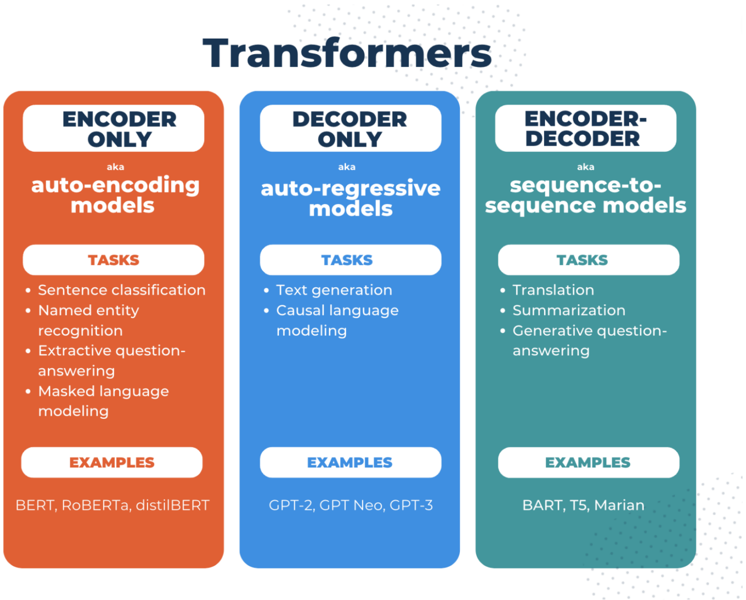
Transformers – Part 2

Summary of Chapter 10 from
Speech and Language Processing,
Jurafsky and Martin, Aug. 20, 2024 draft
Michael Wollowski

Transformers



Transformers

ENCODER ONLY
aka
auto-encoding models

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

EXAMPLES
BERT, RoBERTa, distilBERT

DECODER ONLY
aka
auto-regressive models

TASKS
- Text generation
- Causal language modeling

EXAMPLES
GPT-2, GPT Neo, GPT-3

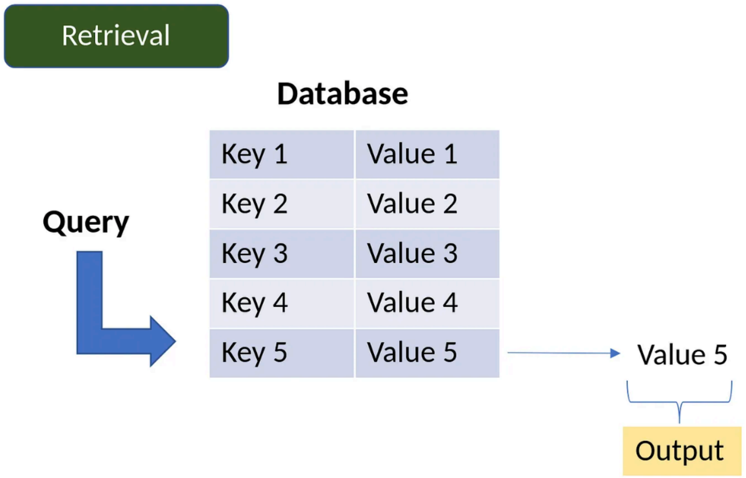
ENCODER-DECODER
aka
sequence-to-sequence models

TASKS
- Translation
- Summarization
- Generative question-answering

EXAMPLES
BART, T5, Marian

Source: https://www.comet.com/site/blog/explainable-ai-for-transformers/

Review of Databases



Retrieval

Database

Key 1	Value 1
Key 2	Value 2
Key 3	Value 3
Key 4	Value 4
Key 5	Value 5

Query

Value 5

Output

Image source: Arjun Sakar: All you need to know about ‘Attention’ and ‘Transformers’ - In-depth Understanding — Part 1

Reminder: Dot Product

- Definition:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

- Outcome: Can be used to calculate the similarity between two vectors.

