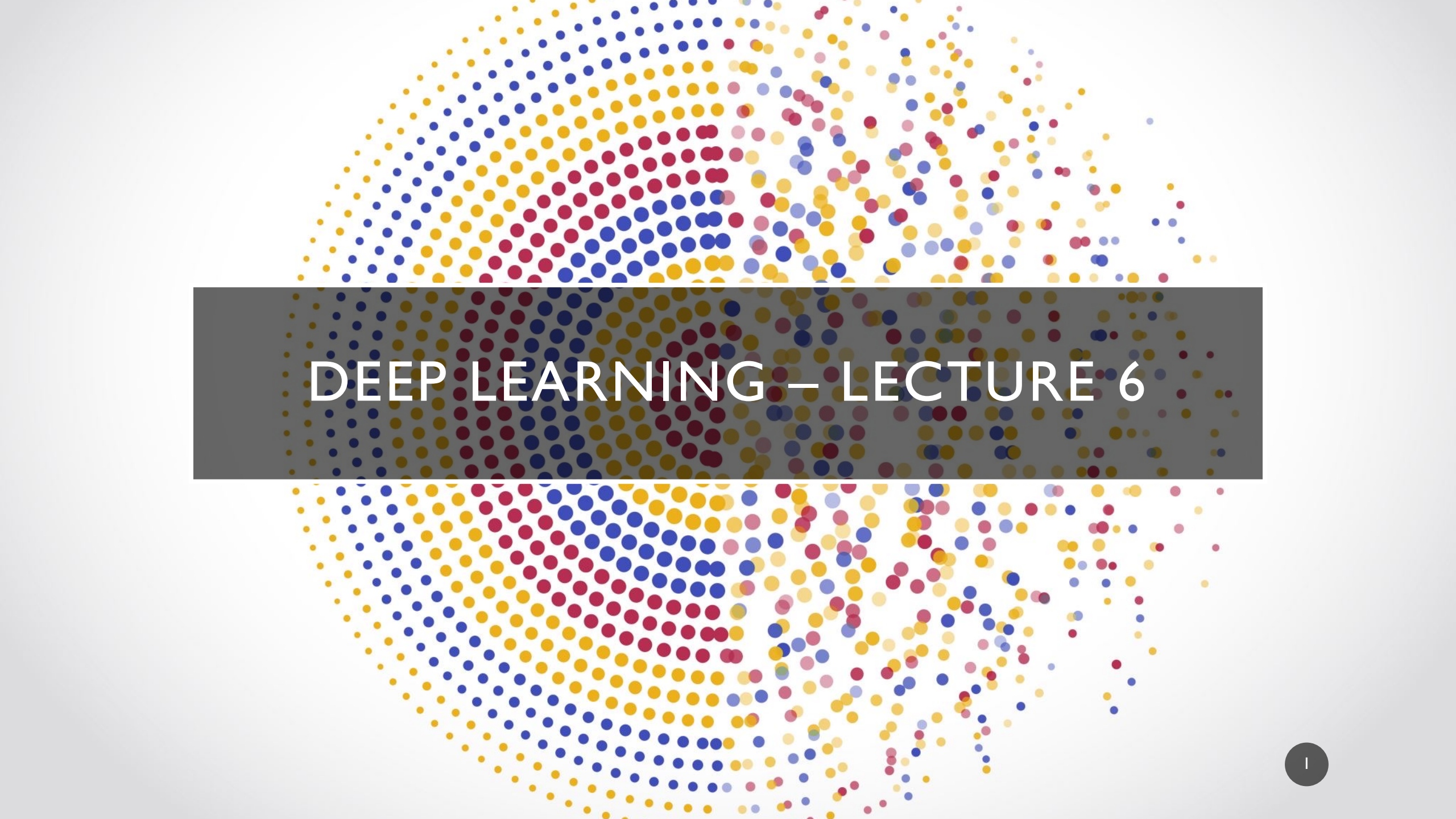
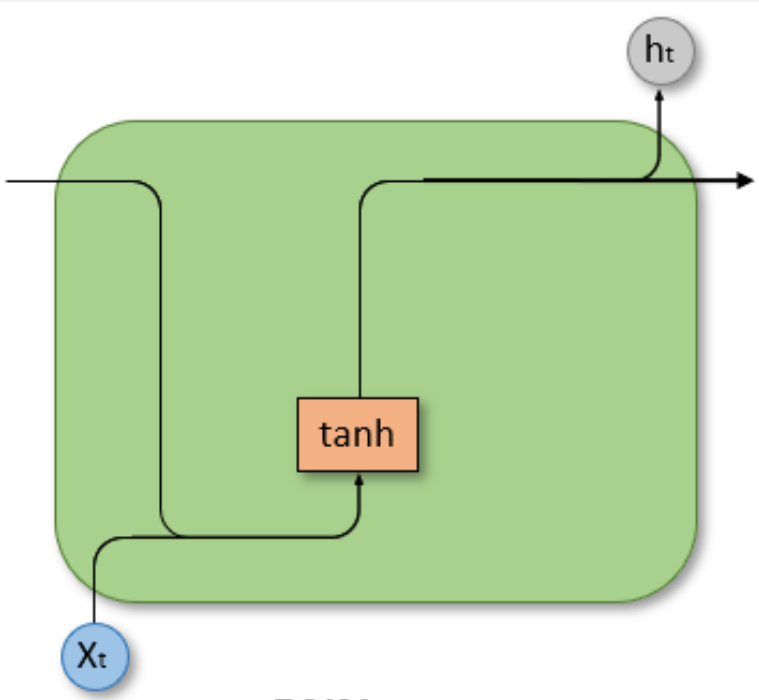
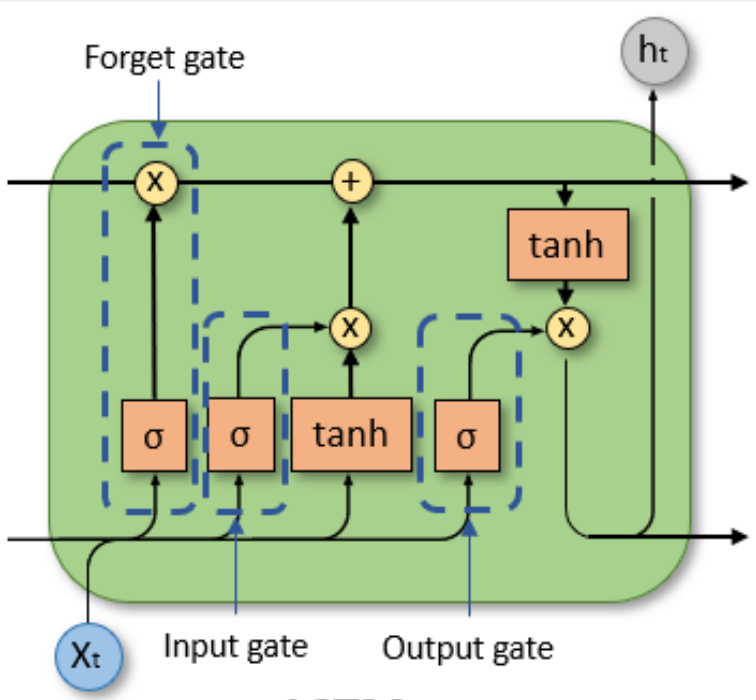


DEEP LEARNING – LECTURE 6

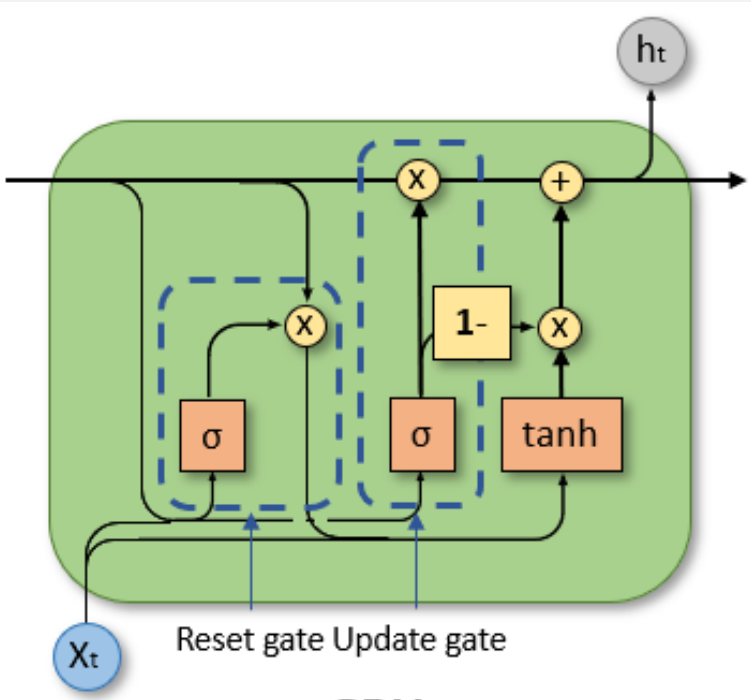
RECAP



RNN



LSTM



GRU

TODAY'S AGENDA

- Seq2Seq with CNNs
- Attention
- Self Attention
- Transformers
- AlphaFold2
- Vision Transformers

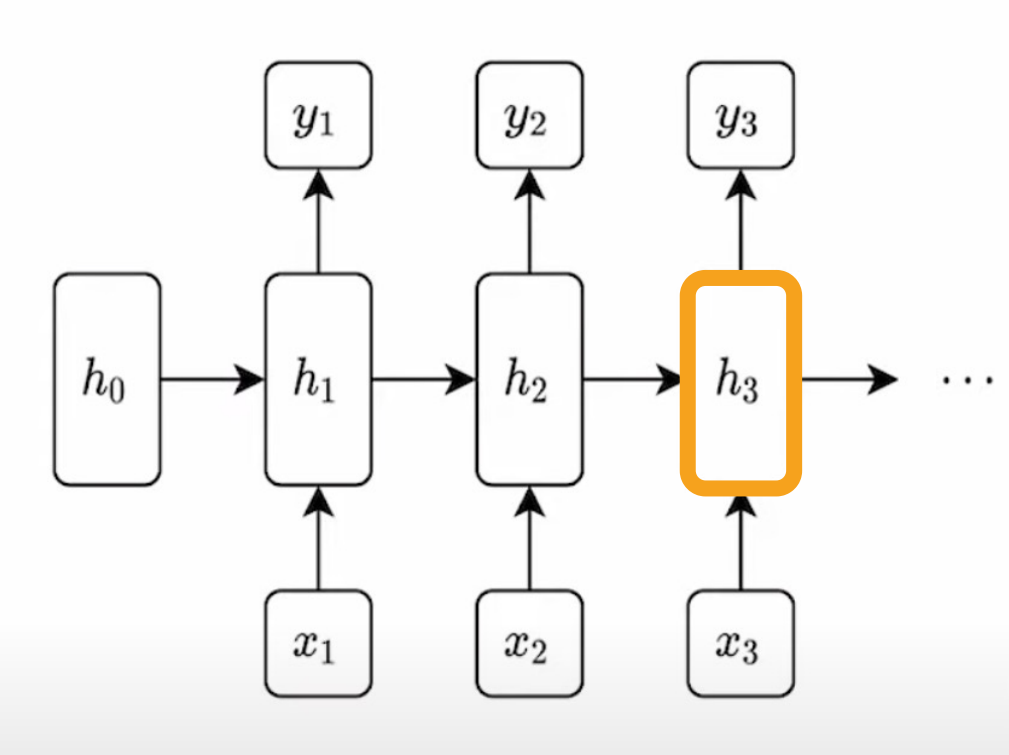
Today's lecture is based on: https://www.youtube.com/watch?v=IFKRf-BAqZo

SEQ2SEQ MODELS

RNN approach to sequence-to-sequence problems: maintain a latent state h_t that summarizes all information up until t.

Pros: potentially "infinite" history saved in a compact representation.

Cons: long compute path between history and current time (vanishing/exploding gradients, hard to learn).



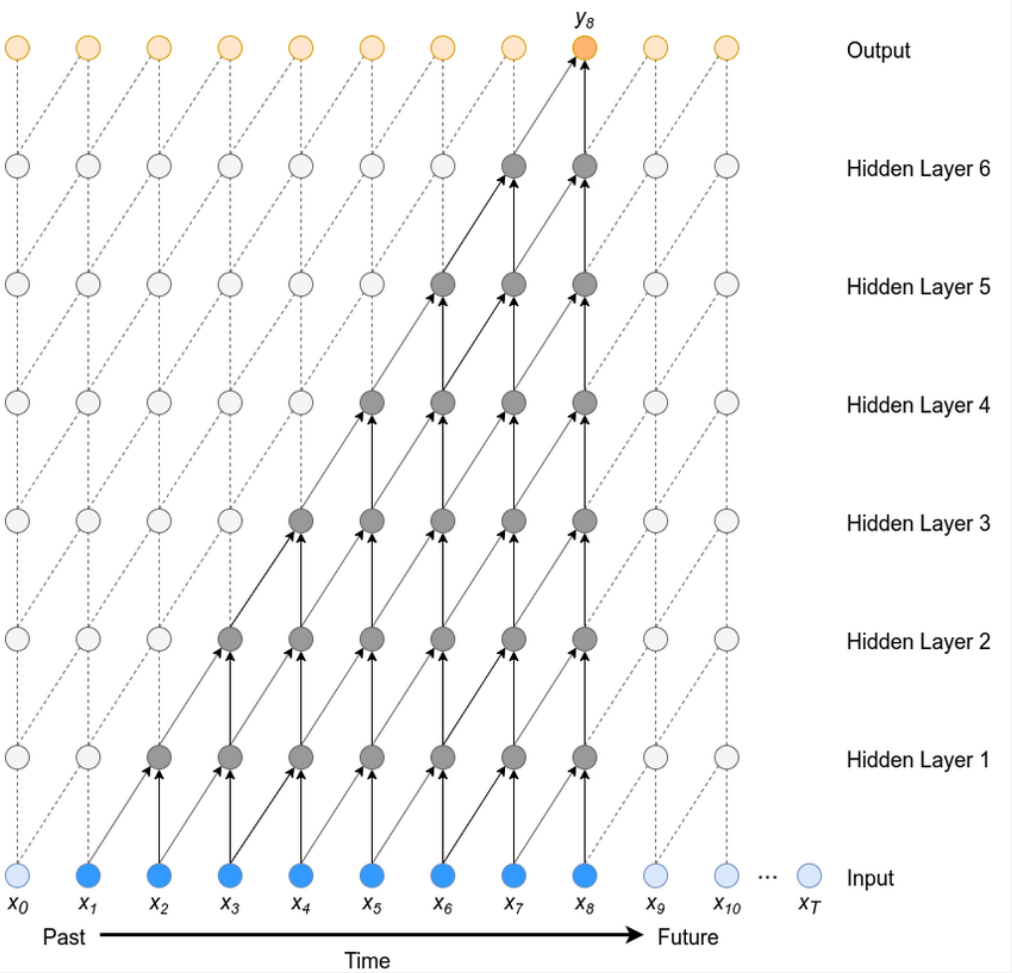
DESIRED CAPABILITIES FOR SEQUENCE MODELING

- Continuous stream of data – allow the network to process inputs as they come and not be restricted to process everything before adding another point.
- Parallelization – we would want to make parallel computations that are simply impossible with RNNs.
- Long memory – we want to refer to the entire history without forgetting

CNNS FOR SEQ2SEQ PREDICTION

Causal convolutions are a type of convolution used for temporal data, which ensures that the model does not violate the order of data.

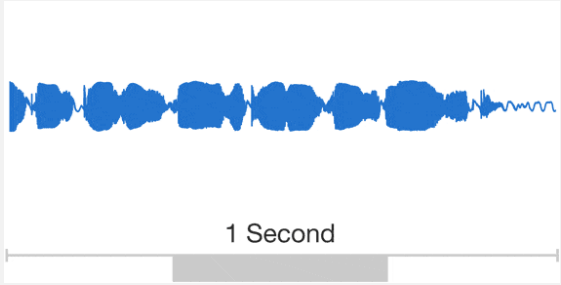
Causal convolutions operate by restricting the accessible input at each time-step, using only the data up to and including the current timestep.



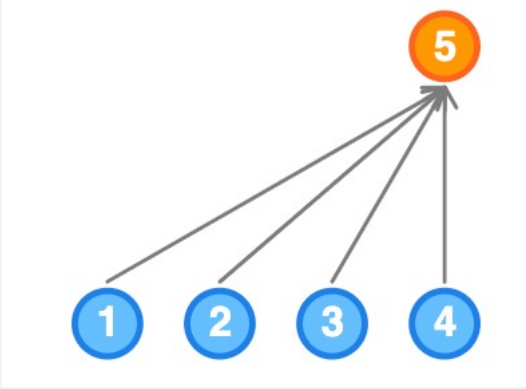
https://www.kilians.net/post/convolution-in-autoregressive-neural-networks/

CAUSAL CNNS – HOW IT WORKS

- Assume we have sequential data, for example, recording of human speech.



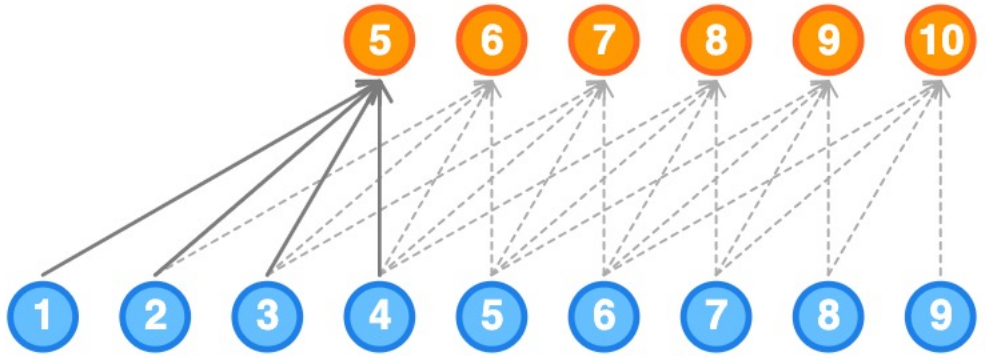
- Based on the recording we have; we can compute a probabilistic model of the value of the next time step given the values at previous time steps.
- A simple approach would be to model the next value using an affine transformation (linear combination + bias) of the four previous values.



