Transformers for physicists

Accelerating multi-messenger astronomy using air shower observations

Outline

- Why and when to use transformers?
- Transformers architecture
  - Top-down approach: General overview
  - Bottom-up approach: Details and explicit formulas
- Hands-on session

The need for transformers

Neural network are all about ***structure*** of the data to be analyzed and ***processing protocols***:
CNNs for images, RNNs for sequences, MLPs for features.

Transformers are designed to analyze data which:

- Consists of similar type elements
- Each element can be represented by a set of features
- *Correlation between elements are important*

For example:

- Natural language processing: sentences, especially long ones.
- Astrophysics: set of detectors triggered in an event.

Transformers are faster and better suited for such data due to the *attention mechanism*.

Transformers in physics

- Charged particle track reconstruction (at LHC and others)
- “Neutrinos in Deep Ice” Kaggle competition: reconstruction of events at IceCube
  (and other neutrino telescopes)
- Gamma-Ray Bursts classification
- Telescope Array (in development)

Top-down approach: General overview

General scheme

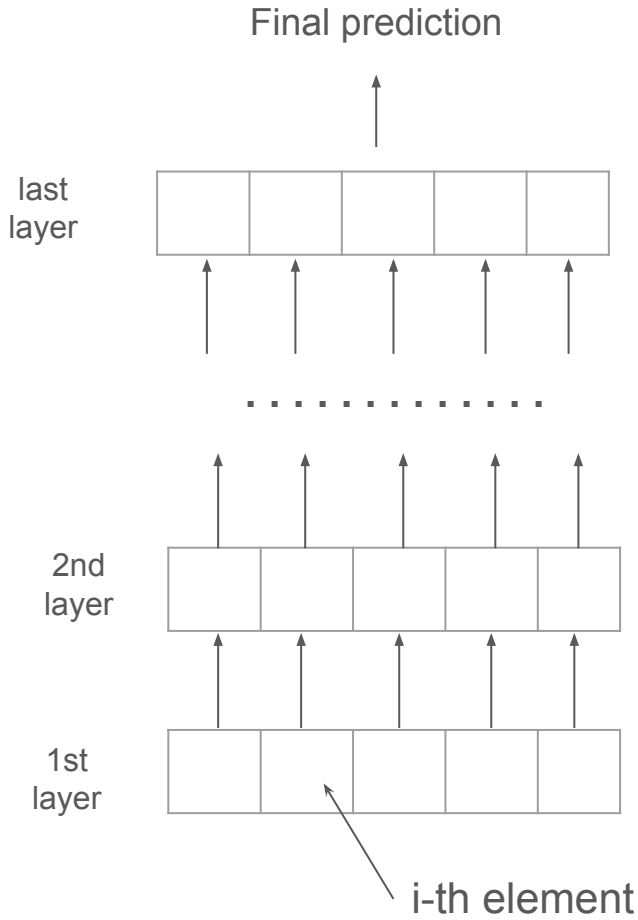
Data: set of elements (generally unordered)

Transformer consists of ***encoder layers***.

Each encoder layer ***updates*** data:
- Updates are ***element-wise***.
- For updating, relevant information is gathered *from all elements* via ***self-attention***.

Multiple encoder layers allows for extraction of *high level features.*

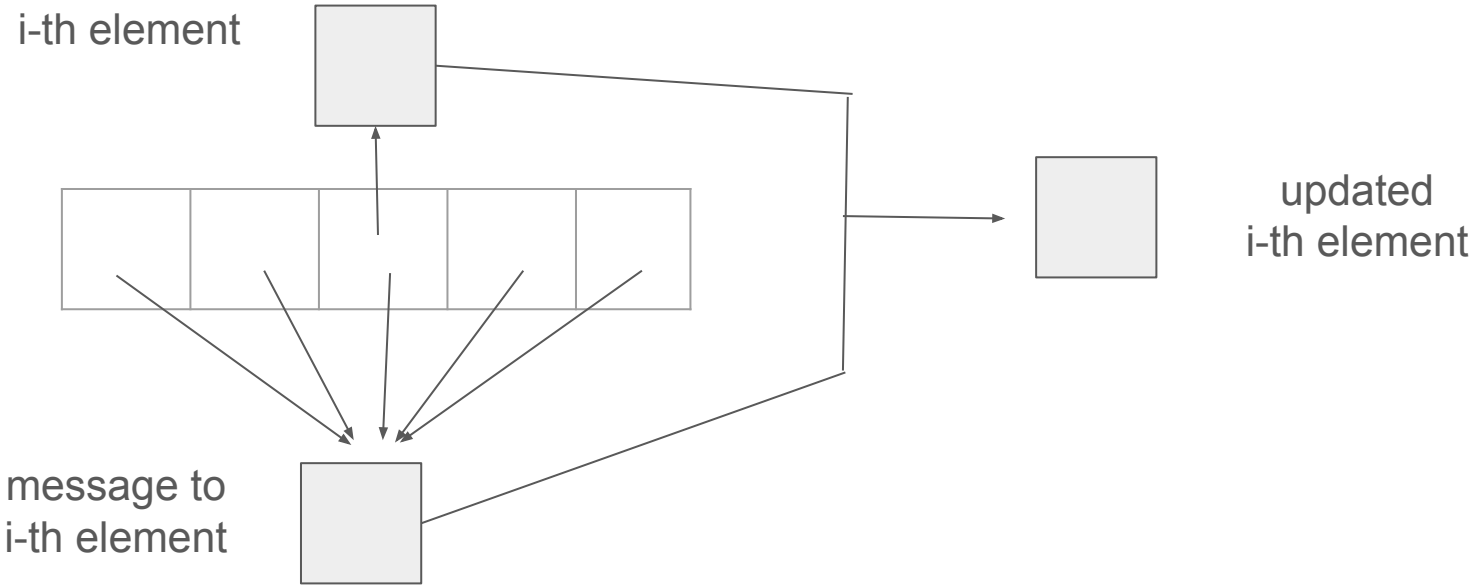
Last encoder layer can be used to make further predictions: element-wise or data-wise.