“Database retrieval” in Transformers

Starting at the bottom, we calculate the dot product between:

X_3 and X_3

X_3 and X_2

X_3 and X_1

This should give us a similarity of the prior tokens to X_3

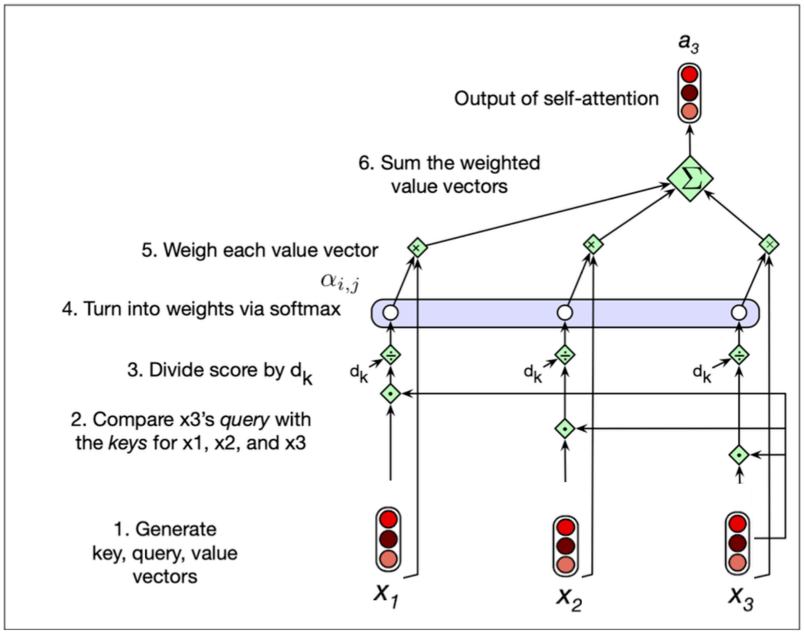

a_3
Output of self-attention
6. Sum the weighted value vectors
5. Weigh each value vector
$\alpha_{i,j}$
4. Turn into weights via softmax
3. Divide score by d_k
d_k
d_k
d_k
2. Compare x3’s *query* with the *keys* for x1, x2, and x3
1. Generate key, query, value vectors
x_1
x_2
x_3

Image source: Modified from Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

“Database retrieval” in Transformers

Next, we normalize the values of the dot product.

Otherwise the values may be quite large and impede training, due to loss of gradients.

We divide by the square root of the dimensionality of the key vector, d_k

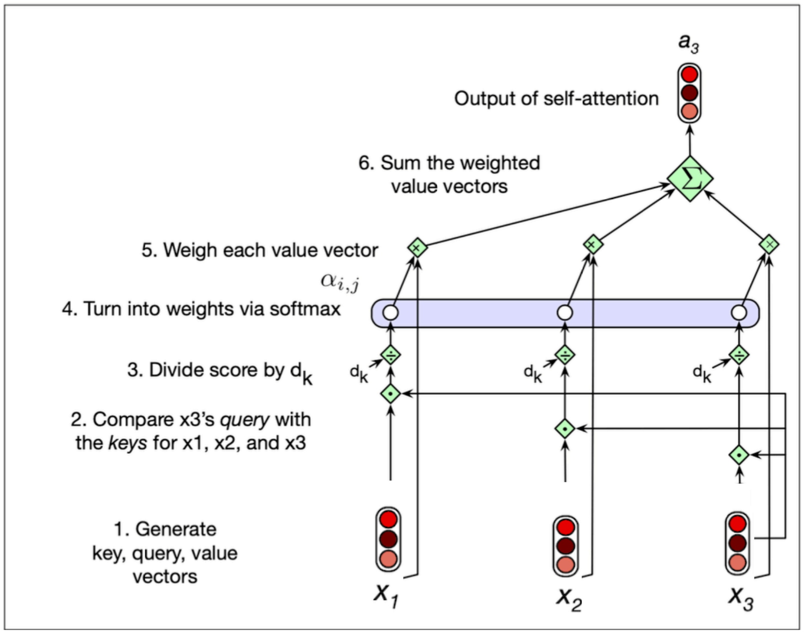

a_3
Output of self-attention
6. Sum the weighted value vectors
5. Weigh each value vector
$\alpha_{i,j}$
4. Turn into weights via softmax
3. Divide score by d_k
d_k
d_k
d_k
2. Compare x3’s *query* with the *keys* for x1, x2, and x3
1. Generate key, query, value vectors
x_1
x_2
x_3

Image source: Modified from Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

“Database retrieval” in Transformers

Now, we run the values through softmax to obtain weights.