CAUSAL CNNS – HOW IT WORKS CONT.

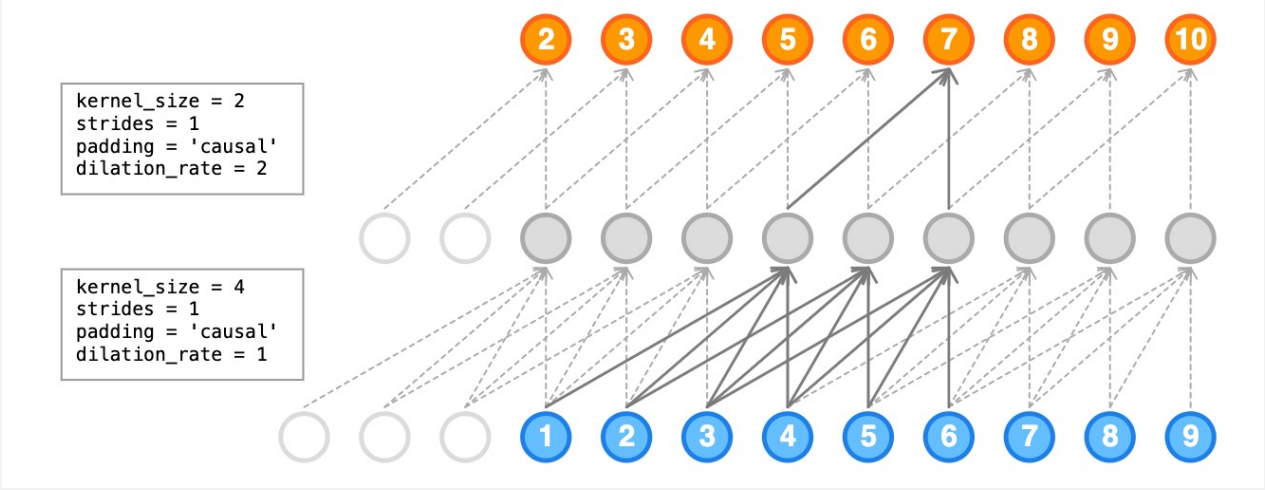
- Since we only predict one time step per feed-forward pass, it takes long to iterate through a single training example, let alone through our entire dataset.
- It would speed up training if we could predict the next time steps for a whole training example in a single feed-forward pass.
- To do this, we need to apply the same four weights to each chunk of time steps in the input frame.

```
kernel_size = 4
strides = 1
padding = 'valid'
```



CHALLENGES WITH CAUSAL CNNS

- Despite their simplicity, CNNs have a notable disadvantage for seq2seq predictions: the **receptive field** for each convolution is relatively small.
- The receptive field size can only be as deep as the depth of the network.
- Potential solutions:
  - Increase kernel size.
  - Pooling layers.
  - Dilated convolution



ATTENTION IS ALL YOU NEED

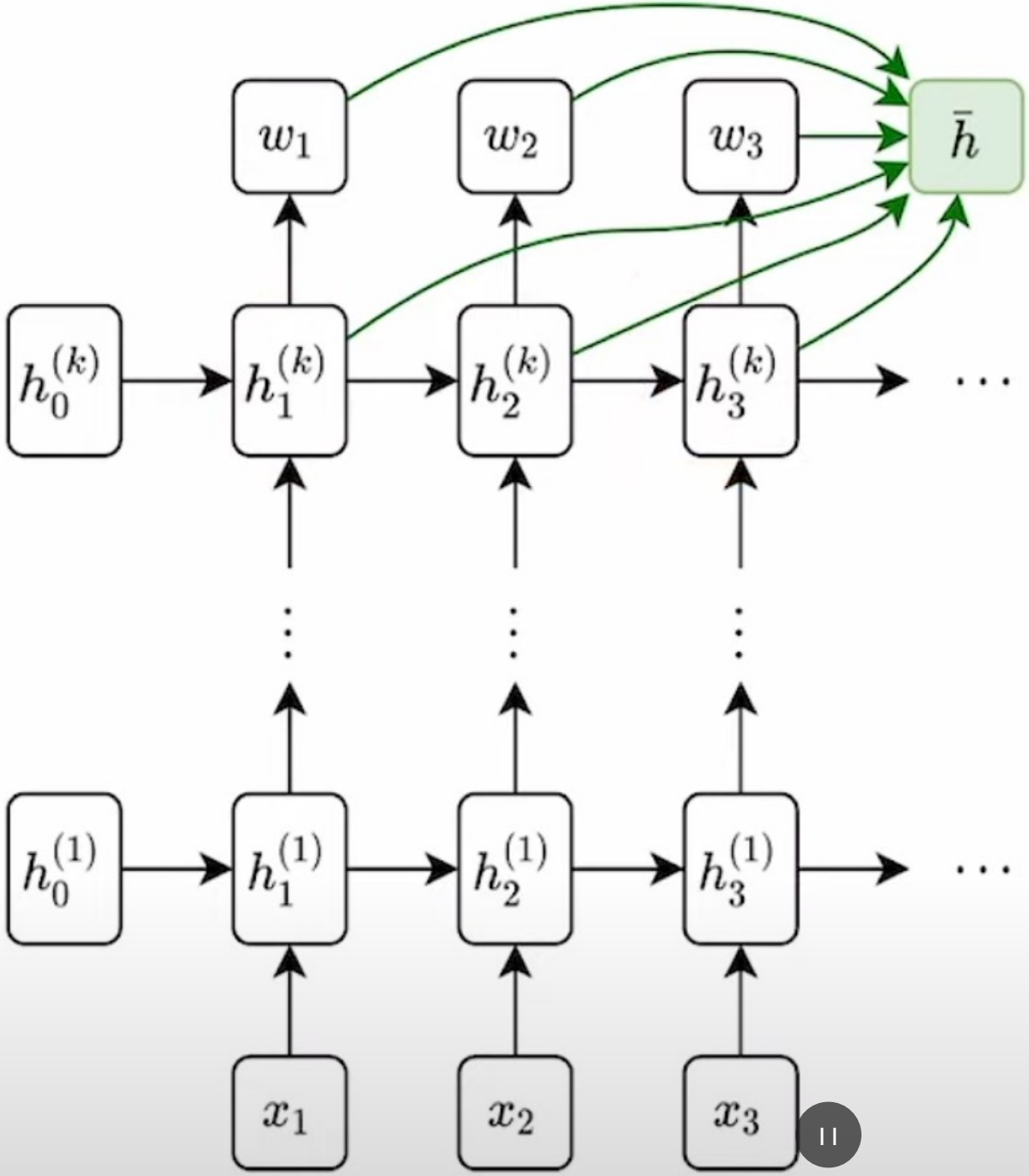
ATTENTION IN RNNS

Attention was used originally in RNNs when one wanted to combine the latent states h_t over **all times** according to some weights instead of just taking the last hidden state $h_3^{(k)}$ as a representation of history.

$$z_t = \theta^T h_t^{(k)}$$

$$w_t = softmax(z_t)$$

$$\bar{h} = \sum_{s=1}^{T} w_s h_s^{(k)}$$



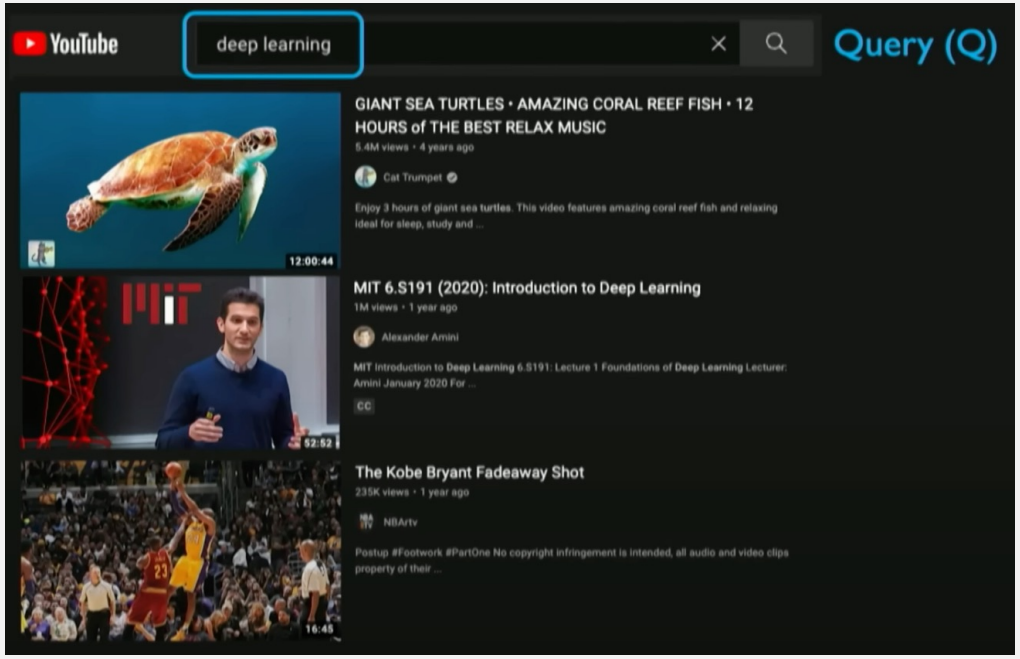
INTUITION BEHIND SELF-ATTENTION

In order to attend to the most important parts of an input, we need to do two things:

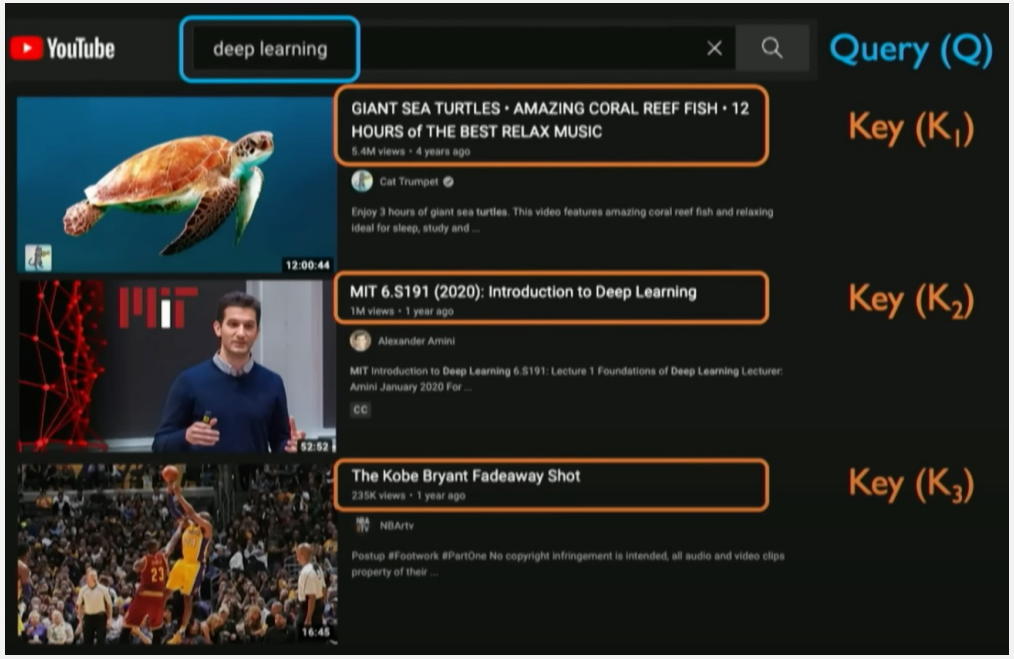
1. Identify which parts to attend to
2. Extract the features with high attention

The first part of this problem is the most difficult one, and it is very similar to the problem of search – as it involves sifting through vast amounts of information to pinpoint the pertinent and significant components.

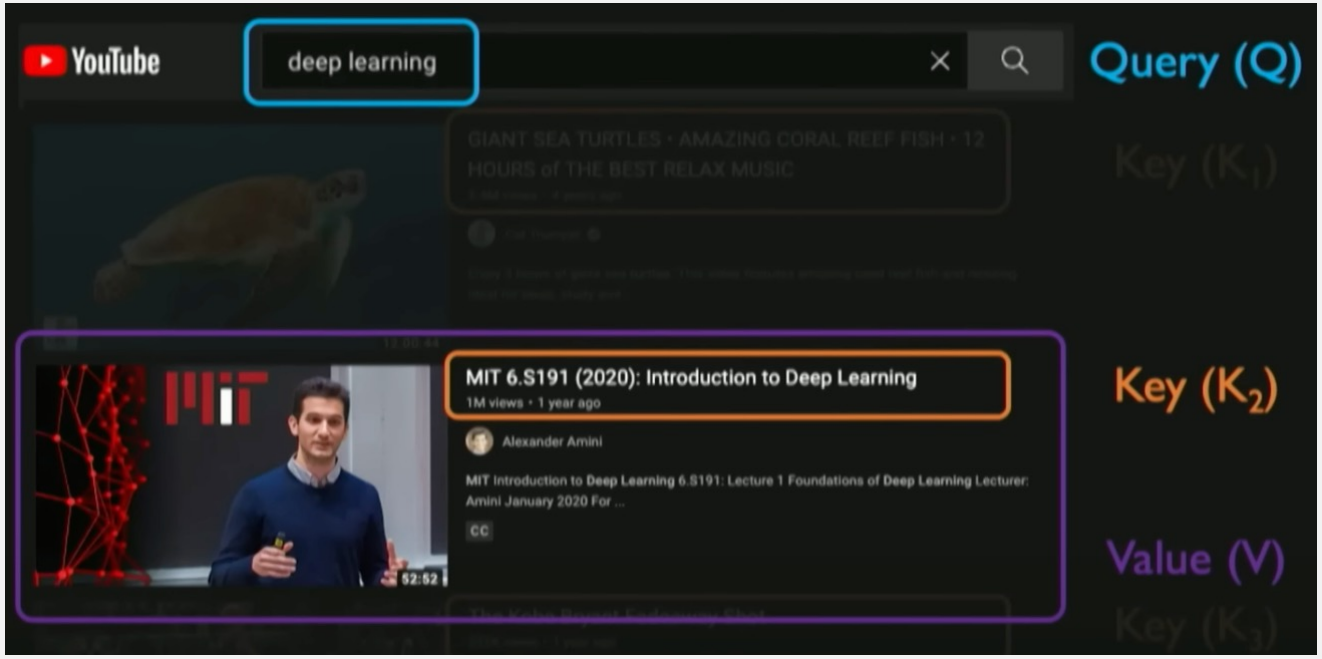
UNDERSTANDING ATTENTION WITH SEARCH



UNDERSTANDING ATTENTION WITH SEARCH



UNDERSTANDING ATTENTION WITH SEARCH



SELF-ATTENTION

- Self-attention refers to a particular form of the attention mechanism, i.e., they form a weighted combination of some inputs where the weights are themselves determined by the inputs.
- Given three matrices $K, Q, V \in \mathbb{R}^{T \times d}$, we define the self attention operation as:

$$SelfAttention(K, Q, V) = softmax\left(\frac{KQ^T}{\sqrt{d}}\right)V$$

- K, Q, V will be defined as a weighted combination of the inputs with some weight parameters of the model, for example: $K = WX$.

$$SelfAttention(K, Q, V) = softmax\left(\frac{KQ^T}{\sqrt{d}}\right)V$$

SELF ATTENTION IN DETAIL

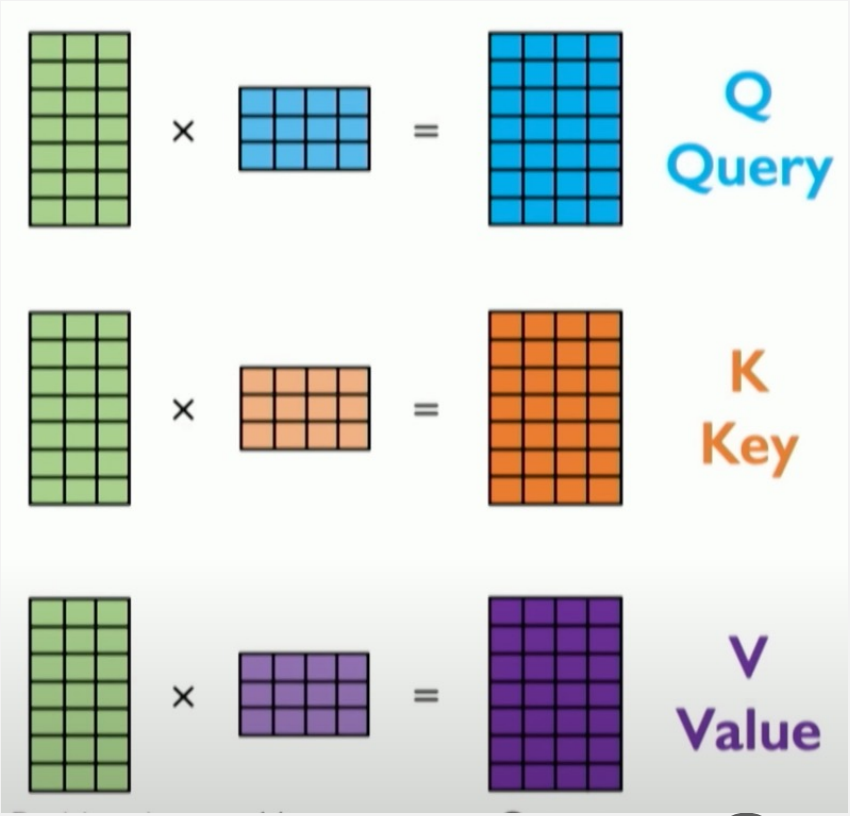
The operation of self attention can be broken down into several steps:

1. Input Transformation

The input sequence is first transformed into three matrices: $Q, K,$ and V.

These vectors are obtained by multiplying the input sequence $X = (x_1, x_2, \ldots, x_T)$ with learned weight matrices W_Q, W_K, W_V.