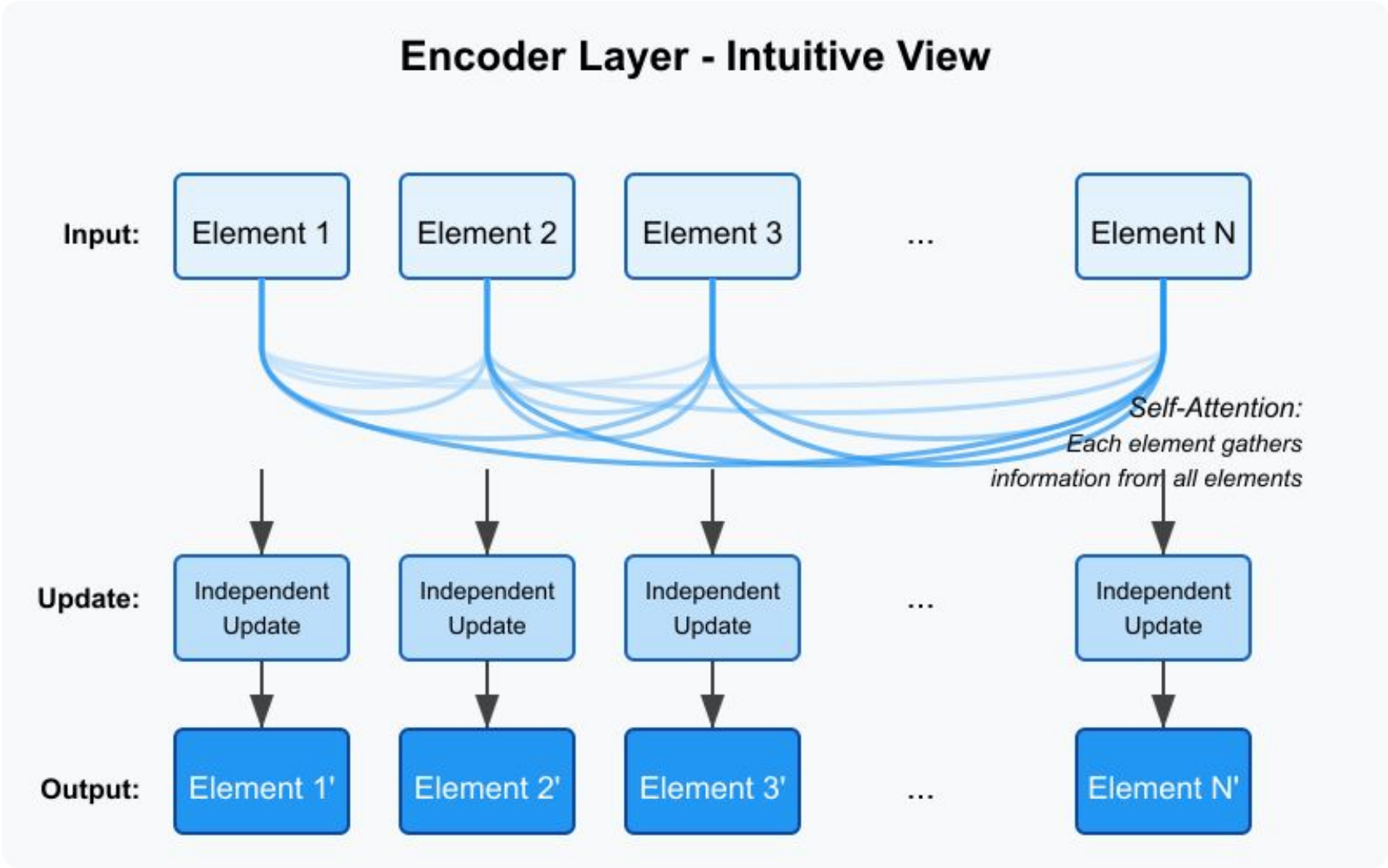


Encoder layer

Encoder updates data representation by updating each element independently.

To update an element one makes ***updating message***: the information gathered from all elements that will be used to update *element* representation.




Encoder Layer - Intuitive View
Input:
Element 1
Element 2
Element 3
...
Element N
Self-Attention:
Each element gathers
information from all elements
Update:
Independent Update
Independent Update
Independent Update
...
Independent Update
Output:
Element 1'
Element 2'
Element 3'
...
Element N'

Encoder layer

In general case, ***updating message*** is generated as:

$$m_i = g[\Theta](n),$$

where g[Θ] is *self-attention mechanism* and n stands for all data elements.

Further, ***the updated element value*** is obtained as:

$$n'_i = f[\Theta](n_i, m_i),$$

where f[Θ] is small neural network (MLP), and n_i is the i-th element.

Standard transformers choice:

$$n'_i = f[\Theta](n_i + m_i), \quad m_i = \sum_j a_{ji} (M[\Theta]n_j), \quad \sum_j a_{ij} = 1,$$

where a_{ij} are attention scores from *self-attention* and M[Θ] is a *learnable matrix*.

Attention layer

How to construct messages? We need to evaluate how relevant is one element to another.

Query, Key, and Value (QKV) concepts from the information retrieval systems:

Query *(Q)*: "what information am I looking for?"

Key *(K)*: "what information do I contain?"

Value *(V)*: "what is my actual content?"