These weights are now used to determine the relevance of each of X_1 to X_3

Finally, we sum up those weights which gives the output.

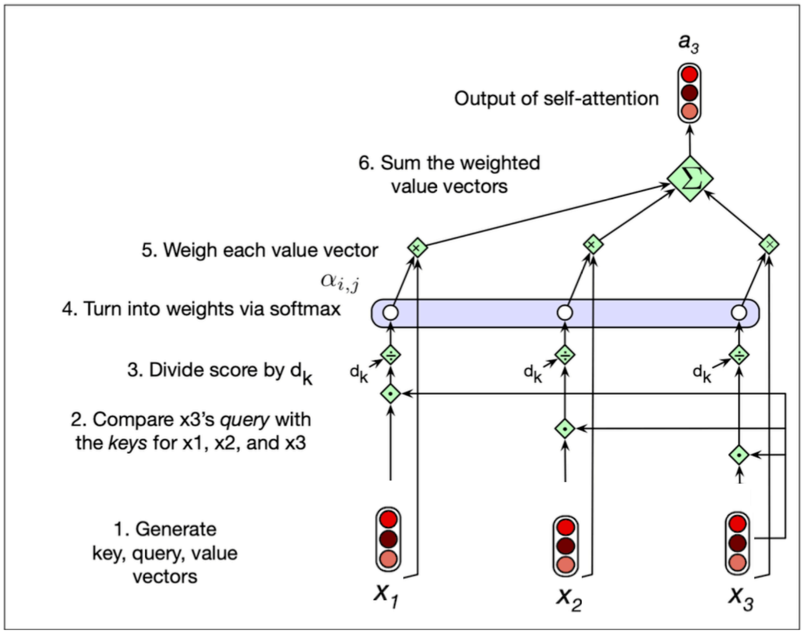

a_3
Output of self-attention
6. Sum the weighted value vectors
Σ
5. Weigh each value vector
$\alpha_{i,j}$
4. Turn into weights via softmax
3. Divide score by d_k
d_k
d_k
d_k
2. Compare x3’s *query* with the *keys* for x1, x2, and x3
1. Generate key, query, value vectors
x_1
x_2
x_3

Image source: Modified from Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

Reminder: Softmax

- Definition:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

- The softmax function takes as input a vector and:
  - turns each component into an interval _(0,1)
  - the components will add up to 1,
  - they can be interpreted as probabilities

“Database retrieval” in Transformers

We should add some weights.

After all that is what NNs are all about.

More seriously, it enables us to move away from a very static distance measure to one that can be trained, based on a token’s context.

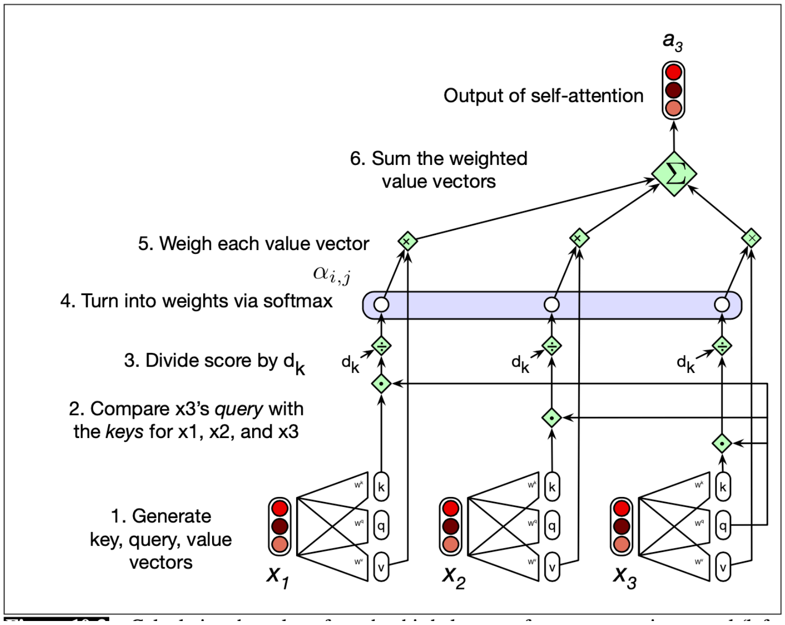

a_3
Output of self-attention
6. Sum the weighted value vectors
Σ
5. Weigh each value vector
$\alpha_{i,j}$
4. Turn into weights via softmax
3. Divide score by d_k
d_k
d_k
d_k
2. Compare x3’s *query* with the *keys* for x1, x2, and x3
1. Generate key, query, value vectors
k
q
v
x_1
x_2
x_3