Each element in the input sequence x_t is associated with a row in the $Q, K,$ and V matrices.



$$SelfAttention(K,Q,V) = softmax\left(\frac{\boldsymbol{KQ^T}}{\sqrt{d}}\right)V$$

SELF ATTENTION IN DETAIL

2. Similarity Calculation: KQ^T

The next step involves calculating the similarity between each pair of elements in the input sequence.

This is done by taking the dot product of the Q row representing one element with the K row of another element.

This results in a similarity score for each pair of elements (cosine similarity is the cosine of the angles between two vectors computed by the dot product of the vectors divided by their lengths).

$$Self\,Attention(K, Q, V) = \boldsymbol{softmax}\left(\frac{\boldsymbol{KQ^T}}{\sqrt{\boldsymbol{d}}}\right)V$$

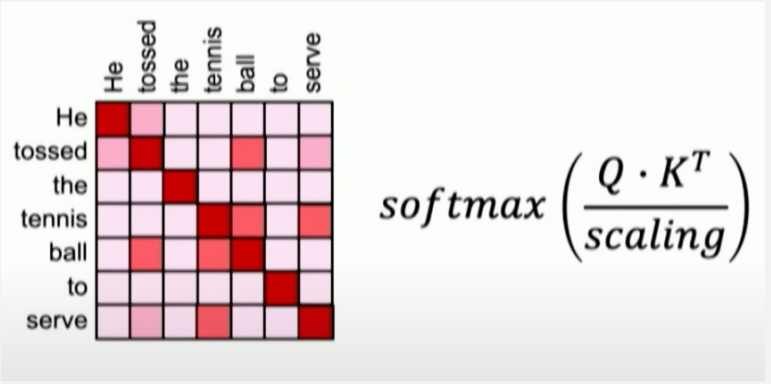
SELF ATTENTION IN DETAIL

3. Attention Weights

The similarity scores are then scaled and passed through a softmax function applied row-wise to obtain attention weights.

These weights indicate how much each element should attend to the other elements in the sequence.

Elements with higher similarity scores will have higher attention weights, meaning they are more influential in the calculation of the output.



$$SelfAttention(K, Q, V) = softmax\left(\frac{KQ^T}{\sqrt{d}}\right)\boldsymbol{V}$$

SELF ATTENTION IN DETAIL

4. Weighted Sum

The attention weights are used to calculate a weighted sum of the V vectors.

This weighted sum represents the importance of each element in the input sequence, taking into account the relationships and dependencies between different elements.

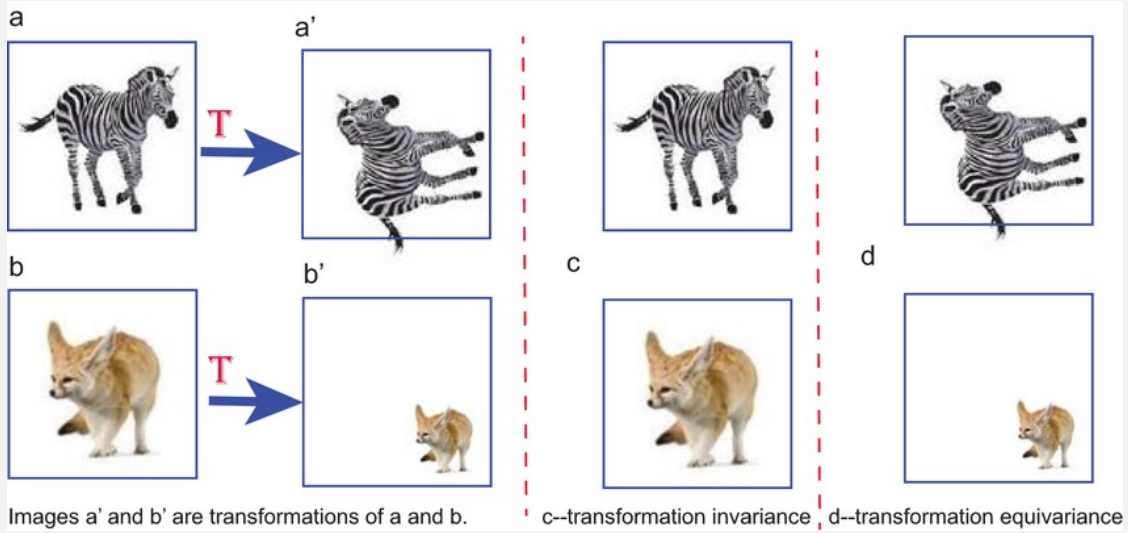
5. Output Calculation

The weighted sum is then passed through a linear transformation to produce the final output of the self-attention layer.

PROPERTIES OF SELF ATTENTION

Equivariance:

Self-attention is equivariant to permutations of the input sequence. This means that if the order of the input elements is changed, the output of the self-attention mechanism will change in the same way.



PROPERTIES OF SELF ATTENTION

Global Context:

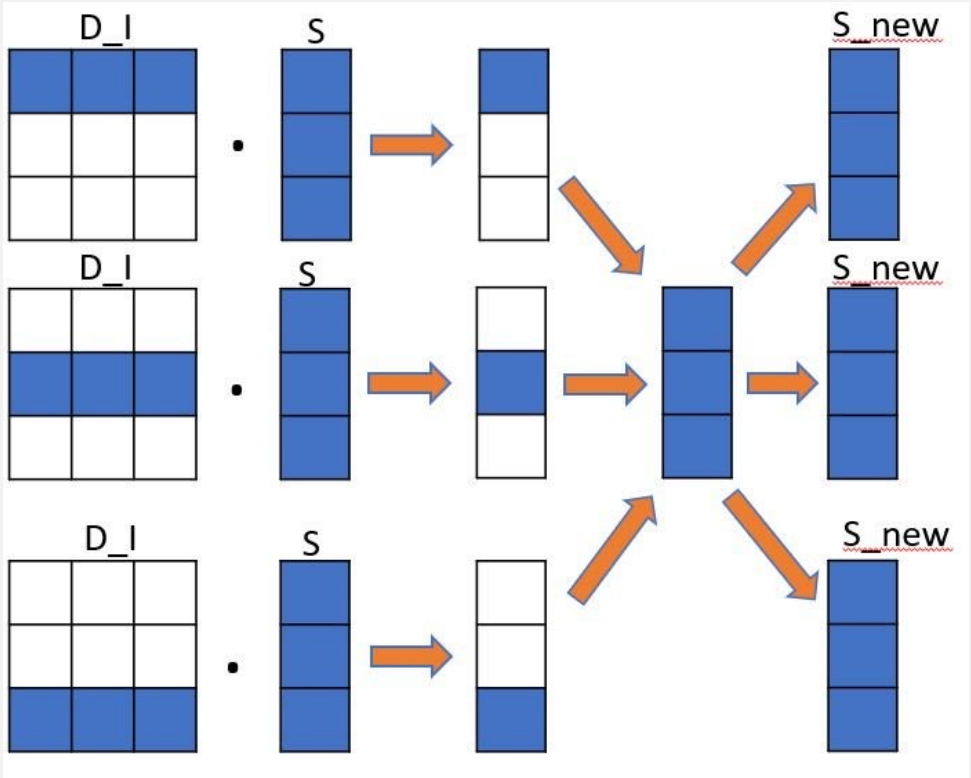
Self-attention allows the model to capture global dependencies and relationships within the input sequence. Unlike traditional recurrent or convolutional neural networks, which have limited receptive fields, self-attention can consider interactions between all elements in the sequence simultaneously.

This property enables the model to effectively capture long-range dependencies and understand the relationships between distant elements in the input sequence.

PROPERTIES OF SELF ATTENTION

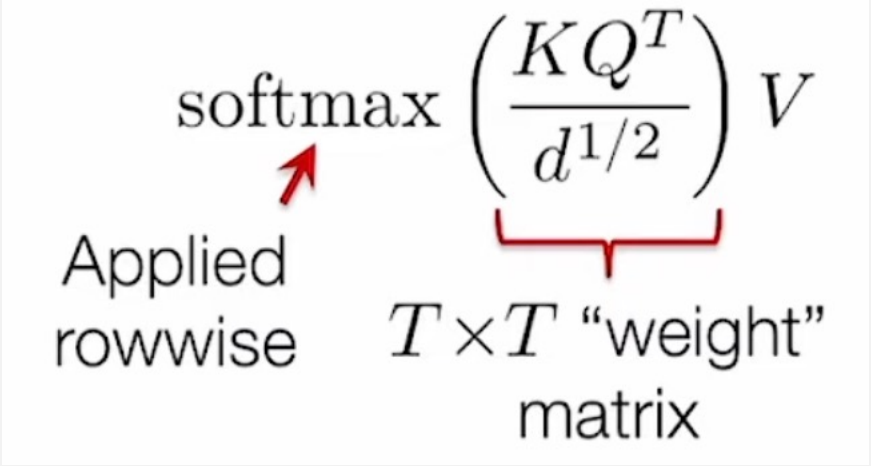
Scalability:

Self-attention is inherently parallelizable, making it scalable to longer sequences. This is because the calculation of attention weights for each pair of input elements can be performed independently, allowing for efficient computation across the entire sequence. As a result, self-attention is well-suited for processing long sequences of data, such as lengthy sentences in natural language processing tasks.



SELF ATTENTION COMPUTE COST

- In self-attention, each token in the input sequence needs to attend to every other token, which requires computing a similarity score between each pair of tokens.
- This involves calculating the dot product of the Q, K, and V matrices for each pair of tokens, resulting in a computational complexity of $O(T^2 d)$.

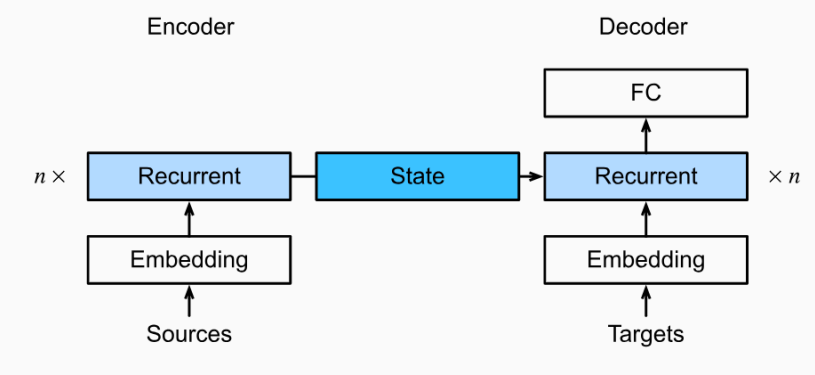


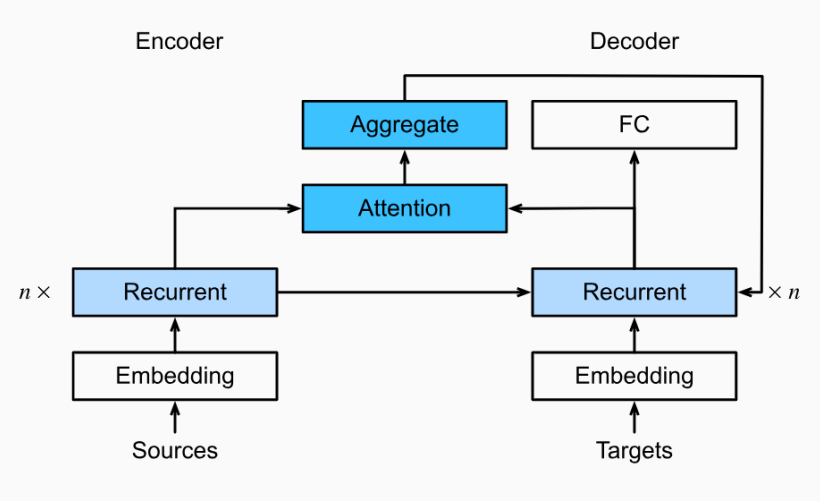
```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class SelfAttention(nn.Module):
    def __init__(self, input_dim):
        super(SelfAttention, self).__init__()
        self.input_dim = input_dim
        self.query = nn.Linear(input_dim, input_dim)
        self.key = nn.Linear(input_dim, input_dim)
        self.value = nn.Linear(input_dim, input_dim)
        self.softmax = nn.Softmax(dim=2)

    def forward(self, x):
        queries = self.query(x)
        keys = self.key(x)
        values = self.value(x)
        scores = torch.bmm(queries, keys.transpose(1, 2)) / (self.input_dim ** 0.5)
        attention = self.softmax(scores)
        weighted = torch.bmm(attention, values)
        return weighted
```

Torch.bmm performs a batch matrix-matrix product of matrices

THE BAHDANAU ATTENTION MECHANISM – FOR SEQ2SEQ PROBLEMS

- When predicting a token, if not all the input tokens are relevant, the model aligns (or attends) only to parts of the input sequence that are deemed relevant to the current prediction. This is then used to update the current state before generating the next token.





THE BAHDANAU ATTENTION MECHANISM

- The key idea is that instead of keeping the state, i.e., the context vector C summarizing the source sentence, as fixed, we dynamically update it, as a function of both the original text (encoder hidden states h_t) and the text that was already generated (decoder hidden states $s_{t'-1}$).
- This yields $C_{t'}$, which is updated after any decoding time step t'.
- Suppose that the input sequence is of length T. In this case the context vector is the output of attention pooling:

$$C_{t'} = \sum_{t=1}^{T} \alpha(s_{t'-1}, h_t) h_t$$

Where $s_{t'-1}$ is the query and h_t is both the key and value.

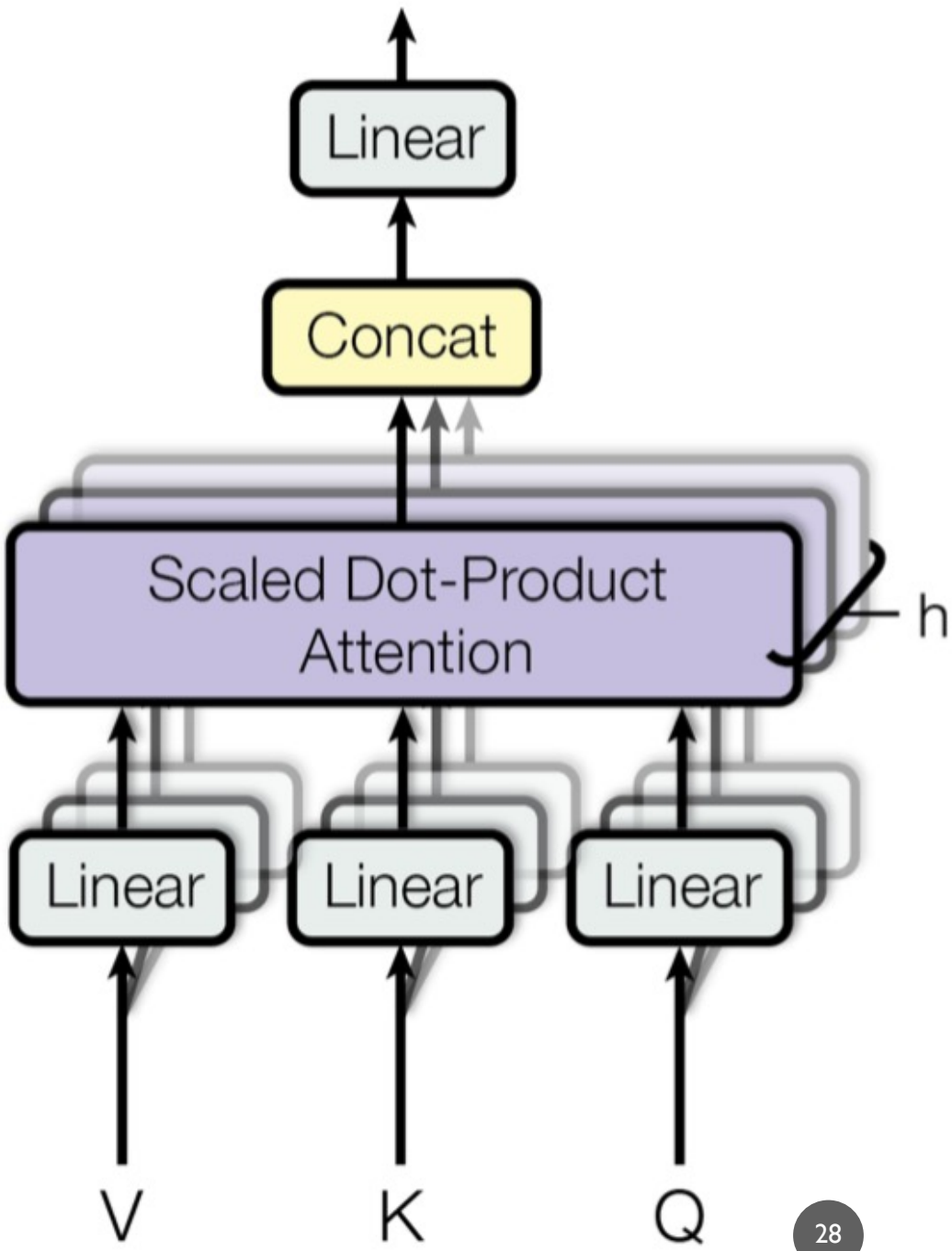
MULTI-HEAD ATTENTION

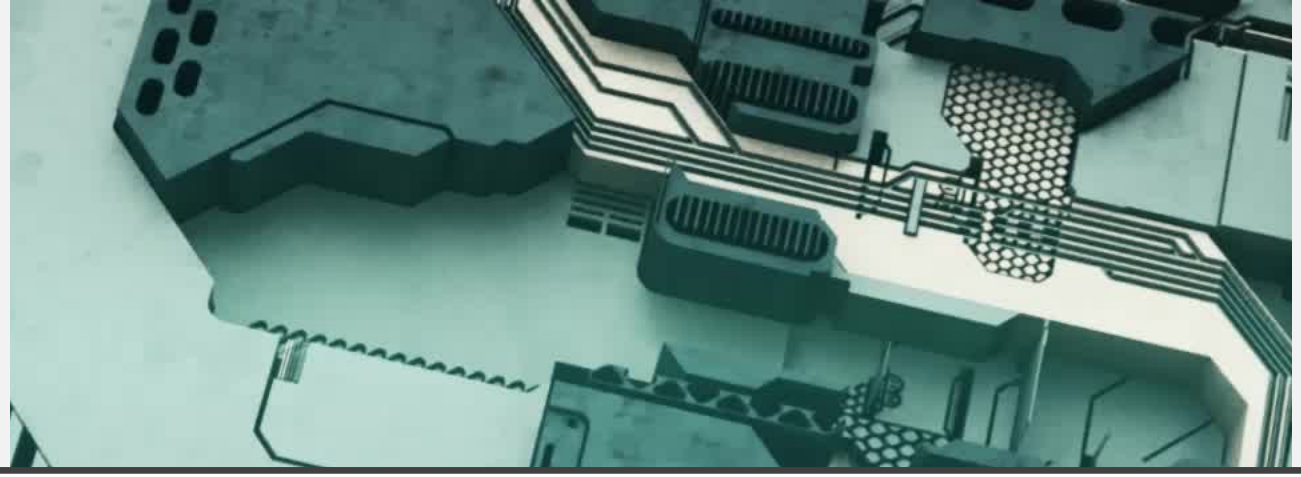
Multiple attention heads capture different aspects of the input sequence.