Q, K, V are obtained for each element as linear projections:

$$Q_i = M[\Theta]_q n_i, \quad K_i = M[\Theta]_k n_i, \quad V_i = M[\Theta]_v n_i,$$

where $M[\Theta]_{\square}$ are dimension reduction matrices: dim(Q,K,V) < dim(n)

Why reducing dimensionality? Memory and general efficiency (this is enough)

Attention layer

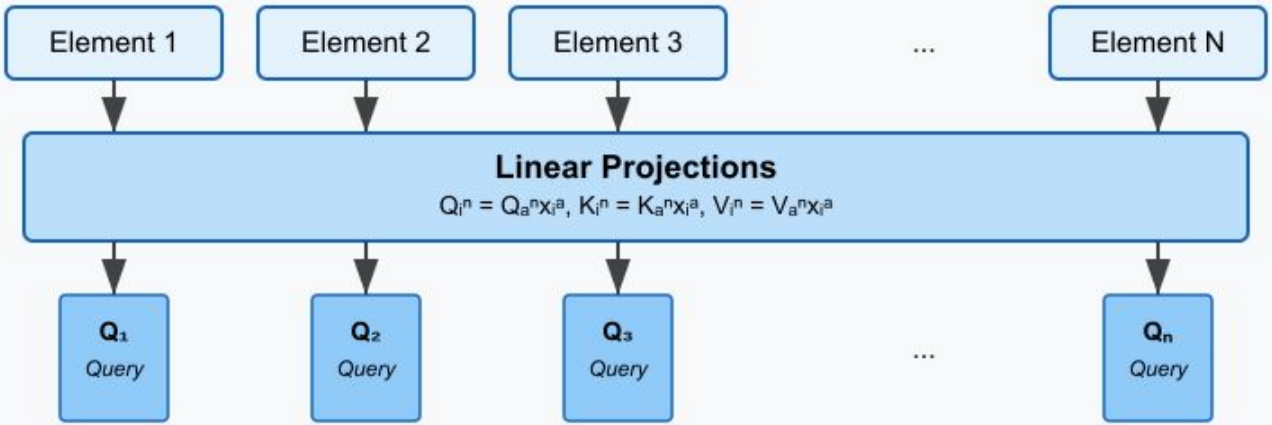
Using Query, Key, Value concepts, one can calculate attention scores:

$$score_{ij} = Q_i K_j^T, \quad weight_{ij} = softmax(score_{ij}),$$
$$softmax(a_k) = e^{(a_k)} / \sum_i e^{(a_i)}$$
(weight$_{ij}$ is a_{ij} defined previously)

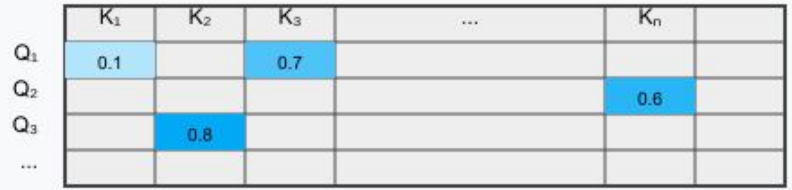
Then the message to element ***i*** is:

$$m_i = \sum_j a_{ji} V_j$$

Self-Attention Mechanism



Query 1



Attention Formula:

$$score_{ij} = Q_i^k K_j^k, \quad weight_{ij} = softmax(score_{ij})$$

$$m_i = \sum_j a_{ji} V_j$$

Multi-head Attention

Multi-head attention extends the basic attention mechanism by allowing the model to *jointly attend to information from* ***different representation subspaces***.

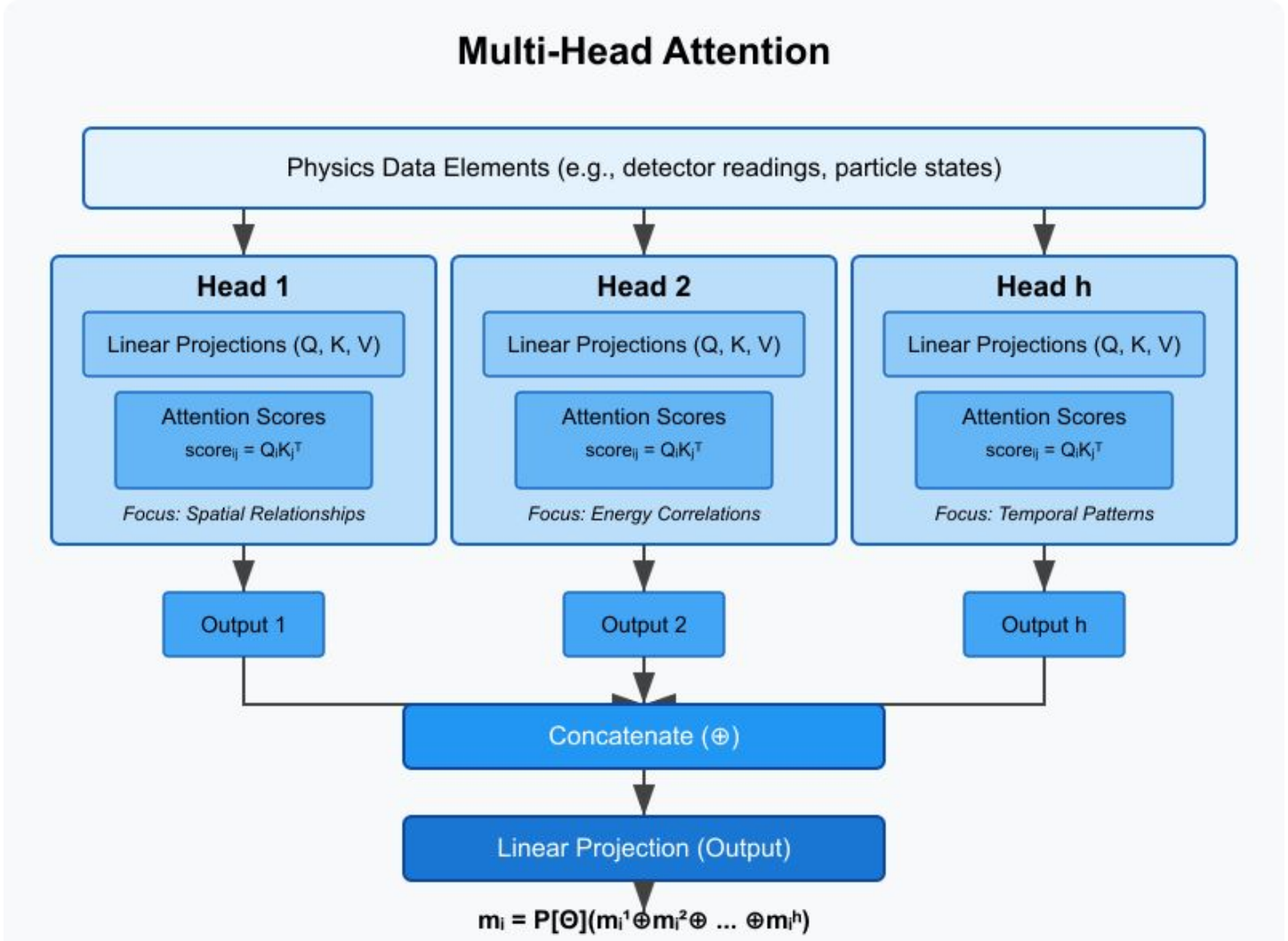
Think of multi-head attention as having multiple "perspectives" or "viewpoints" from which to analyze the relationships between detectors:

- Different Relationship Types: Some heads might focus on spatial relationships, others on timing patterns, others on energy deposition correlations.
- Complementary Information: Each head can capture different aspects of the detector relationships, and together they provide a more comprehensive understanding.

The **message** is then the projected concatenation of messages from all attention heads:

$$m_i = P[\Theta](m_i^1 \oplus m_i^2 \oplus \dots \oplus m_i^h)$$

P[Θ] should project sum of m's to the same dimensionality as that of elements.


Multi-Head Attention
Physics Data Elements (e.g., detector readings, particle states)
Head 1
Linear Projections (Q, K, V)
Attention Scores
$score_{ij} = Q_i K_j^T$
Focus: Spatial Relationships
Head 2
Linear Projections (Q, K, V)
Attention Scores
$score_{ij} = Q_i K_j^T$
Focus: Energy Correlations
Head h
Linear Projections (Q, K, V)
Attention Scores
$score_{ij} = Q_i K_j^T$
Focus: Temporal Patterns
Output 1
Output 2
Output h
Concatenate (⊕)
Linear Projection (Output)
$m_i = P[\Theta](m_i^1 \oplus m_i^2 \oplus \ldots \oplus m_i^h)$

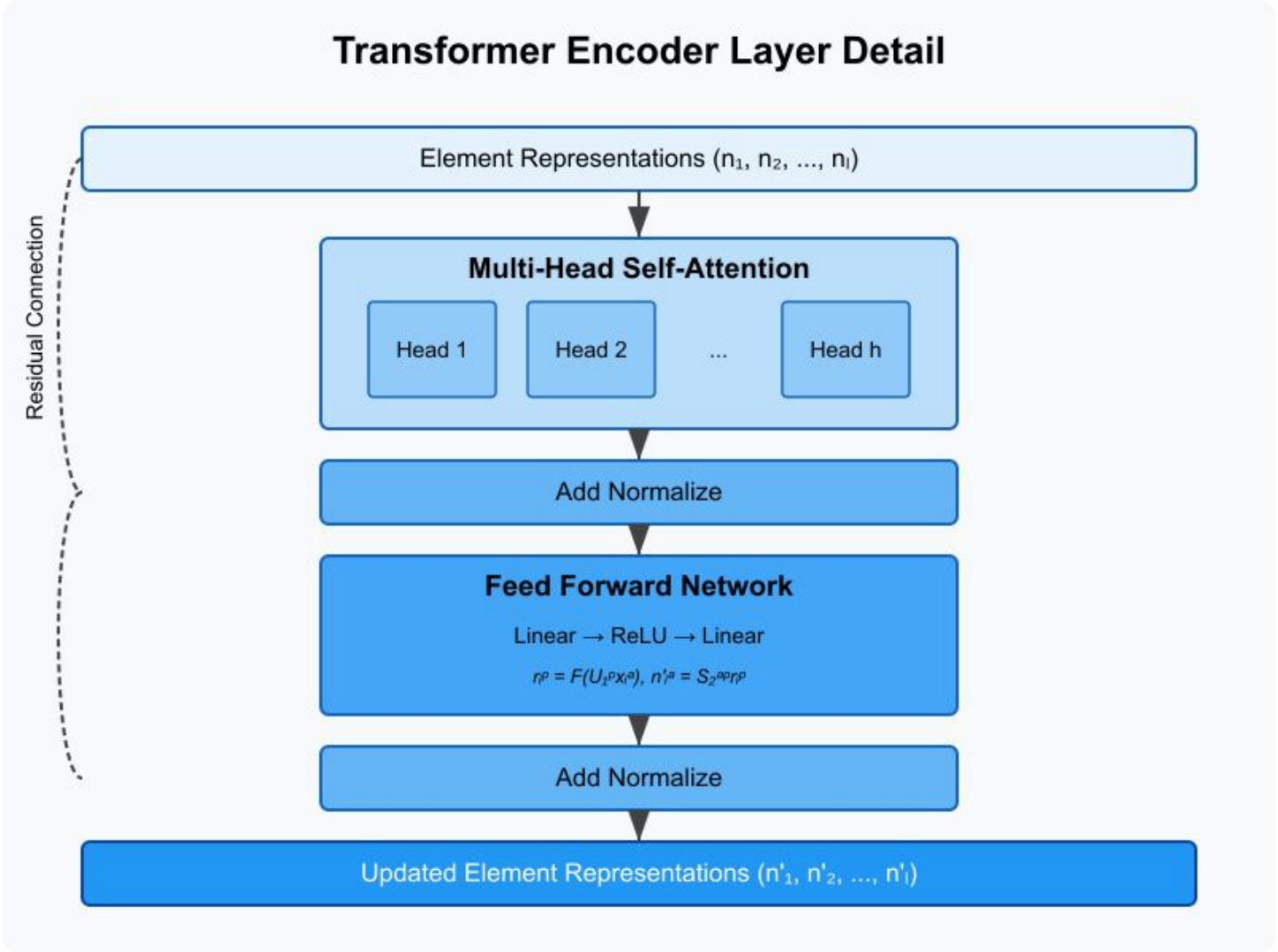
Feed Forward Layer

Finally, one needs to ***update elements values n'$_i$***. Typically, this is done by combining two linear layers with intermediate non-linear activation function $f(\cdot)$:

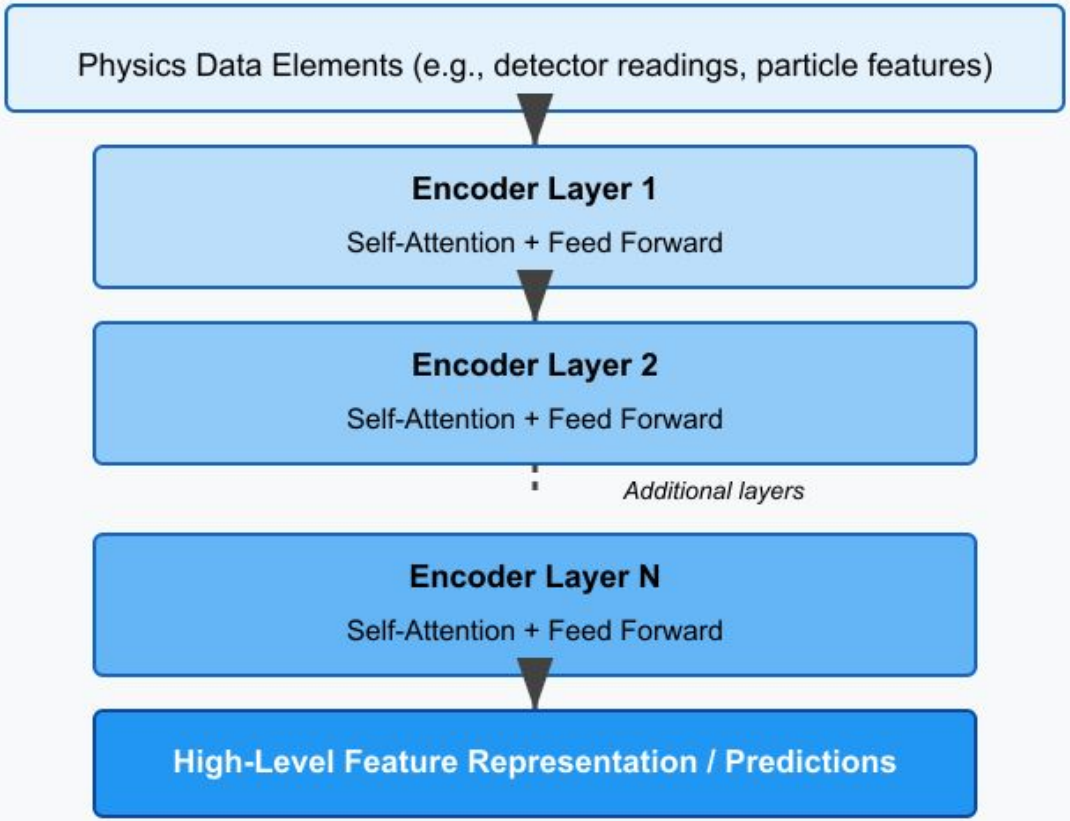
$$r_i = f(M_1[\Theta](n_i+m_i))$$
$$n'_i = M_2[\Theta](r_i)$$

M_1 projects to higher dimensionality space that that of n.

By expanding to a higher dimension and then projecting back (via $M_2[\Theta]$), one increases the model's representational capacity.


Transformer Encoder Layer Detail
Element Representations (n_1, n_2, ..., n_l)
Residual Connection
Multi-Head Self-Attention
Head 1
Head 2
...
Head h
Add Normalize
Feed Forward Network
Linear → ReLU → Linear
$r_i^p = F(U_1^p x_i^a)$, $n'^a_i = S_2^{ap} r_i^p$
Add Normalize
Updated Element Representations (n'_1, n'_2, ..., n'_l)

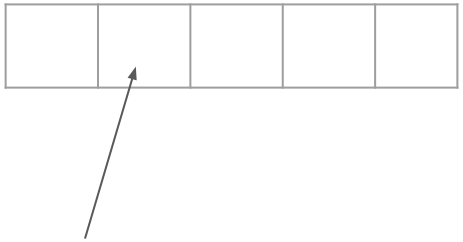
Transformer Architecture



This is it!
One should stack multiple encoder layers to get better analysis quality.

Let’s do the same but from bottom-up perspective: explicitly trace how each element is updated.

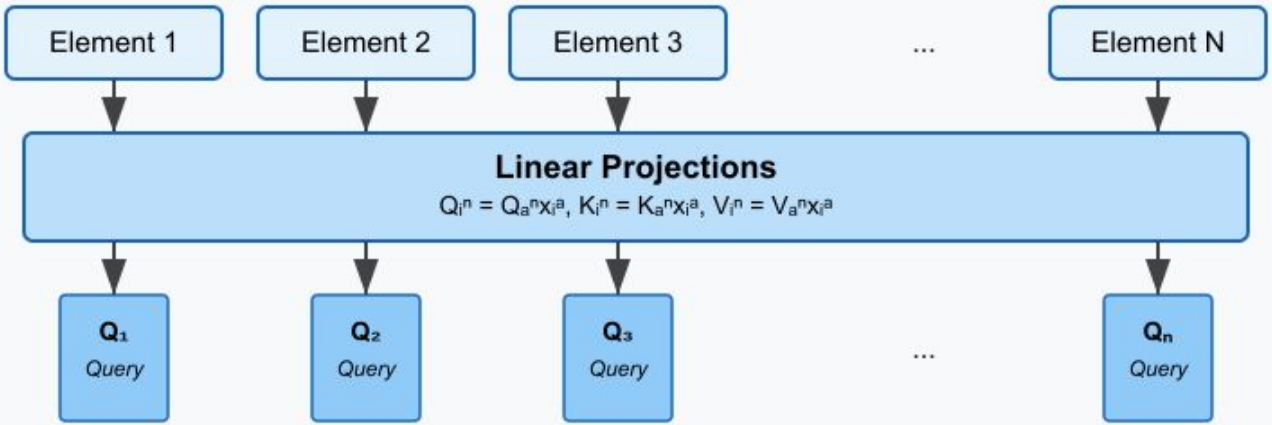
Notation



Data element: (l,f)

x_i^a:
i - “position” index (1 .. l),
a - feature index (1 .. f)