Image source: Speech and Language Processing, Jurafsky and Martin, Feb. 3, 2024 draft

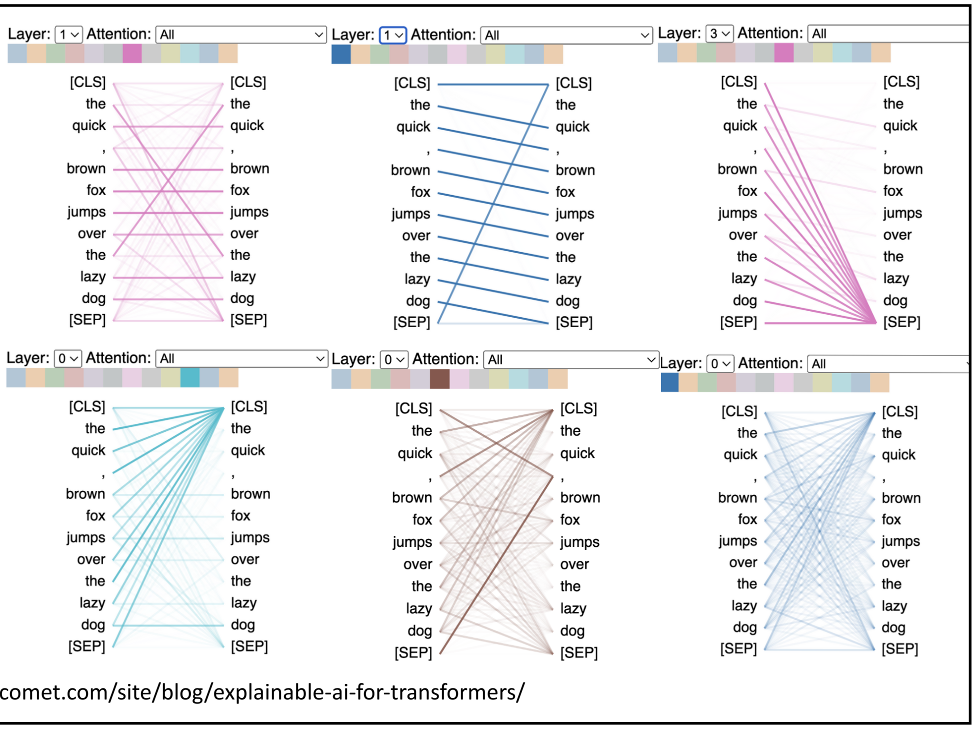
Multi-head Attention

- In the context of:

- We will add more than one attention head!
- More attention heads, more weights, more things to pay attention to.
- Recall that for CNNs, we applied several filters to a matrix, to “look” at different aspects of an image.

Image source: https://muppet.fandom.com/wiki/Me_Want_Cookie?file=MeWantCookie.jpg

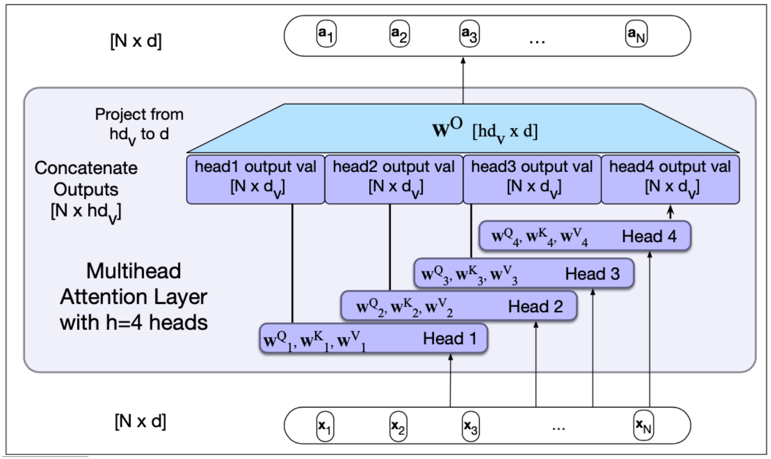
BertViz shows that attention captures various patterns in language, including positional patterns, delimiter patterns, and bag-of-words.