Each head calculates its own set of attention scores, and the results are concatenated and transformed to produce the final attention weights.

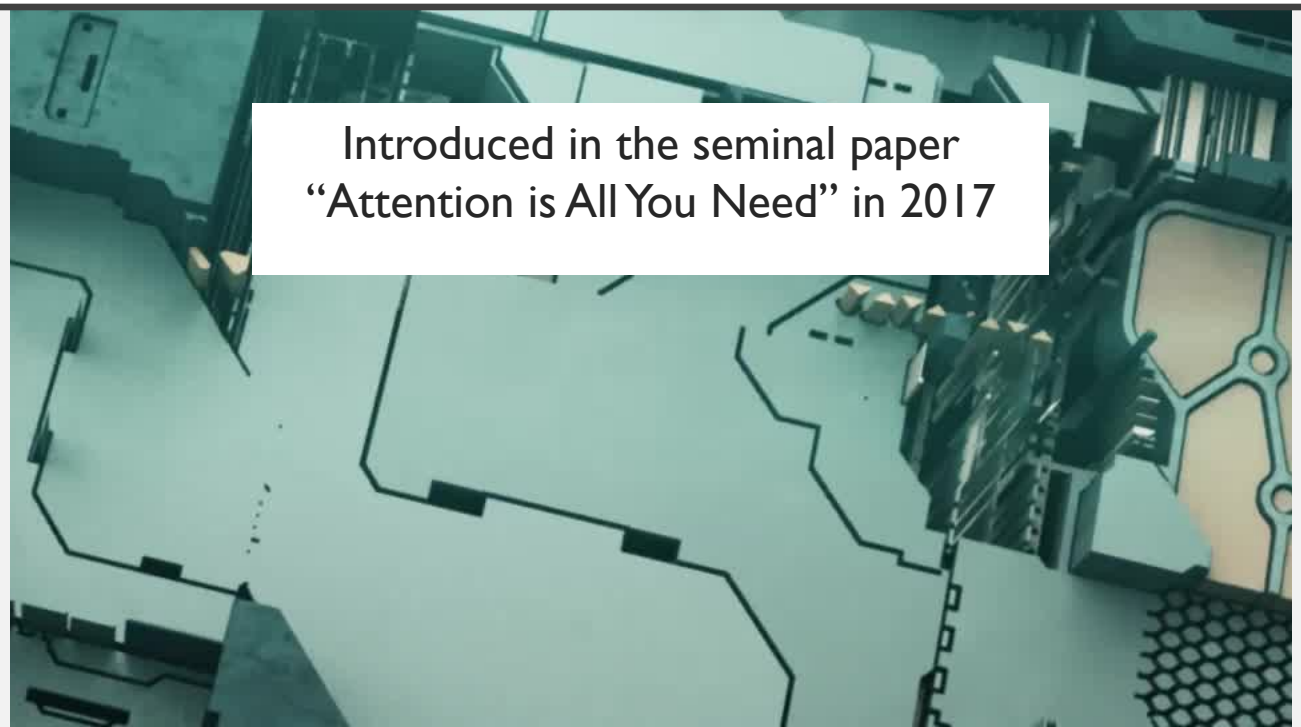
$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\text{head}_1, \ldots, \text{head}_h]\mathbf{W}_0$$

$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right)$$





TRANSFORMERS



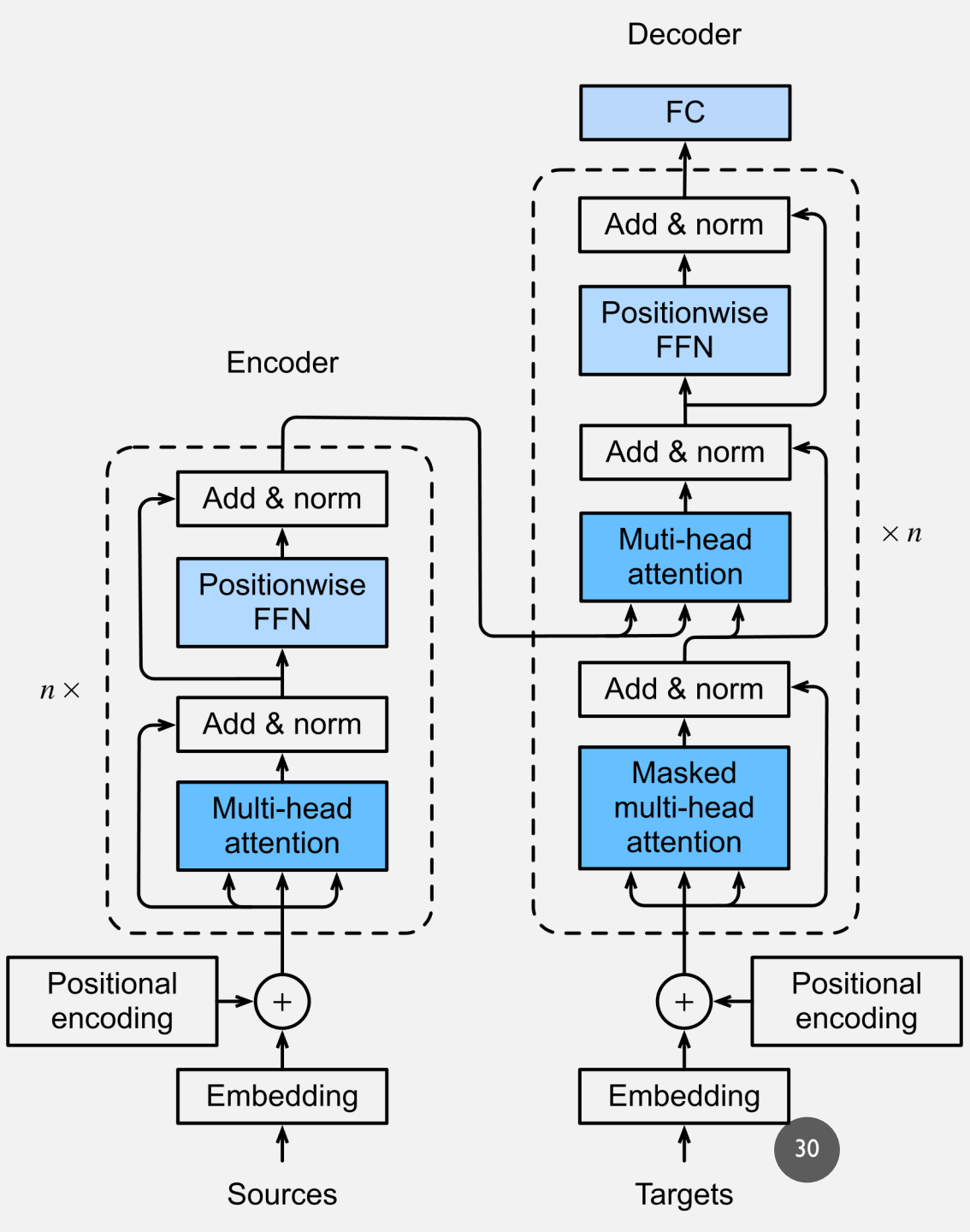
Introduced in the seminal paper
"Attention is All You Need" in 2017

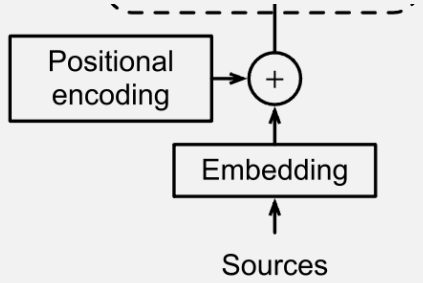
TRANSFORMERS FOR SEQUENCE MODELING

The transformer architectures uses a series of attention mechanisms (and feedforward layers) to process a sequence.

The transformer is composed of an encoder and decoder for S2S.

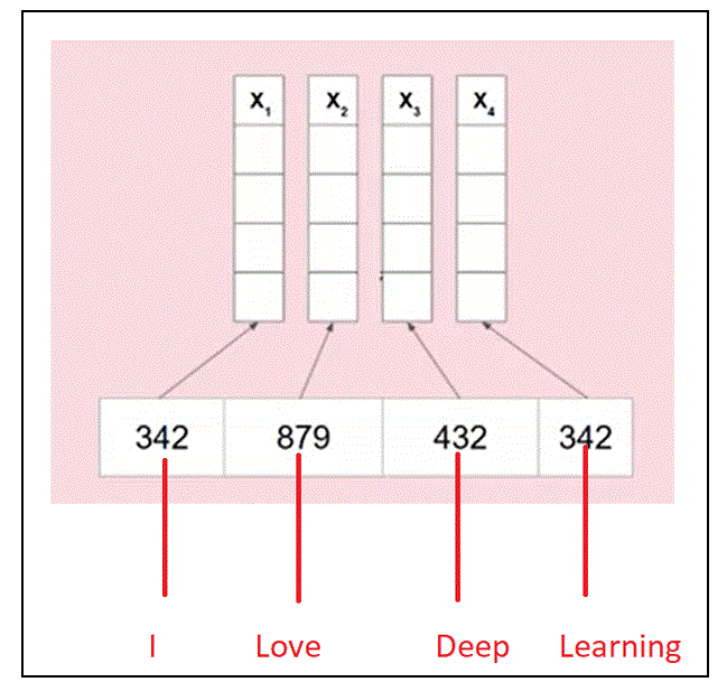
All time steps are processed in parallel, avoids the need of sequential processing like in RNNs.

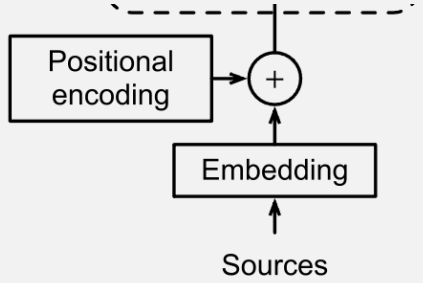




POSITIONAL ENCODING

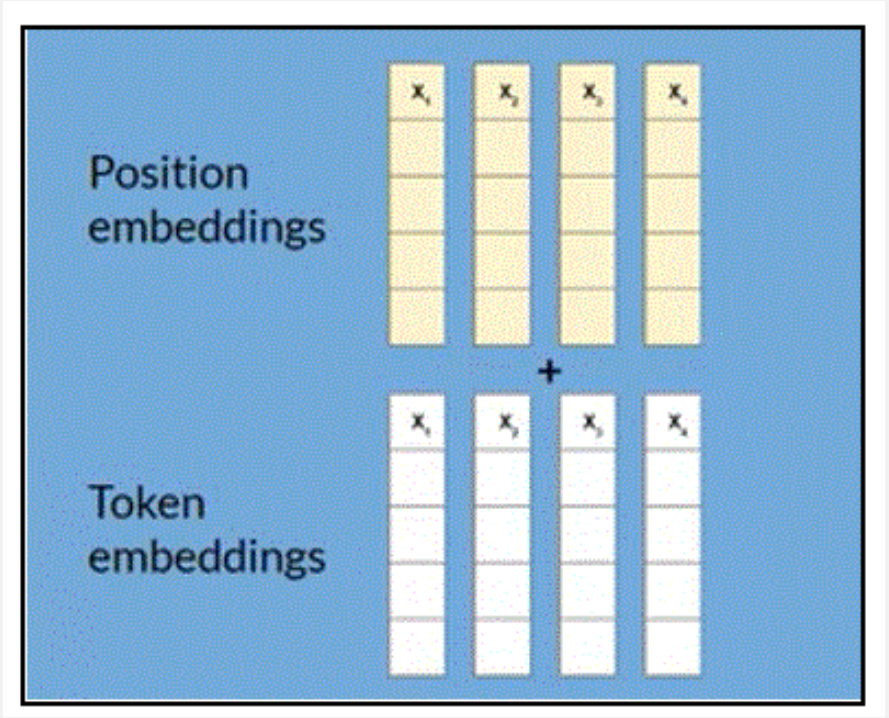
- We already saw that in order to input text into a neural net, we need to create an embedding because these models expect vectors.
- However, in the self-attention mechanism, the information about the position of the word is not saved because we calculate similarity between all pairs!
- Hence, we need to add another component to the embedding which will encode the positional information of each work (token).

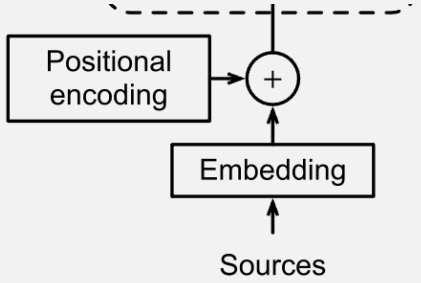




POSITIONAL ENCODING

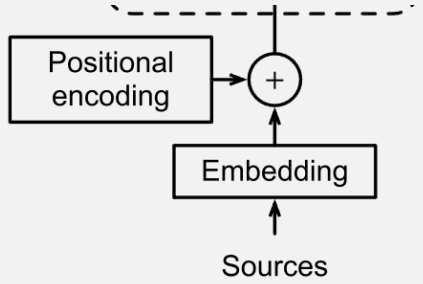
- Positional Encodings can be looked upon as an identifier – that tells the word embeddings of the transformer of the whereabouts about the piece of word /input within a sequence of words.
- These embedding are then added to the initial vector representation of the input.





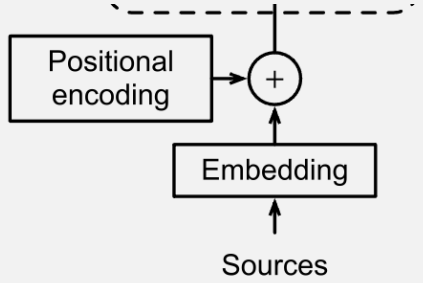
REQUIREMENTS FOR POSITIONAL EMBEDDINGS

1. The positional embedding should not be affected by the sequence length or what input it refers to. It should only depend on the position.
2. Since positional embeddings are offsets that get added to the original word embedding, they cannot be too large. Positional embeddings that are too large might change the semantic meaning of the word embedded.



FORMULATION OF POSITIONAL EMBEDDINGS

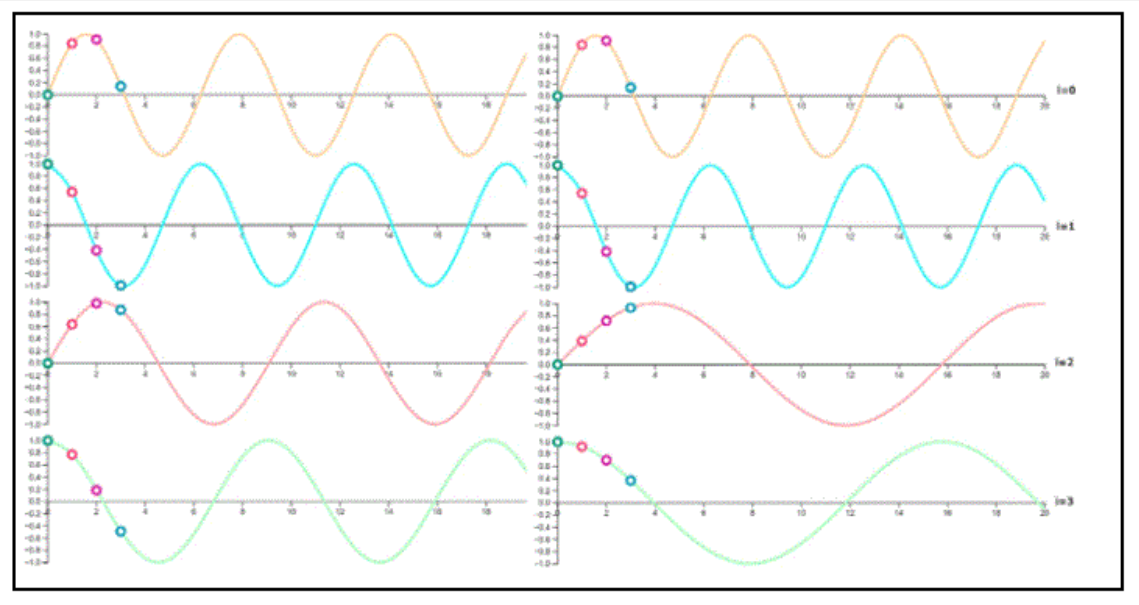
- We want a bounded function – to not influence the embedding vectors too much.
- We want a function that is not limited in its range to allow infinite sequences.
- Solution – use Sine or Cosine functions! Bounded between -1 and 1 and defined up to infinity.
- Problem – they are periodic function and may repeat values for different positions.