Self-Attention Mechanism



Query 1

	K_1	K_2	K_3	...	K_n	
Q_1	0.1		0.7			
Q_2					0.6	
Q_3		0.8				
...						

Attention Formula:

$$score_{ij} = Q_i^k K_j^k, \quad weight_{ij} = softmax(score_{ij})$$

$$m_i = \sum_j a_{ji} V_j$$

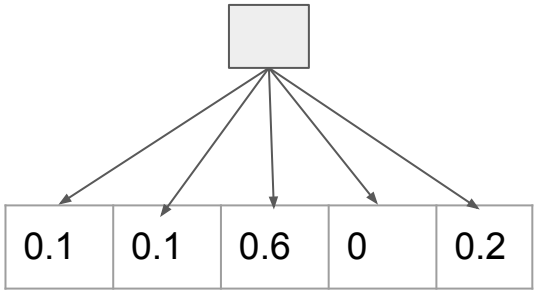
1. Attention layer

1. Project to Query, Key, Value for self-attention: ($n \in 1..d_a$)

$$q_i^n = Q^n{}_a x_i^a, \qquad k_i^n = K^n{}_a x_i^a \qquad v_i^n = V^n{}_a x_i^a$$

2. Calculate attention scores:

$$w_{ij} = \text{softmax}(q_i^n k_j^n)$$



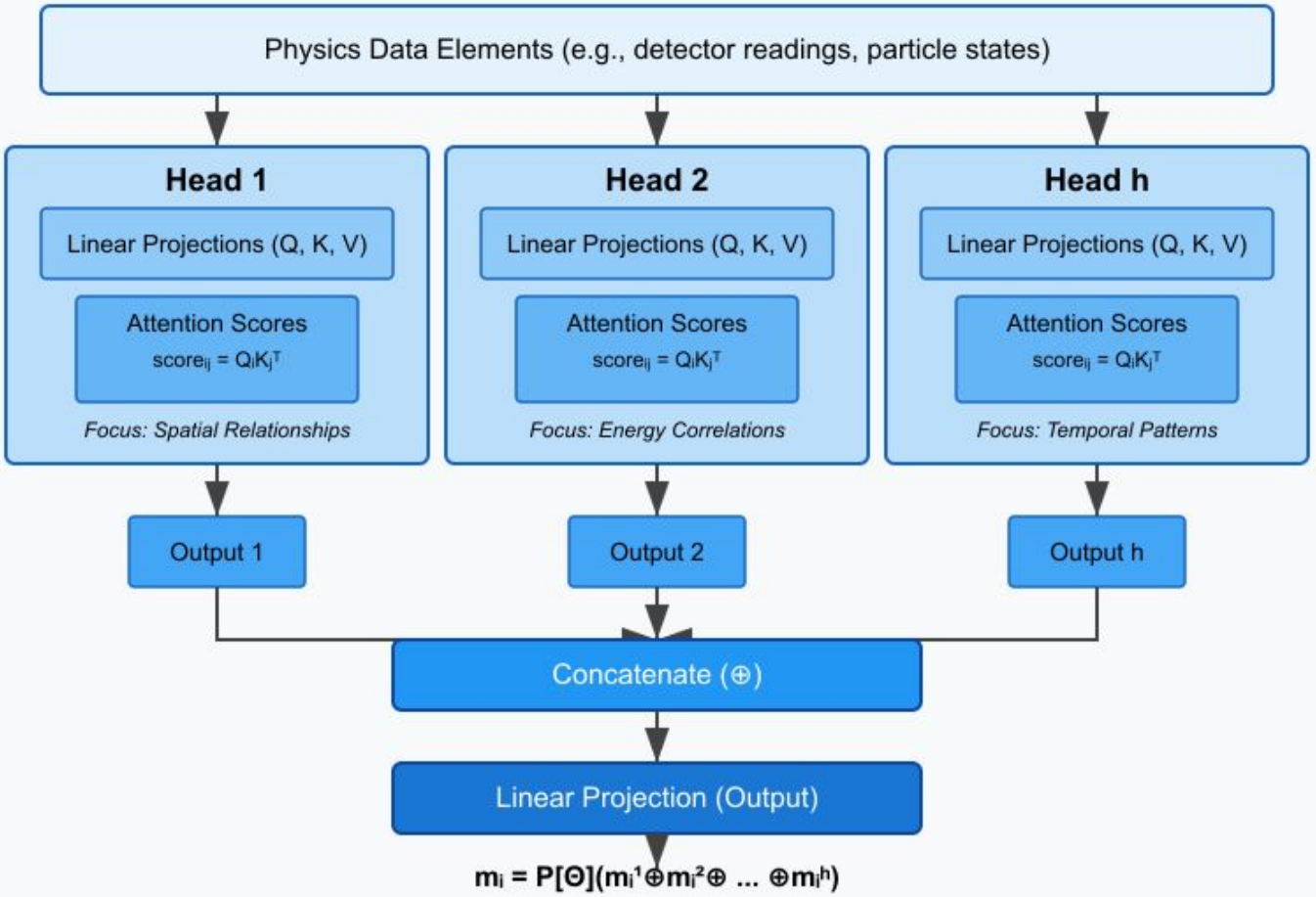
3. Generate messages:

$$m_i^n = w_{ij} v_j^n$$

$$m = 0.1v_1 + 0.1v_2 + 0.6v_3 + 0.2v_5$$

Exercise: write this down for general case (Q, K, V originate from different data)