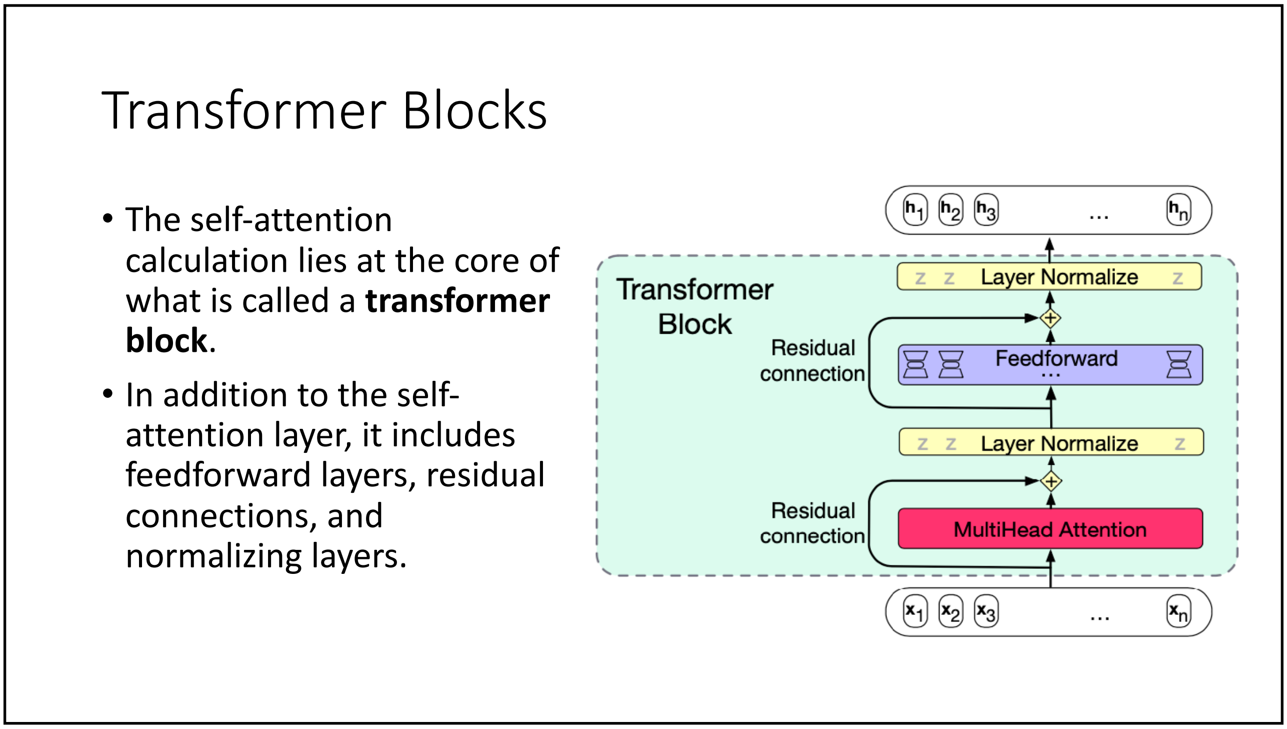


Source: https://www.comet.com/site/blog/explainable-ai-for-transformers/

Multi-head Attention

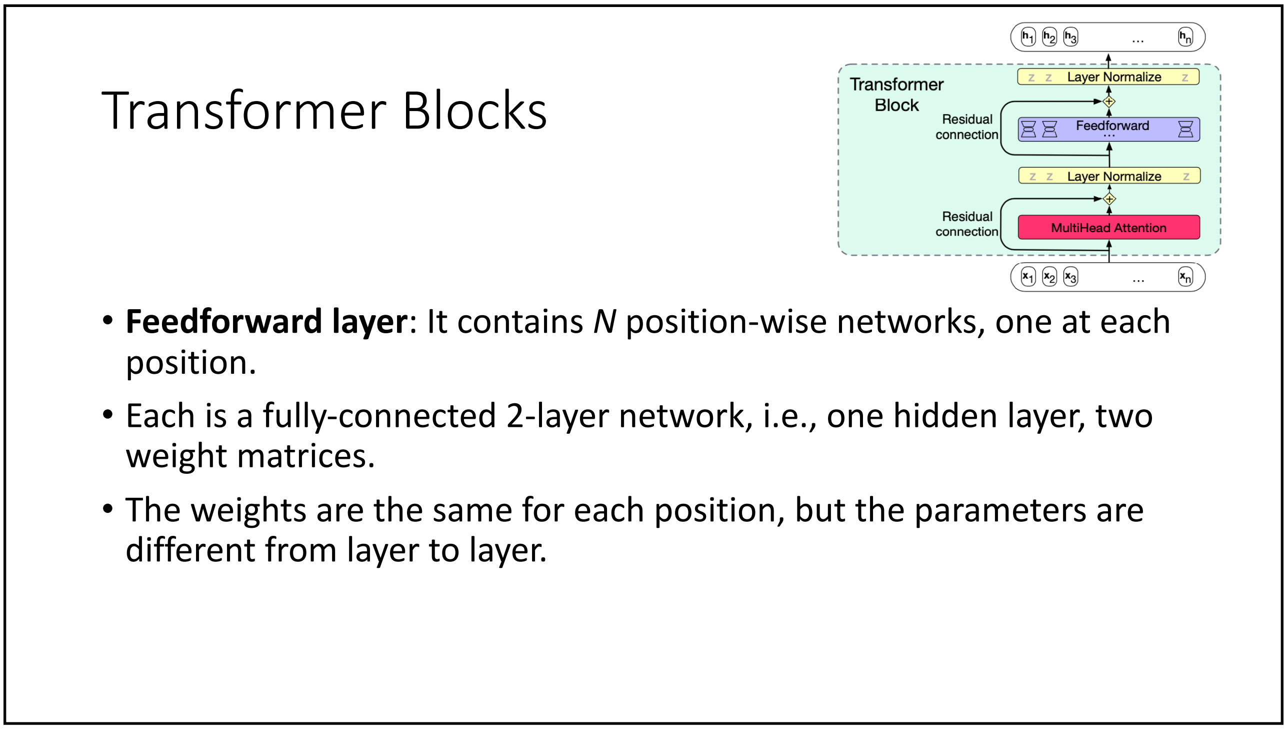
- Each of the multi-head self-attention layers is provided with its own set of key, query and value weight matrices.
- The outputs from each of the layers are concatenated.
- They are then projected to *d*.
- Thus producing an output of the same size as the input.


[N x d]
a_1
a_2
a_3
…
a_N
Project from hd_v to d
W^O $[hd_v \times d]$
Concatenate Outputs $[N \times hd_v]$
head1 output val $[N \times d_v]$
head2 output val $[N \times d_v]$
head3 output val $[N \times d_v]$
head4 output val $[N \times d_v]$
W^Q_4, W^K_4, W^V_4 Head 4
W^Q_3, W^K_3, W^V_3 Head 3
W^Q_2, W^K_2, W^V_2 Head 2
W^Q_1, W^K_1, W^V_1 Head 1
Multihead Attention Layer with h=4 heads
[N x d]
x_1
x_2
x_3
…
x_N



Transformer Blocks

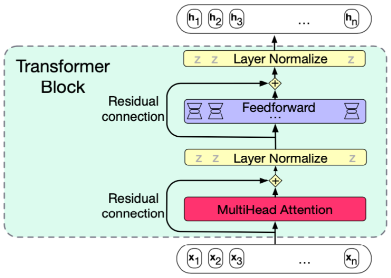
- The self-attention calculation lies at the core of what is called a **transformer block**.
- In addition to the self-attention layer, it includes feedforward layers, residual connections, and normalizing layers.



Transformer Blocks

- **Feedforward layer**: It contains *N* position-wise networks, one at each position.
- Each is a fully-connected 2-layer network, i.e., one hidden layer, two weight matrices.
- The weights are the same for each position, but the parameters are different from layer to layer.

Transformer Blocks



- **Residual connections**: They pass information from a lower layer to a higher layer without going through the intermediate layer.
- **Layer normalization (Layer norm)**. Summed vectors are normalized.
- It is used to improve training performance in deep neural networks.
- It keeps the values of a hidden layer in a range that facilitates gradient-based training.
- Layer norm z-score, from statistics applied to a single vector in a hidden layer.