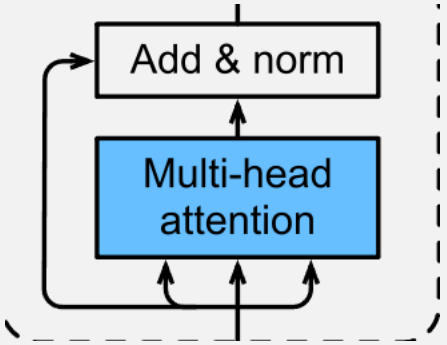
POSITIONAL EMBEDDINGS

- Solution: choose a combination of sines and cosines with different frequencies.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



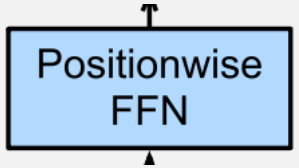
ADD & LAYER NORM



- Inspired by Resnet, a residual connection is employed at every step of the transformer (Add).
- Layer Normalization - computes the layer normalization statistics over all the hidden units in the same layer as follows:

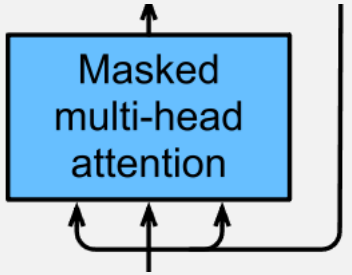
$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$



POINTWISE FFN

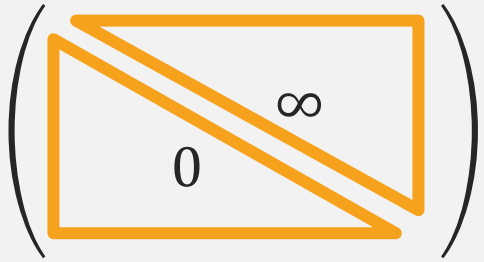
- We apply feed forward layers to the output from the self-attention layer after normalization to allow the network to learn non-linear relations within the data.

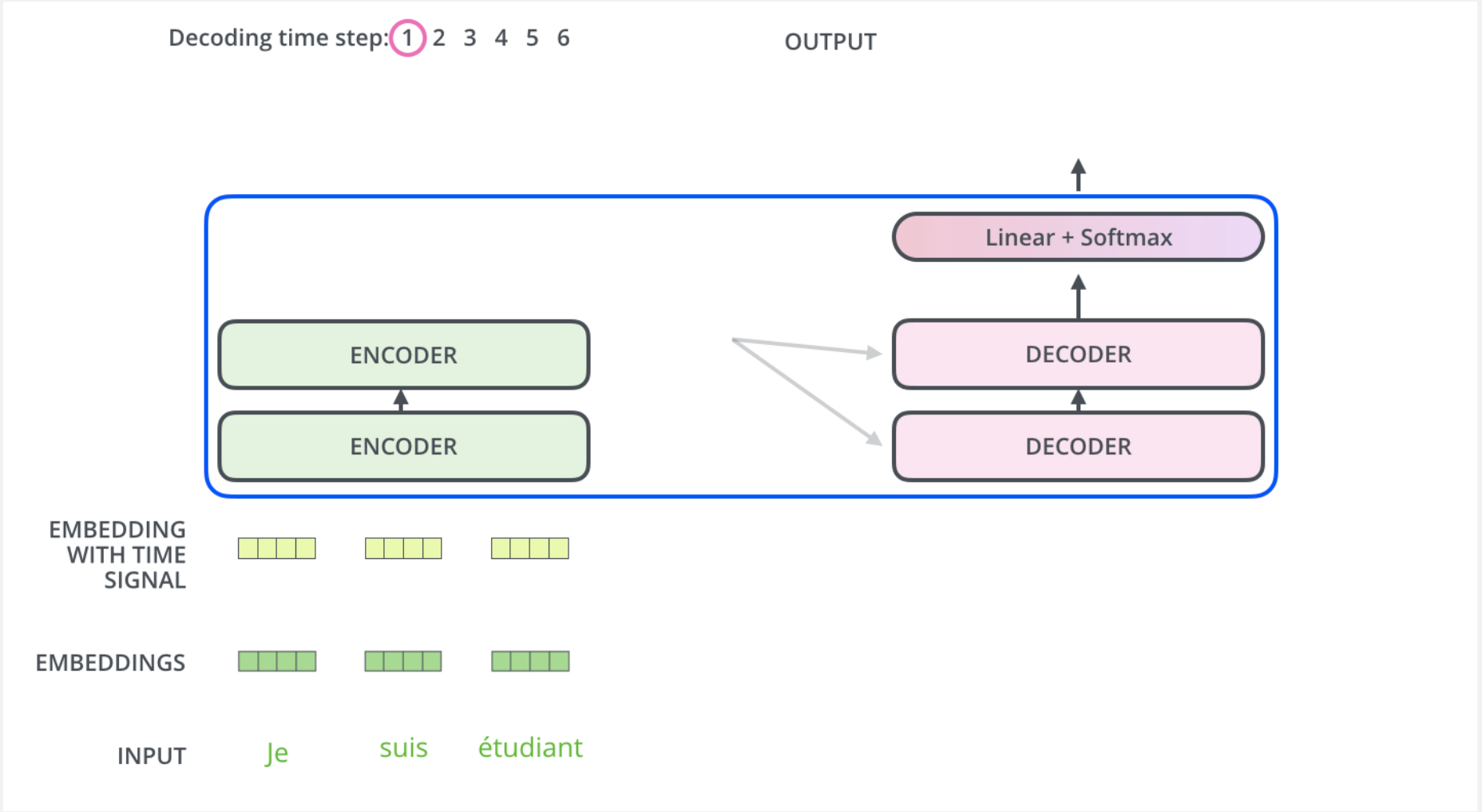


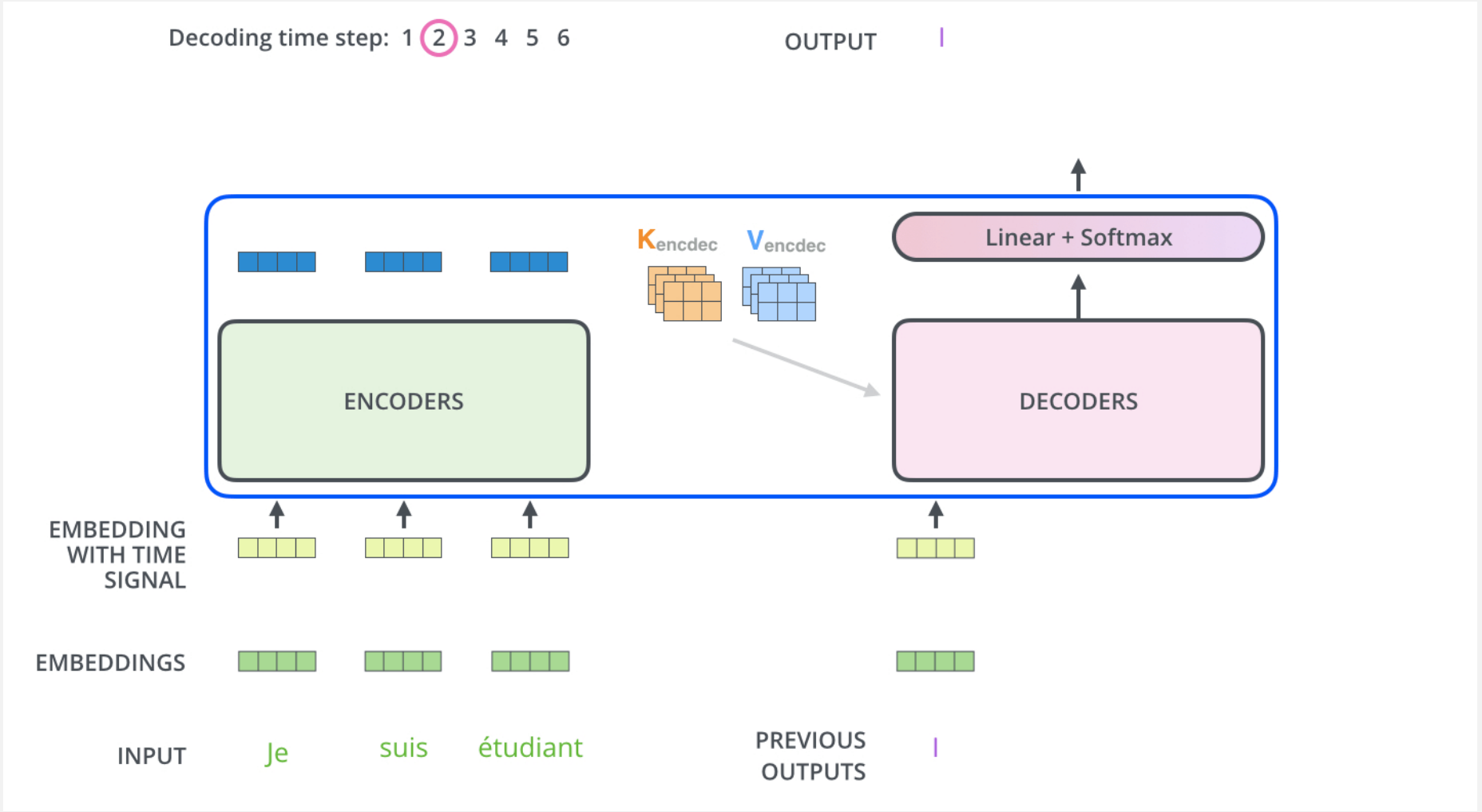
MASKED SELF-ATTENTION

- For autoregressive tasks, we don't want all outputs to depend on all inputs. For example – the prediction in time 6 should not depend on the input from time 10.
- To solve this problem, we can mask the softmax operator to assign zero weights to any "future" time steps.

$$softmax\left(\frac{KQ^T}{d^{1/2}} - M\right)V$$

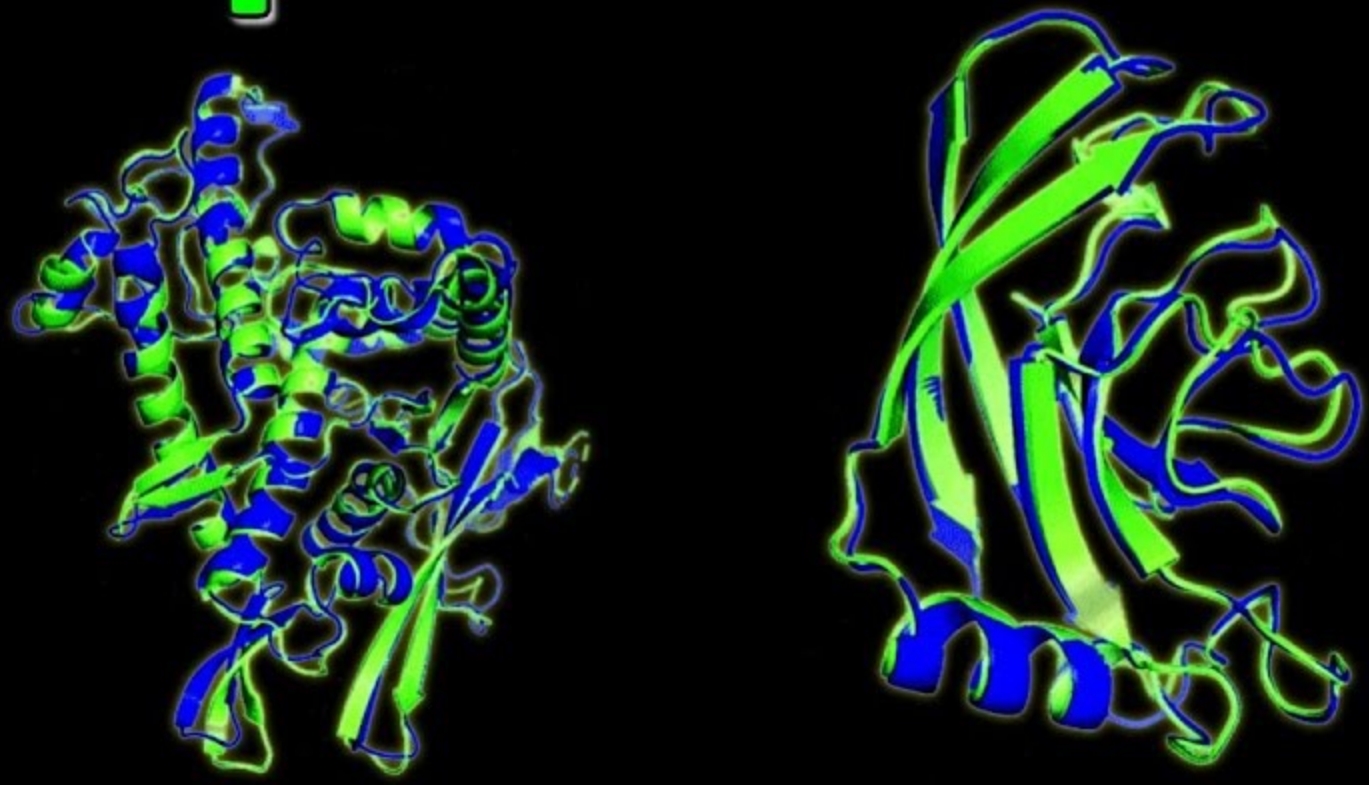
Where $M =$ 


Decoding time step: 1 2 3 4 5 6
OUTPUT
Linear + Softmax
ENCODER
ENCODER
DECODER
DECODER
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
K_{encdec}
V_{encdec}
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Google DeepMind's

AlphaFold 2



AI Breakthrough in Biology

https://borisburkov.net/2021-12-25-1/

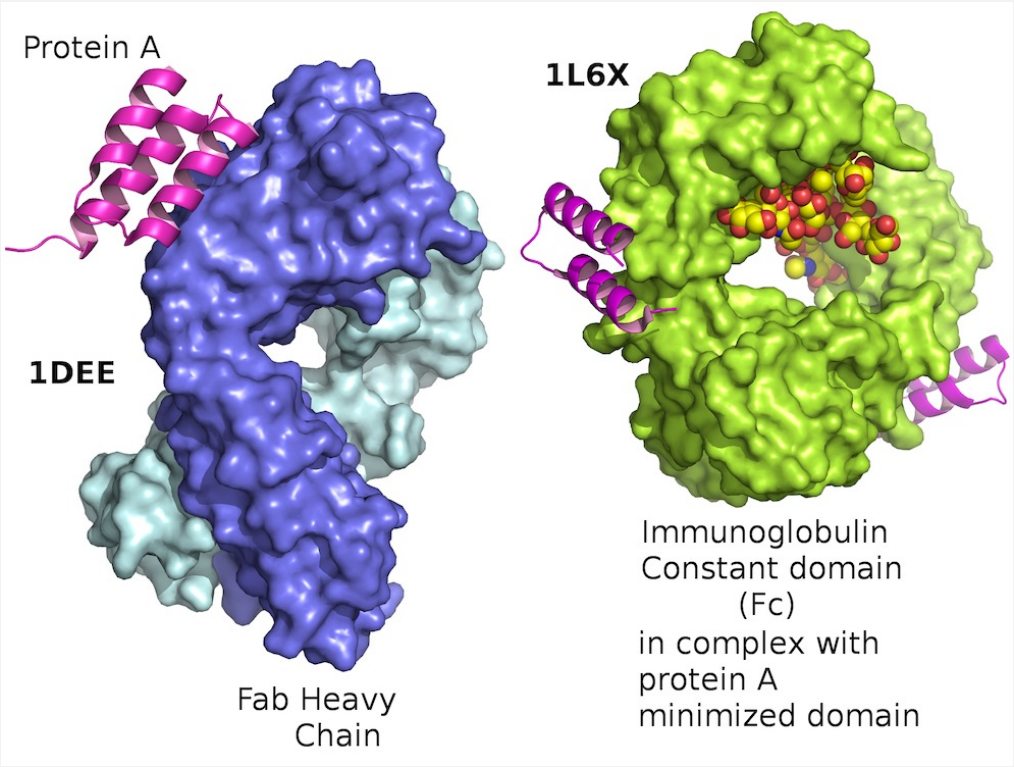
https://rootsofprogress.org/alphafold-protein-folding-explainer

THE PROTEIN FOLDING PROBLEM

Proteins are long chains of aminoacids.

Your DNA encodes these sequences, and RNA helps manufacture proteins according to this genetic blueprint.

Proteins are synthesized as linear chains, but they don't stay that way. They fold up in complex, globular shapes.

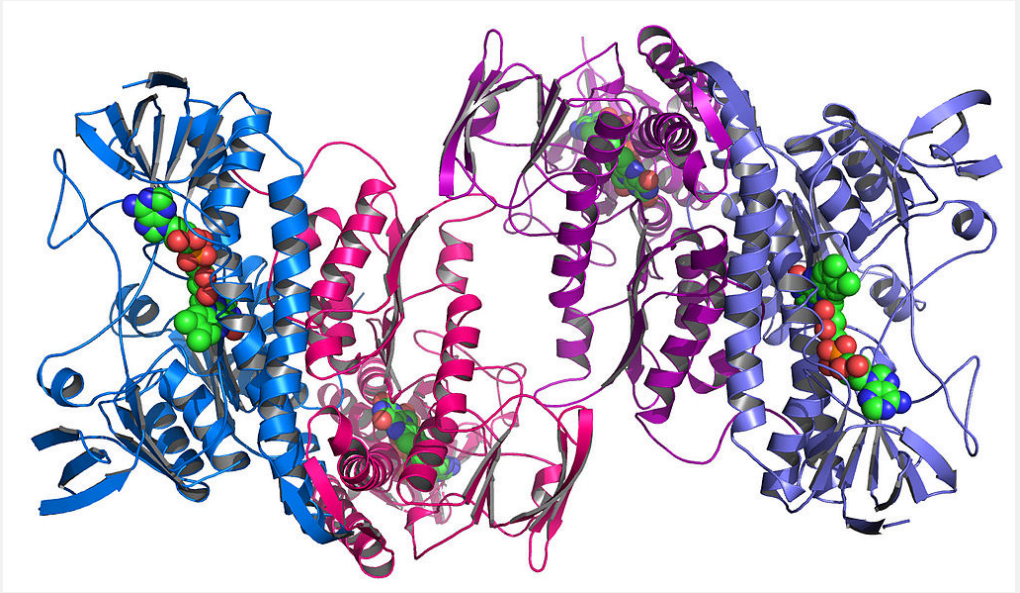


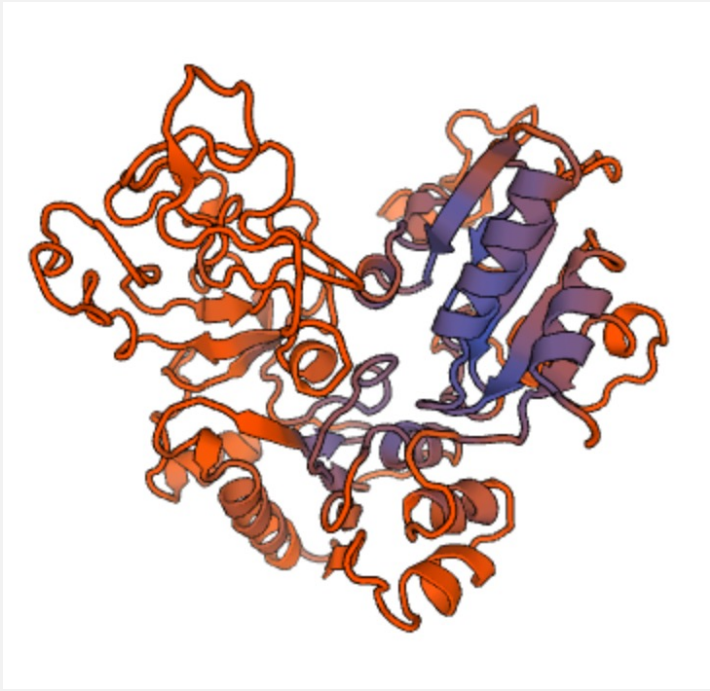
WHY DOES THIS BIG TANGLE OF AMINO ACIDS MATTER?