Multi-Head Attention



2. Multi-head attention

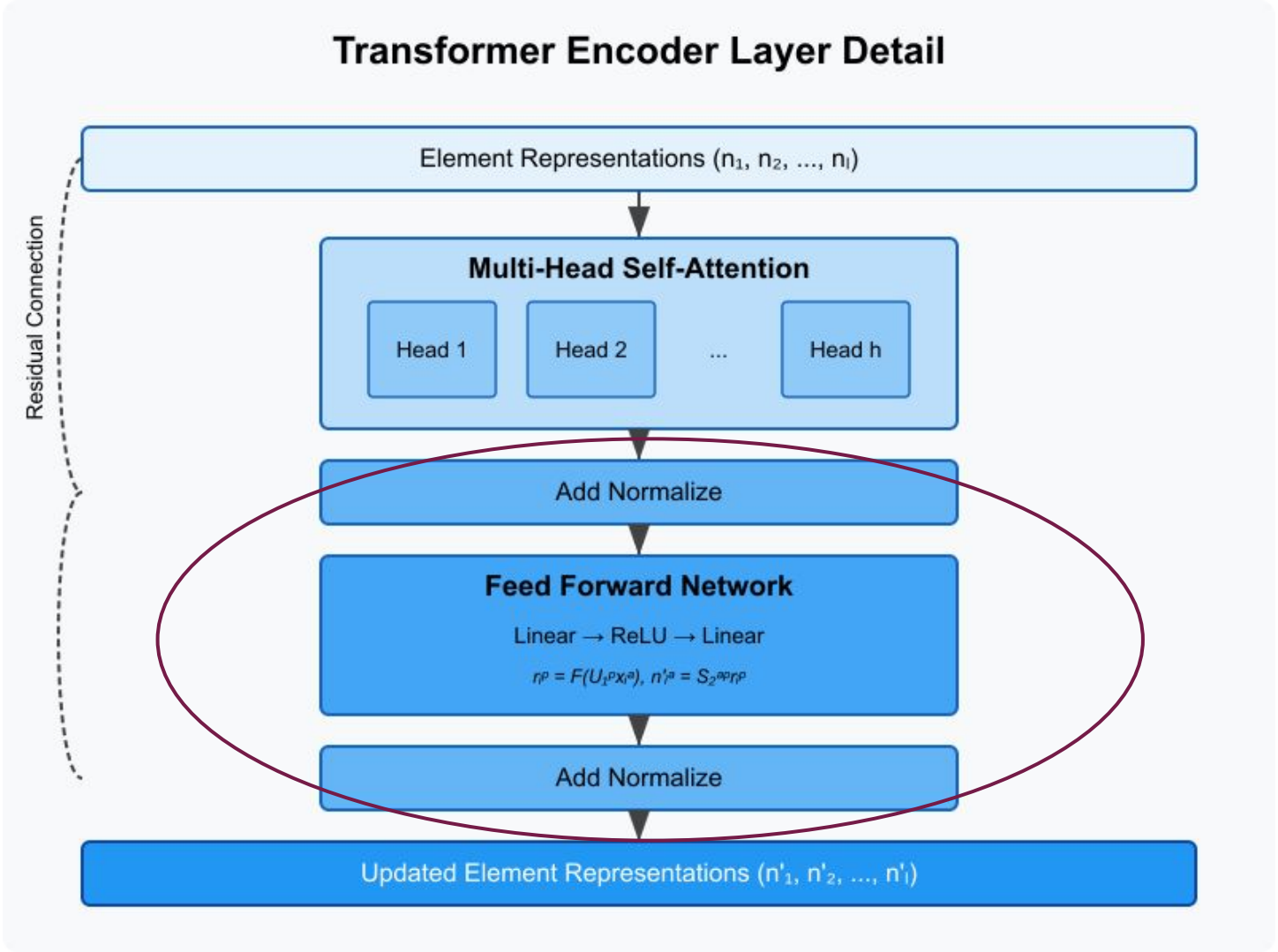
We have k different messages: $m_i^{nh} = m_i^1 \oplus \ldots \oplus m_i^h, \quad (h \in 1..k)$

Consider (nh) as one index p: m_i^p

We need to project back to n dimensions:

$$m'^n_i = M_p^n m_i^p$$

m'^n_i will be used to update element value: x_i^n


Transformer Encoder Layer Detail
Element Representations (n_1, n_2, ..., n_l)
Residual Connection
Multi-Head Self-Attention
Head 1
Head 2
...
Head h
Add Normalize
Feed Forward Network
Linear → ReLU → Linear
$r_i^p = F(U_1^p x_i^a)$, $n'^a_i = S_2^{ap} r_i^p$
Add Normalize
Updated Element Representations (n'_1, n'_2, ..., n'_l)

3. Intermediate update

1. Sum message and node value as an update:
$$x'^{n}_{i} = x^{n}_{i} + m'^{n}_{i}$$

2. Apply Layer normalization for better gradient propagation:
$$x'^{n}_{i} = LN(x'^{n}_{i})$$

Layer normalization: $\text{Mean}(x'^{n}_{i}) = 0, \quad \text{Std}(x'^{n}_{i}) = 1$
(Unlike Batch normalization, Layer normalization works data-wise)

Optionally, apply dropout for regularization.