Protein structure is not random!

Each protein folds in a specific, unique, and largely predictable way that is essential to its function.

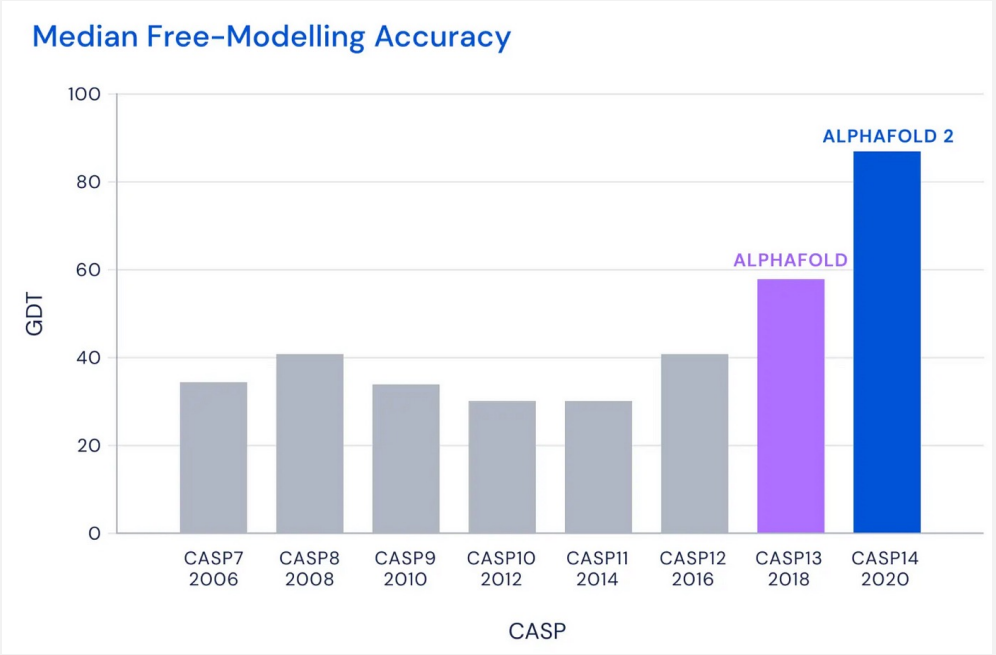
Protein 3D structures are useful for understanding their function and drug discovery.





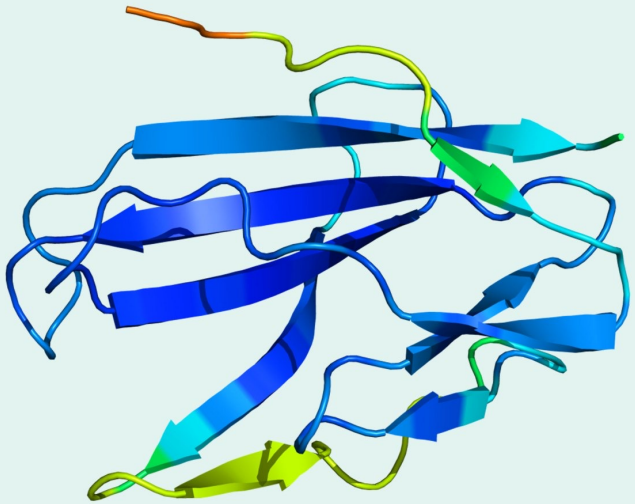
PERFORMANCE ON THE PROTEIN FOLDING PROBLEM

An ongoing research project/competition called Critical Assessment of Protein Structure Prediction (CASP) has been running since 1994 annually to assess the quality of the computational approaches to determining protein structures.



EXAMPLE: SARS-COV-2

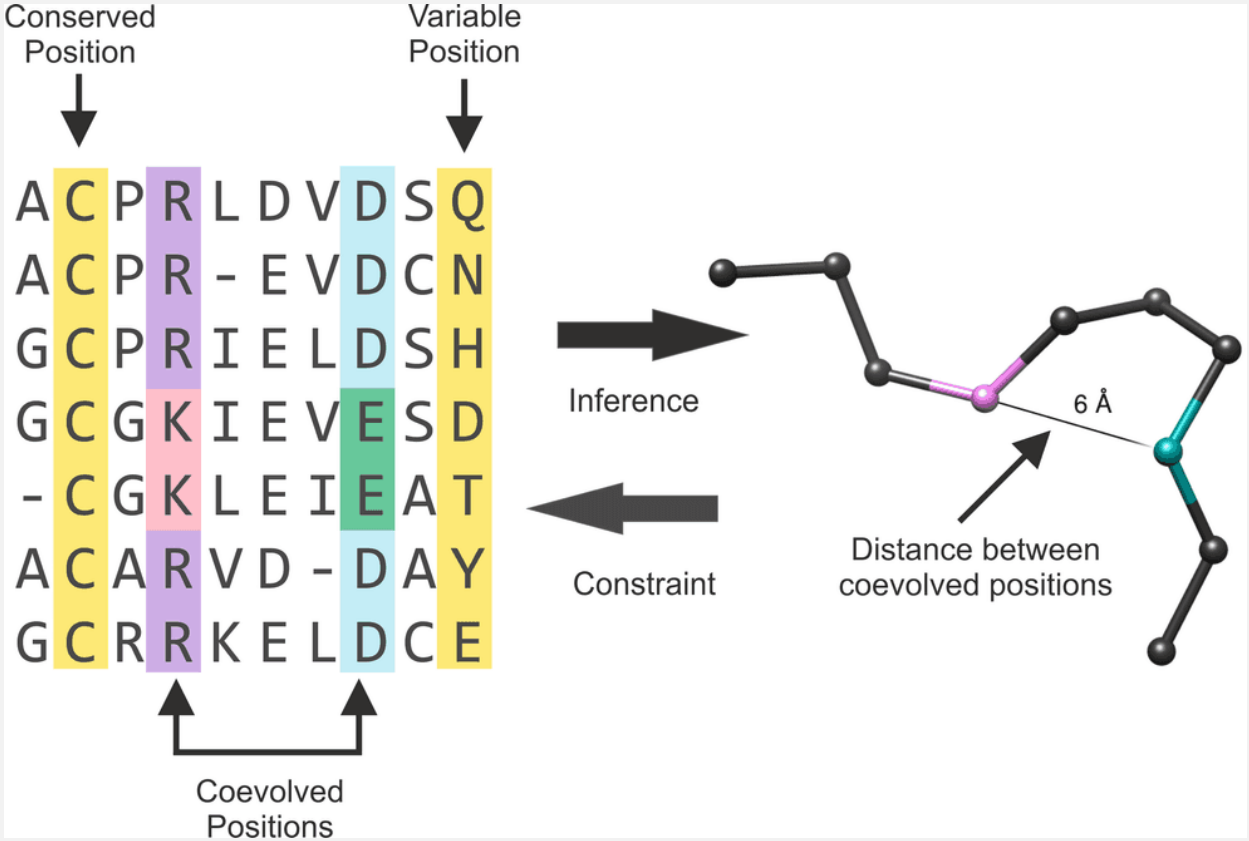
- AlphaFold has been used to predict structures of proteins of SARS-CoV-2, the causative agent of COVID-19.
- Specifically, AlphaFold 2's prediction of the structure of the ORF3a protein was very similar to the structure determined by researchers at University of California, Berkeley using cryo-electron microscopy.
- This specific protein is believed to assist the virus in breaking out of the host cell once it replicates. This protein is also believed to play a role in triggering the inflammatory response to the infection.



PRINCIPLE OF OPERATION

- Oftentimes a protein is conservative in evolution. E.g. human, horse and fish have their own versions of haemoglobin that evolved from the same protein.
- Such versions of the same protein in different species are called homologues.
- Comparison of homologues from different species carries important information and is usually represented as a 2D table, called multiple sequence alignment (MSA).

Protein sequences from different species are written in rows,
so that corresponding residues end up in the same columns.

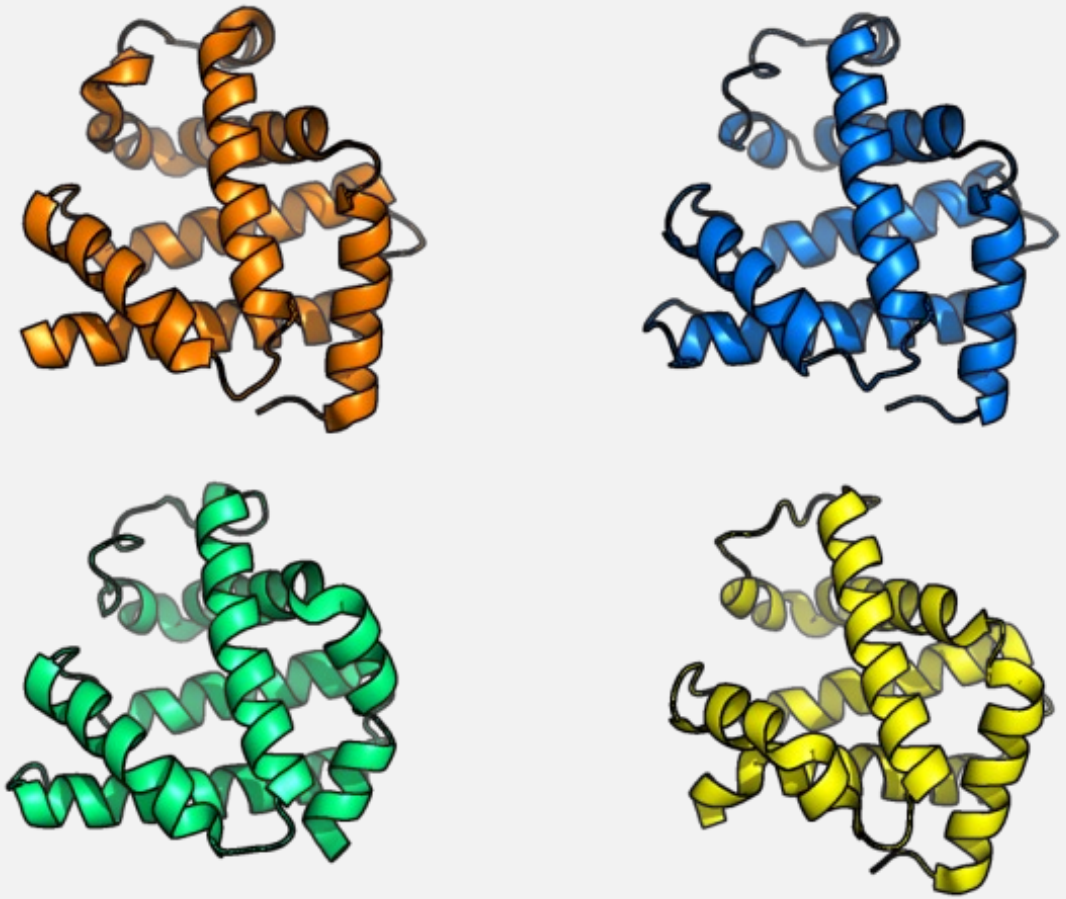


- Conservation of a position in alignment usually implies its importance for protein folding or function.
- Co-evolution of two aminoacid residues of a protein often implies interaction between those aminoacids. This information can be used as the basis for 3D structure prediction.

LEARNING WITHOUT LABELED DATA

DeepMind trained the AlphaFold model on a public dataset of 170,000 proteins with known structures (labeled data) and a much larger dataset of protein sequences with unknown structures (unlabeled data).

How to train from unlabeled data? The more similar those amino acid sequences, the more likely those proteins serve a similar purpose for the organisms they're made in, which means, in turn, they're more likely to share a similar structure.

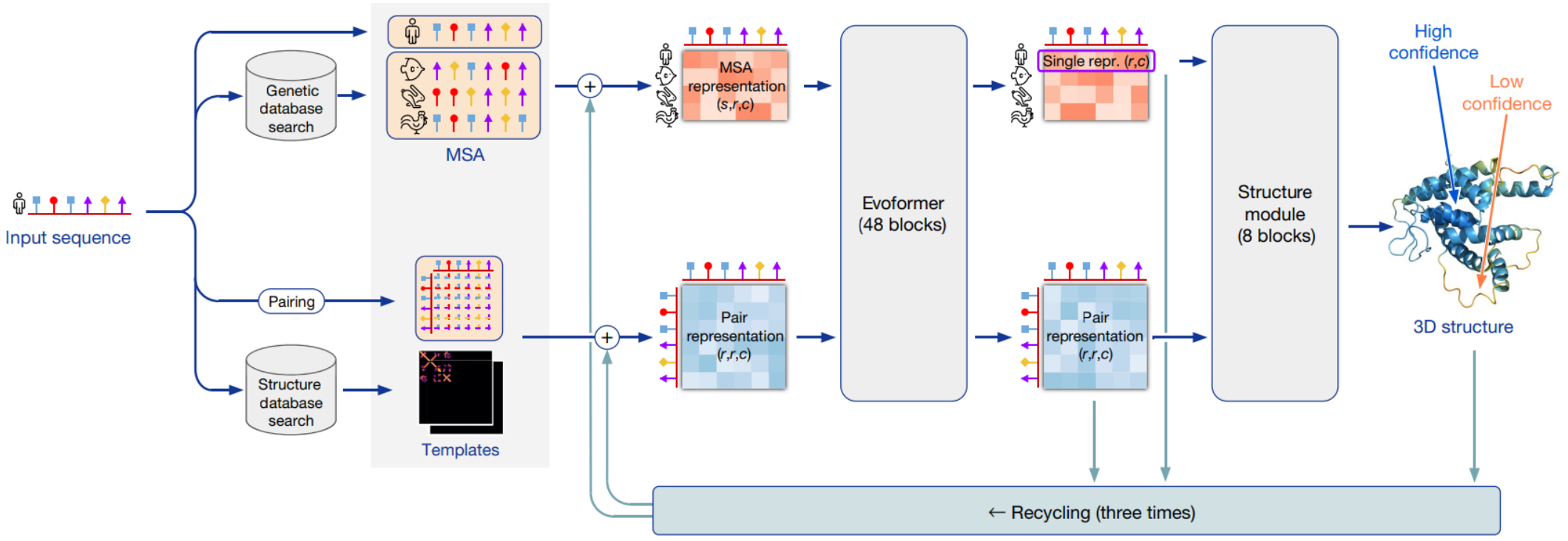


Protein structures of human myoglobin (top left), African elephant myoglobin (top right, 80% sequence identity), blackfin tuna myoglobin (bottom right, 45% sequence identity) and pigeon myoglobin (bottom left, 25% sequence identity)

INPUT TO ALPHAFOLD2

- AlphaFold2 as a system takes a protein sequence on input and predicts its 3D structure. In order to do that, it makes use of multiple databases and programs.
- First, AF2 uses HMMER software to find homologues of input sequence in sequence databases, Uniprot and MGnify.
- AF2 also uses HH-suite package to check, whether any of our homologues has a 3D structure available in PDB (less than 0.1% have such template available).
- Then AF2 generates vector embeddings out of each aminoacid residue of alignment and out of each residue pair in pair representation.

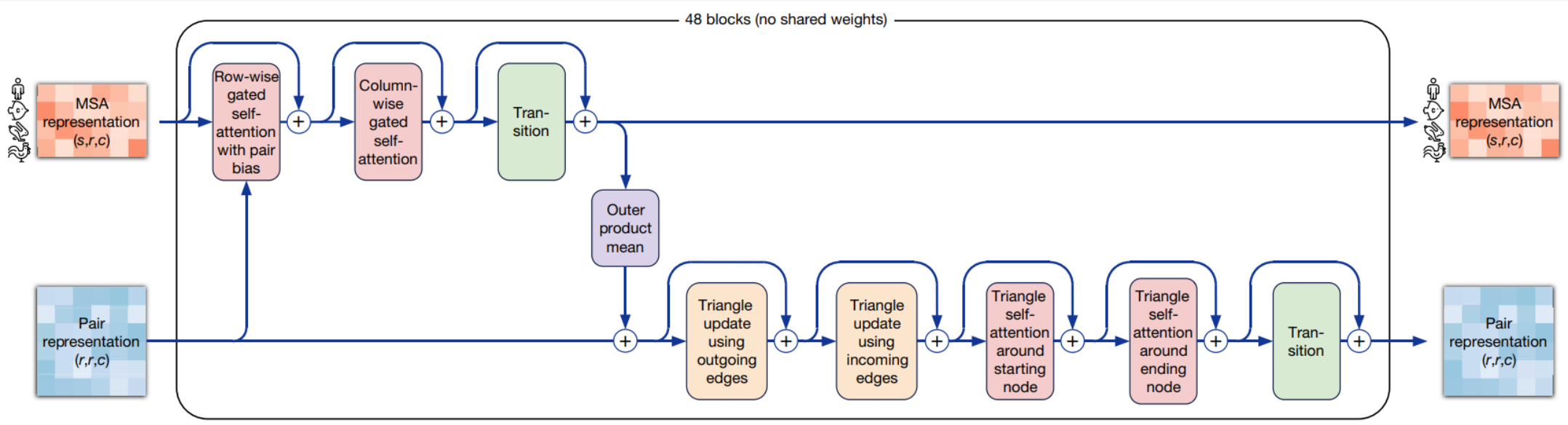
THE ALPHAFOLD2



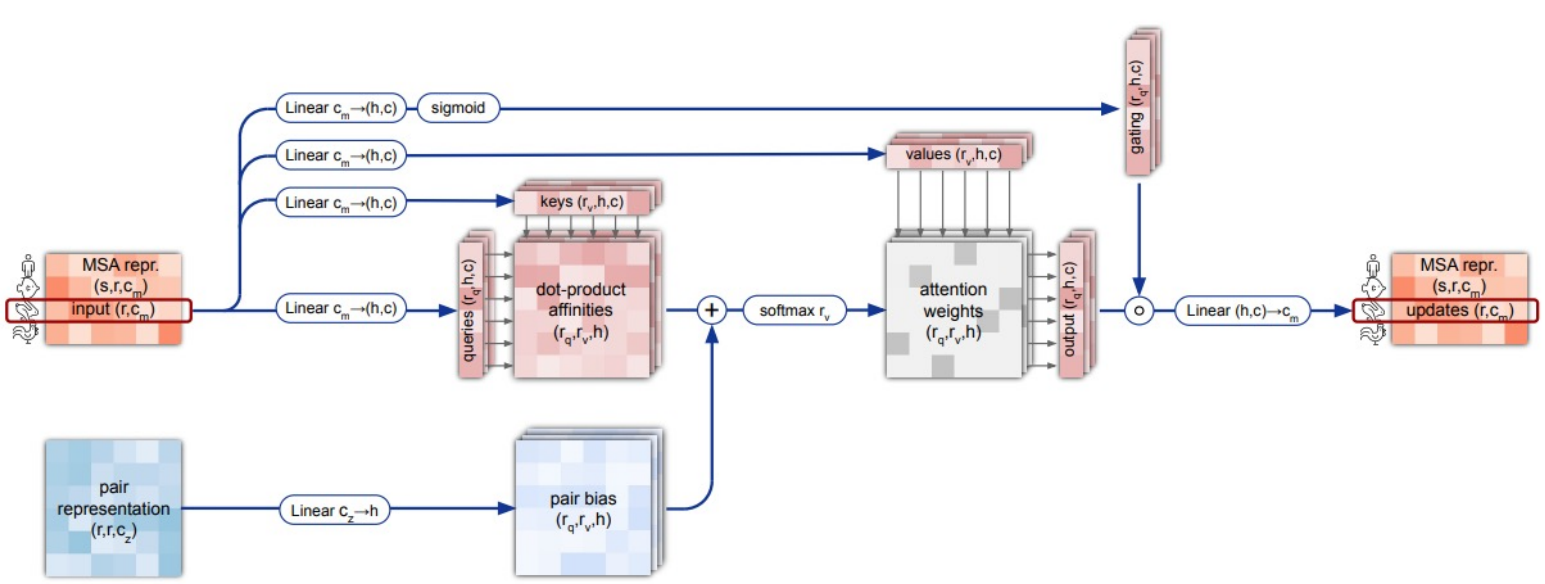
STEP 2: GRAPH AND ATTENTION OVER CONVOLUTIONS

- AF1 is defined as a CNN trained on known protein structures and features derived from evolutionarily related sequences (MSA) to predict the distance between pairs of amino acids of an input sequence to generate a 3D structure.
- An essential component of AF2 is the Evoformer, which uses attention-based mechanisms that allow it to learn correlations in the input data. This attention mechanism gives more flexibility to the network, allowing it to dynamically learn interactions between non-neighboring nodes.

EVOFORMER


48 blocks (no shared weights)
MSA representation (s,r,c)
Row-wise gated self-attention with pair bias
Column-wise gated self-attention
Tran-sition
Outer product mean
Pair representation (r,r,c)
Triangle update using outgoing edges
Triangle update using incoming edges
Triangle self-attention around starting node
Triangle self-attention around ending node
Tran-sition
MSA representation (s,r,c)
Pair representation (r,r,c)

MSA ROW-WISE GATED SELF ATTENTION WITH PAIR BIAS



Supplementary Figure 2 | MSA row-wise gated self-attention with pair bias. Dimensions: s: sequences, r: residues, c: channels, h: heads.

- In the section that deals with the MSA representation, the network first computes row-wise attention (horizontal direction), identifying which amino acids in the sequence are more related to each other.
- When updating the relation between amino acids, the network adds a pair bias term that is calculated directly from the current pair representation.

Algorithm 7 MSA row-wise gated self-attention with pair bias

def $\text{MSARowAttentionWithPairBias}(\{\mathbf{m}_{si}\}, \{\mathbf{z}_{ij}\}, c = 32, N_{\text{head}} = 8)$:

 # Input projections

$\mathbf{m}_{si} \leftarrow \text{LayerNorm}(\mathbf{m}_{si})$

$\mathbf{q}_{si}^h, \mathbf{k}_{si}^h, \mathbf{v}_{si}^h = \text{LinearNoBias}(\mathbf{m}_{si})$ $\qquad \mathbf{q}_{si}^h, \mathbf{k}_{si}^h, \mathbf{v}_{si}^h \in \mathbb{R}^c,\ h \in \{1, \ldots, N_{\text{head}}\}$

$b_{ij}^h = \text{LinearNoBias}(\text{LayerNorm}(\mathbf{z}_{ij}))$

$\mathbf{g}_{si}^h = \text{sigmoid}\left(\text{Linear}(\mathbf{m}_{si})\right)$ $\qquad \mathbf{g}_{si}^h \in \mathbb{R}^c$

 # Attention

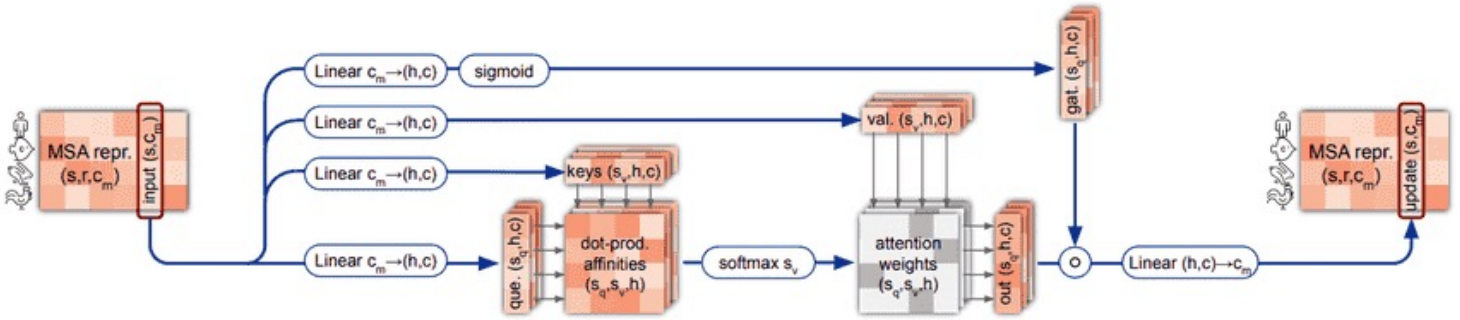
$a_{sij}^h = \text{softmax}_j\left(\frac{1}{\sqrt{c}}\, \mathbf{q}_{si}^{h\top} \mathbf{k}_{sj}^h + b_{ij}^h\right)$

$\mathbf{o}_{si}^h = \mathbf{g}_{si}^h \odot \sum_j a_{sij}^h \mathbf{v}_{sj}^h$

 # Output projection

$\tilde{\mathbf{m}}_{si} = \text{Linear}\left(\text{concat}_h(\mathbf{o}_{si}^h)\right)$ $\qquad \tilde{\mathbf{m}}_{si} \in \mathbb{R}^{c_m}$

return $\{\tilde{\mathbf{m}}_{si}\}$



Supplementary Figure 3 | MSA column-wise gated self-attention. Dimensions: s: sequences, r: residues, c: channels, h: heads.

Algorithm 8 MSA column-wise gated self-attention

def $\text{MSAColumnAttention}(\{\mathbf{m}_{si}\}, c = 32, N_{\text{head}} = 8)$:

 # Input projections

$\mathbf{m}_{si} \leftarrow \text{LayerNorm}(\mathbf{m}_{si})$

$\mathbf{q}^h_{si}, \mathbf{k}^h_{si}, \mathbf{v}^h_{si} = \text{LinearNoBias}(\mathbf{m}_{si})$ $\quad \mathbf{q}^h_{si}, \mathbf{k}^h_{si}, \mathbf{v}^h_{si} \in \mathbb{R}^c,\ h \in \{1, \ldots, N_{\text{head}}\}$

$\mathbf{g}^h_{si} = \text{sigmoid}\left(\text{Linear}(\mathbf{m}_{si})\right)$ $\quad \mathbf{g}^h_{si} \in \mathbb{R}^c$

 # Attention

$a^h_{sti} = \text{softmax}_t\left(\frac{1}{\sqrt{c}}\, \mathbf{q}^{h\top}_{si} \mathbf{k}^h_{ti}\right)$

$\mathbf{o}^h_{si} = \mathbf{g}^h_{si} \odot \sum_t a^h_{sti} \mathbf{v}^h_{st}$

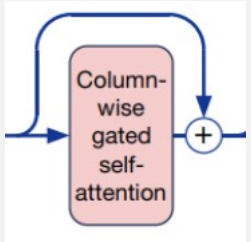
 # Output projection

$\tilde{\mathbf{m}}_{si} = \text{Linear}\left(\text{concat}_h(\mathbf{o}^h_{si})\right)$ $\quad \tilde{\mathbf{m}}_{si} \in \mathbb{R}^{c_m}$

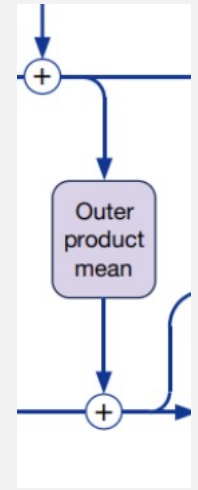
return $\{\tilde{\mathbf{m}}_{si}\}$

MSA REP. CONT.

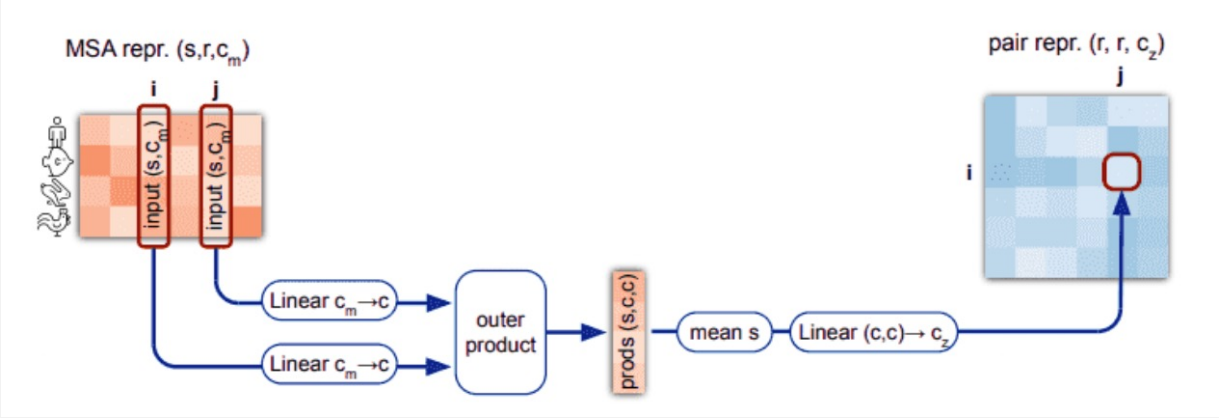
- Column-wise attention (vertical direction) is applied to identify which sequences are more important.
- This step helps AF2 identify conservative or co-evolved positions.



MSA REP. CONT.

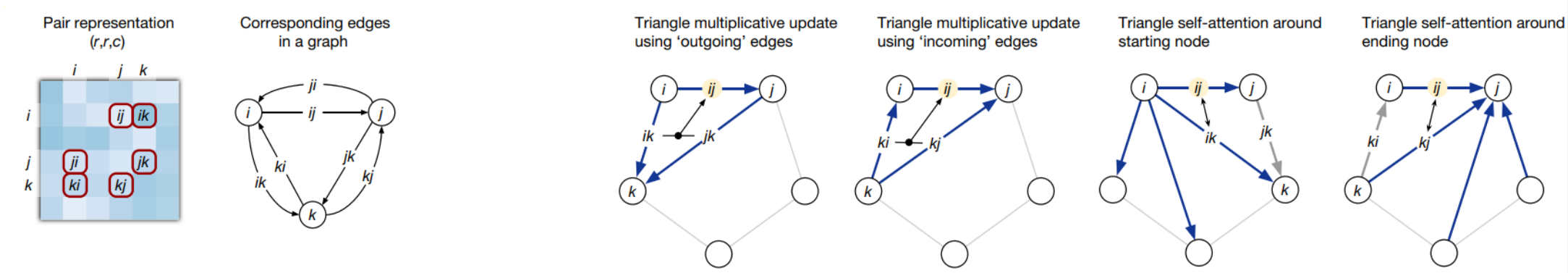


- Two points of communication to refine the updates can be found in the Evoformer section.
- The first is the bias in where the pair representation communicates and helps update the MSA.
- The outer product mean block is the other communication point, where the MSA representation provides a way to update the pair representation.

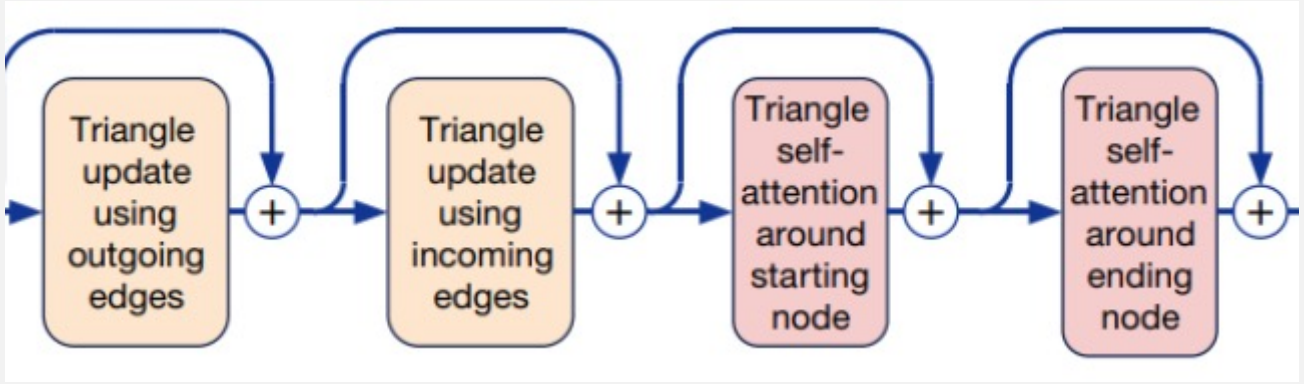


PAIR REPRESENTATION PROCESSING

- AF2 uses attention to process how residue pairs relate to one another in a graph representation.
- This graph representation is arranged in terms of a triplet of amino acids (nodes) and their corresponding edges.



- The pair representation is being updated in a process called triangle multiplicative update that updates an edge (distance between residues) based on the two other nodes and their edges.
- Then these three nodes are updated using triangular self-attention.
- Attention allows further geometrical and chemical constraints to be learned by the algorithm while triangular multiplicative updates ensure that no excess attention is paid to a particular residue subsequence.

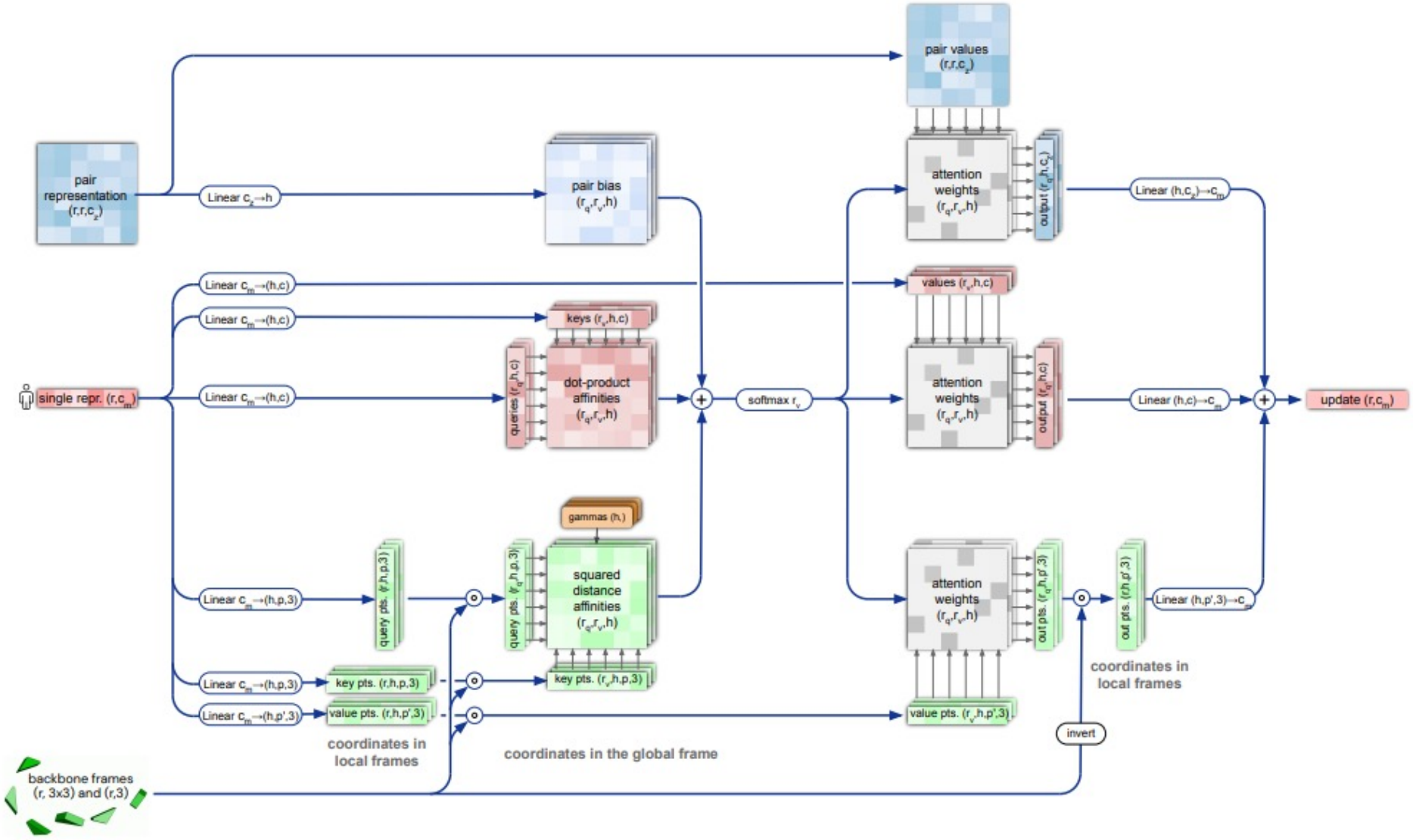


STEP 3: THE STRUCTURE MODULE

- Now that we are done with MSA embeddings and pair representations, it is time to predict the actual protein structure.
- The question is now: how do we get a structure from these representations?
- Answer: the Invariant Point Attention (IPA) module models rotations and displacing of each independent amino acid (as triangles in space) to produce a final 3D structure.