N.B. one may use different updating protocol

4. Feed forward layer

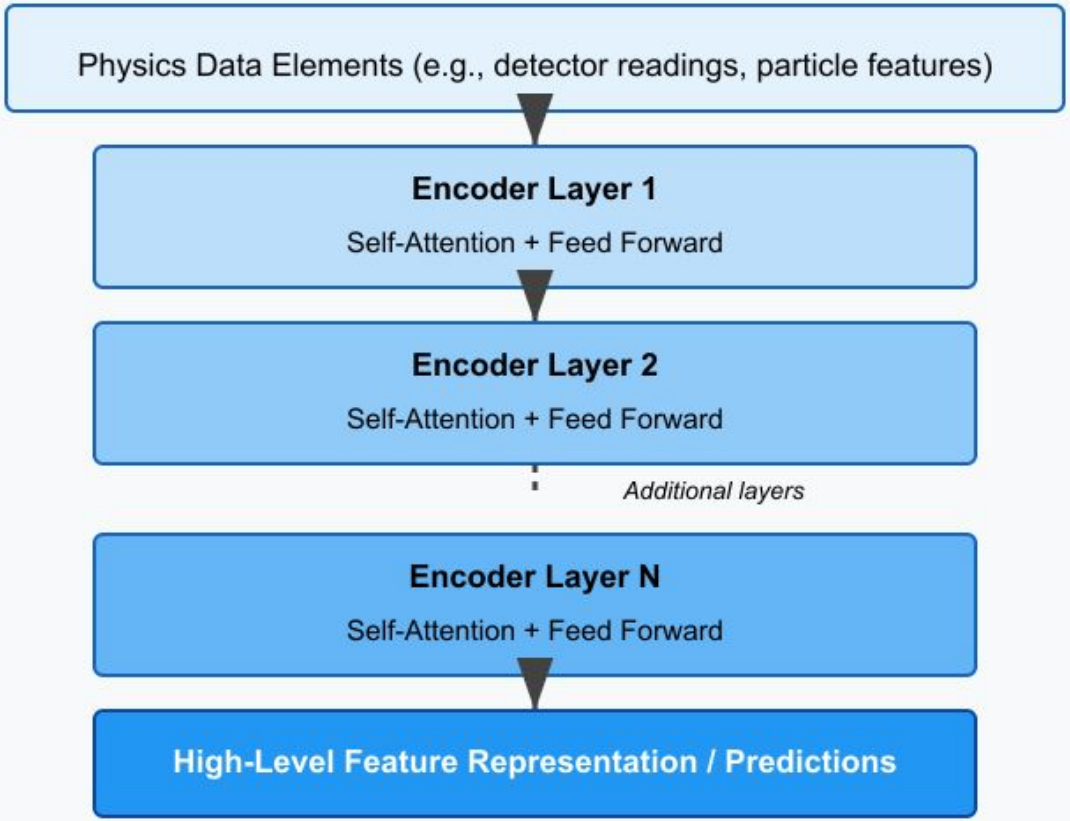
To enhance representation abilities, we first extend representation space, and only then apply non-linearity and reduce to transformer dimensionality n:

$$r_i^k = F(U_a{}^k x_i^a), \quad (k \in 1..d_f, \ d_f > n)$$
$$x'^a_i = S_k{}^a r_i^k$$

This process is done in parallel for all elements.
Thanks to this parallelization and ***self-attention*** transformers are so powerful.

N.B. one may use different updating protocol

Transformer Architecture



Masking

Sometimes we want to ***prohibit*** elements from paying attention to other elements
(for example: due to causality, padding)

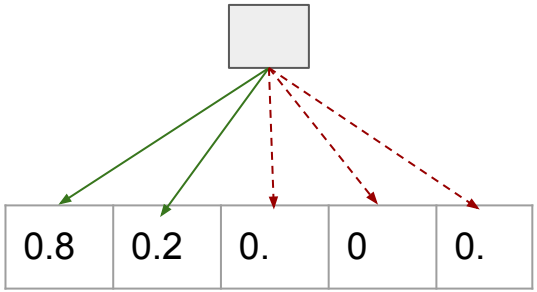
One can mask the corresponding elements by using a ***mask array*** of shape (length, length):

$mask_{ij}$ = 1 for possible connections,

$mask_{ij}$ = 0 otherwise

(native PyTorch conventions are opposite)

To implement this mechanism, one should set attention scores to negative infinity for the masked elements
(see hand on session notebook for technical details)



Attention Masking Mechanism

Attention Score Matrix

Raw QK^T Scores

	E1	E2	E3	E4	E5
E1	0.8	0.3	0.2	0.1	0.5
E2	0.1	0.9	0.4	0.3	0.2
E3	0.3	0.2	0.7	0.2	0.1
E4	0.2	0.1	0.4	0.8	0.3
E5	0.4	0.2	0.3	0.1	0.9

$\odot$

Attention Mask

	E1	E2	E3	E4	E5
E1	1	1	1	1	0
E2	1	1	0	0	0
E3	1	1	1	1	1
E4	1	1	1	0	0
E5	1	1	0	0	0

Masked Attention Scores

	E1	E2	E3	E4	E5
E1	0.8	0.3	0.2	0.1	$-\infty$
E2	0.1	0.9	$-\infty$	$-\infty$	$-\infty$
E3	0.3	0.2	0.7	0.2	0.1
E4	0.2	0.1	0.4	$-\infty$	$-\infty$
E5	0.4	0.2	$-\infty$	$-\infty$	$-\infty$

$mask_{ij}$ = 1 for allowed connections, $mask_{ij}$ = 0 for masked connections (variable-length sequences)

Embedding layer

Number of the initial features (detectors features) is usually low.

To enhance representation capabilities, one uses embedding layer:
It project initial features to higher dimensional space.

The transformer works with embedded features.

$$x_i^a = N_b^a y_i^b, \quad \dim(a)>\dim(b)$$

where y_i^b are initial features and N_b^a is an embedding matrix.

Positional encodings

Originally, Transformers were used to analyze sentences.

In transformers, words are characterized by high dimensionality vectors (space of words “meaning”).

However, words in a sentence lose their “position” information.
To get it back, positional encoding is required.

For astrophysical data, we already have “positional” information:
detectors activation times or locations.

Thus we do not need it.