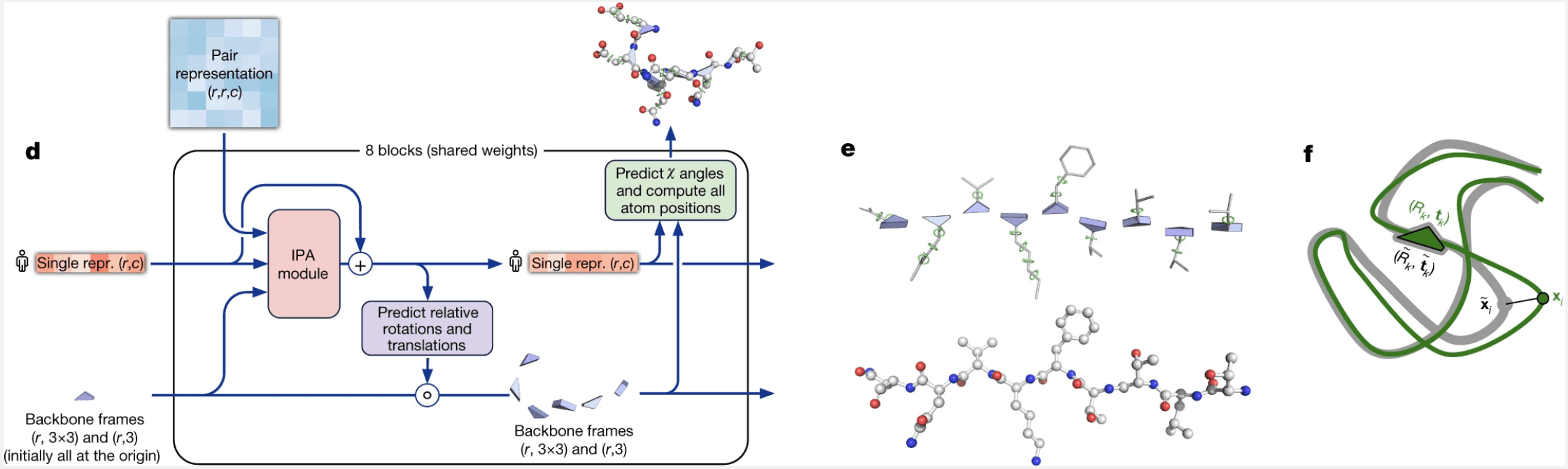
INVARIANT POINT ATTENTION (IPA)

IPA is a new type of attention devised specifically for working with 3D.



Supplementary Figure 8 | Invariant Point Attention Module. (**top, blue arrays**) modulation by the pair representation. (**middle, red arrays**) standard attention on abstract features. (**bottom, green arrays**) Invariant point attention. Dimensions: r: residues, c: channels, h: heads, p: points.

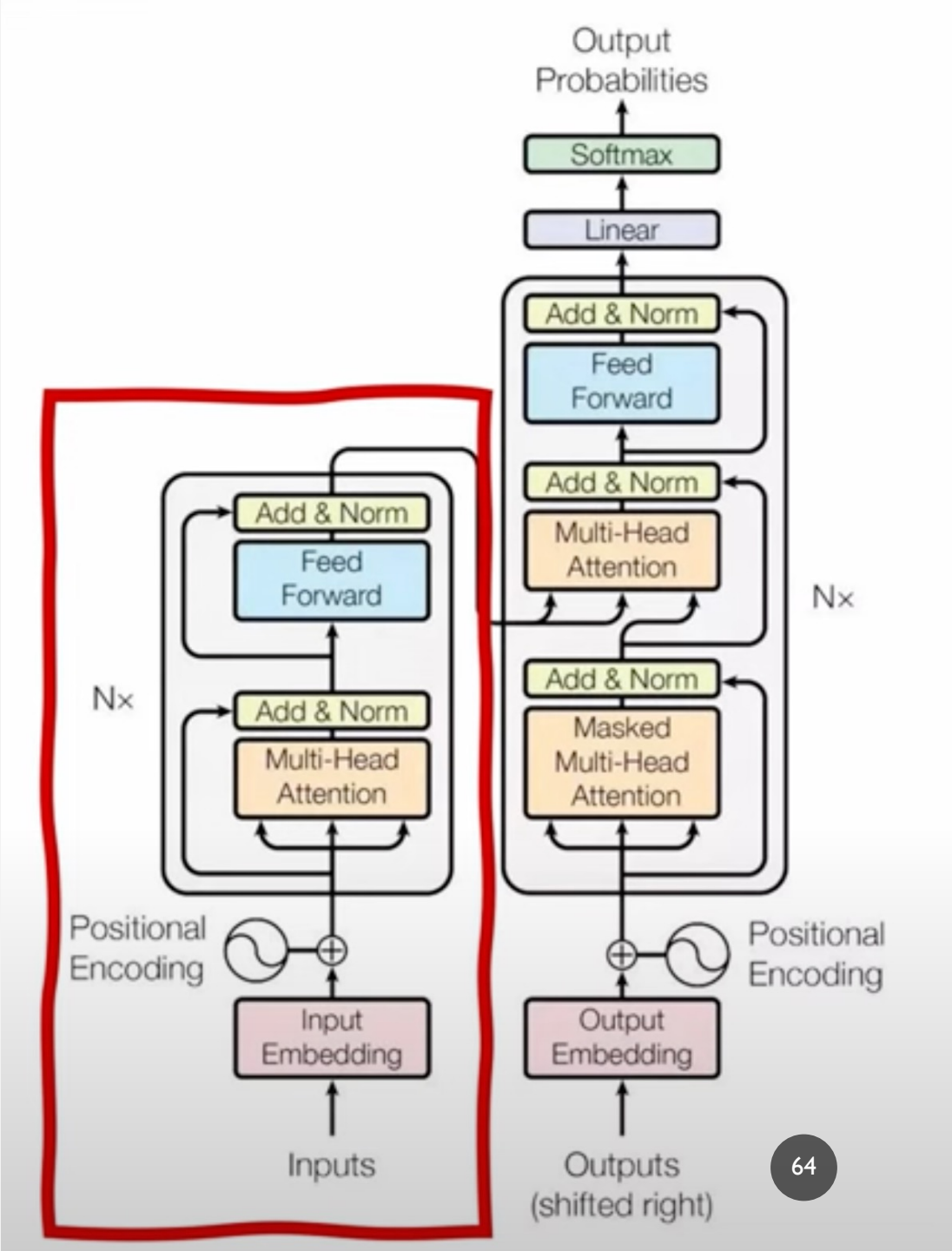
FOR ENRICHMENT



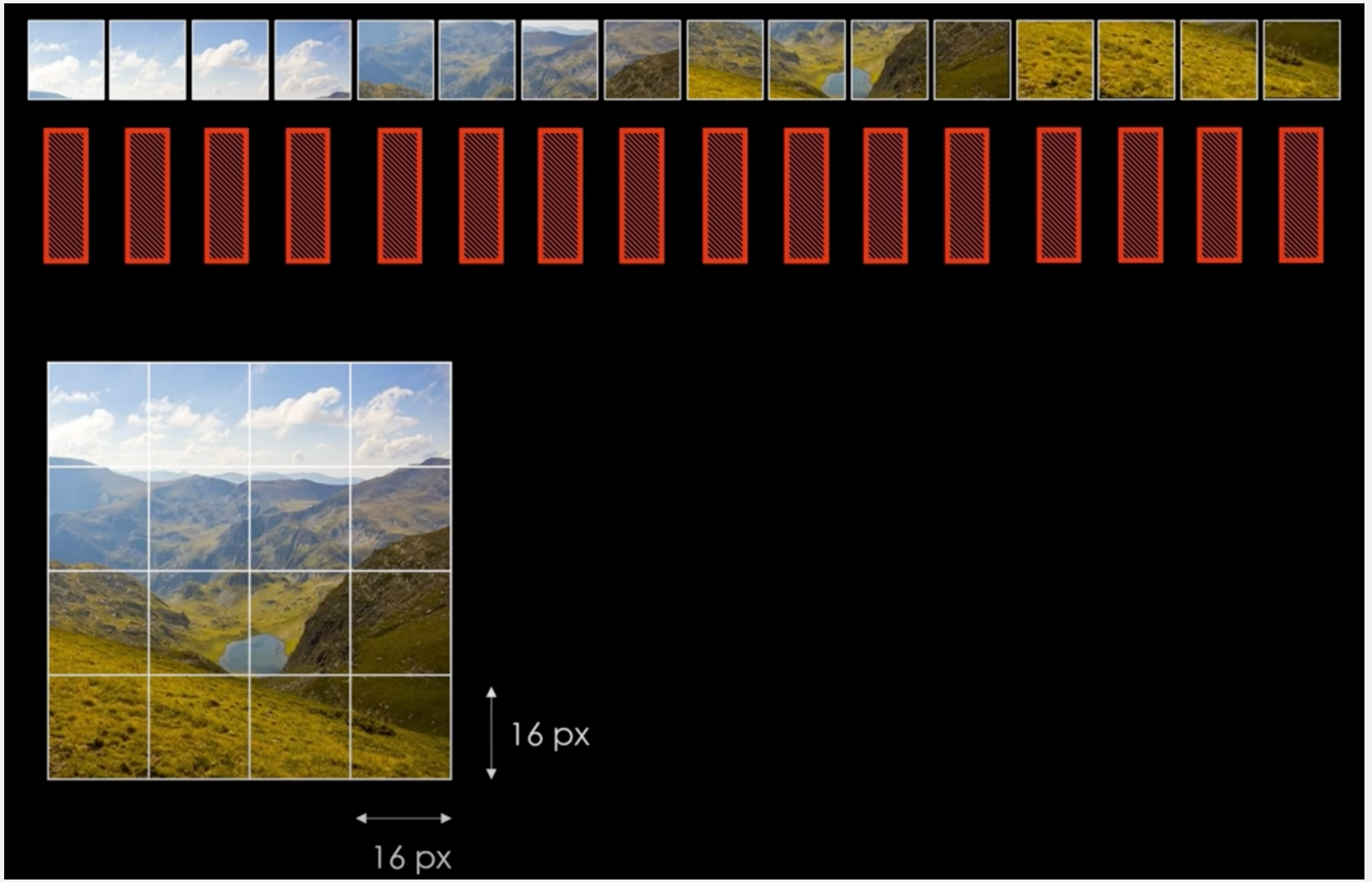
Dosovitskiy et al, 2020. ICLR

TRANSFORMERS FOR IMAGE RECOGNITION

It is important to know that vision transformers only make use of the encoder part of the transformers because we do not aim to generate anything.



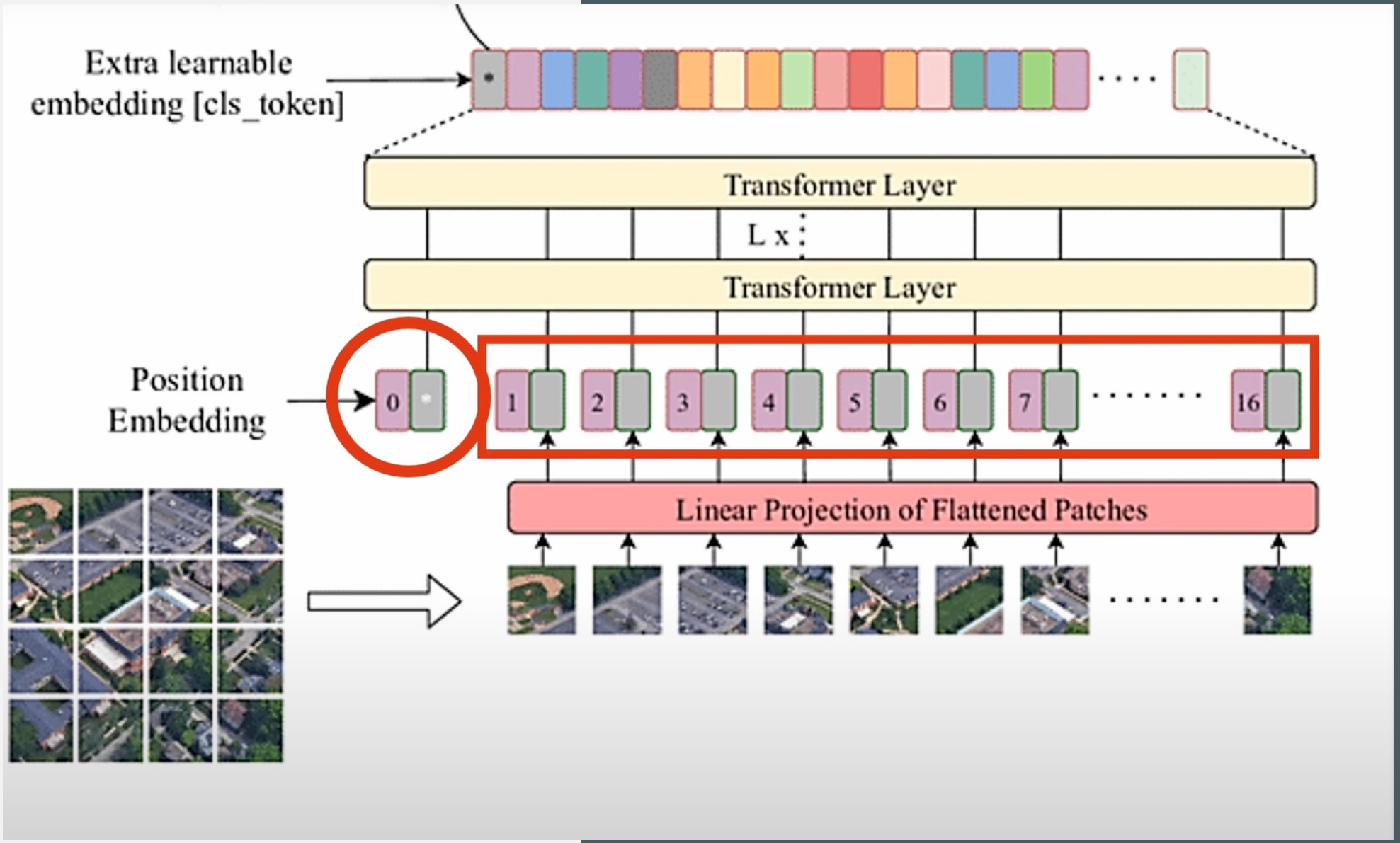
EMBEDDING OF IMAGES



- We "transform" an image into a sequence using patching.
- We create the embeddings using CNNs or FF layers.

THE CLS TOKEN

- The CLS token stands for the classification token. It will tell us what is the label of the image.
- As the transformer is a sequence to sequence model, we would have several outputs at the end, so the best way to get a single output is to take the output related to an additional token that we add at the beginning.


Extra learnable
embedding [cls_token]
Transformer Layer
L x
Transformer Layer
Position
Embedding
0
1
2
3
4
5
6
7
16
Linear Projection of Flattened Patches

POSITION EMBEDDINGS

- In vision transformers, we have a fixed size number of patches and therefore don't need to use the same positional embeddings like in NLP.
- In vision transformers the positional vectors are typically added in top of the input embedding vectors and not concatenated.
- In the implementation the positional embeddings are simply a learnable parameter vector per patch with a specific dimension.

```
class ViT(nn.Module):
    def __init__(self, ch=3, img_size=224, patch_size=16, emb_dim=128,
                 n_layers=5, out_dim=37, dropout=0.1):
        super(ViT, self).__init__()

        # Attributes
        self.channels = ch
        self.height = img_size
        self.width = img_size
        self.patch_size = patch_size
        self.n_layers = n_layers

        # Patching
        self.patch_embedding = PatchEmbedding(in_channels=ch,
                                              patch_size=patch_size,
                                              emb_size=emb_dim)

        # Learnable params
        num_patches = (img_size // patch_size) ** 2
        self.pos_embedding = nn.Parameter(
            torch.randn(1, num_patches + 1, emb_dim))
        self.cls_token = nn.Parameter(torch.rand(1, 1, emb_dim))
```

CNNS VS. VIT

CNNS

- Known for their compact size and efficient memory utilization, making them suitable for resource-constrained environments.

VISION TRANSFORMERS

- Capture global dependencies and contextual understanding in images, resulting in improved performance in certain tasks.
- Larger model sizes and higher memory requirements compared to CNNs.
- Capable of visualizing attention on pixels.