Transformer Block: Layer Normalization

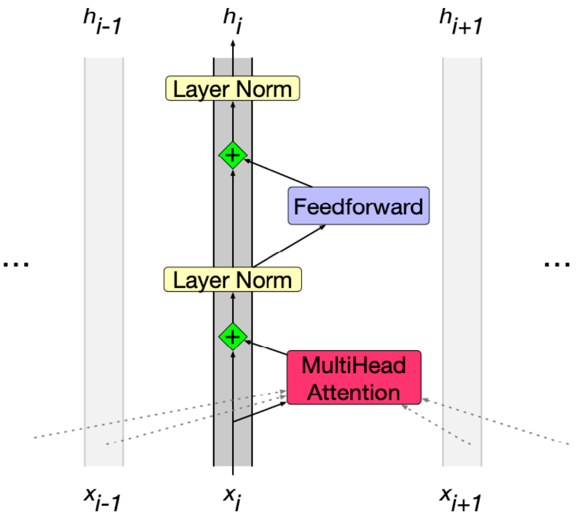
- Typical: z-score

$$z = \frac{x - \mu}{\sigma}$$

- Z = standard score
- X = observed value
- μ = mean of sample
- σ = standard deviation of the sample

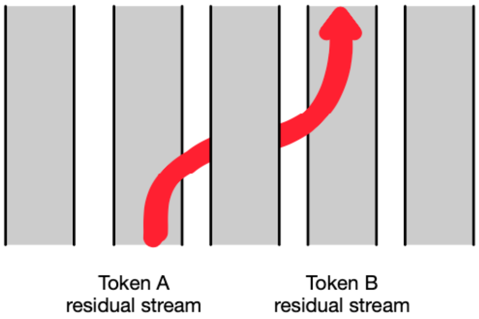
Residual Stream View of the Transformer

- An alternate view of a transformer is to trace the processing of an individual token vector x_i.
- The output of the feedforward and multi-head attention layers are added to x_i.
- The Multi Head Attention component reads information from the other residual streams in the context.



Moving Information

- As such, the attention head can move information from token A’s residual stream into token B’s residual stream.

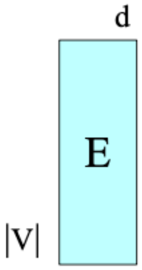


Embeddings and Such

- A token embedding is a vector of dimension *d* that will be the initial representation for the input token.
- As the vector is passed up through the transformer layers in the residual stream, this embedding representation will change and grow, incorporating context and playing a different role depending on the kind of language model we are building

Embeddings and Such

- Given an input token string like *"thanks for all the"* the transformer architecture first convert the tokens into vocabulary indices.
- Let V be the vocabulary and |V| be the size of V.
- Let E be the embedding matrix.
- The representation of "*thanks for all the"* might be w = [5, 4000, 10532, 2224].
- We treat the values of w as indices to corresponding rows from E, (row 5, row 4000, row 10532, row 2224).

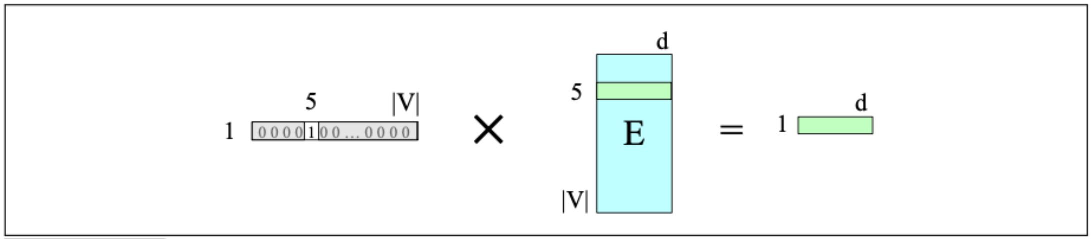


One hot-hot-hot vector

- In a *one-hot* vector all the elements are 0 except for one, the element whose dimension is the word's index in the vocabulary.
- If the word "thanks" has index 5 in the vocabulary, then x_5 =1, and all other x_i =0

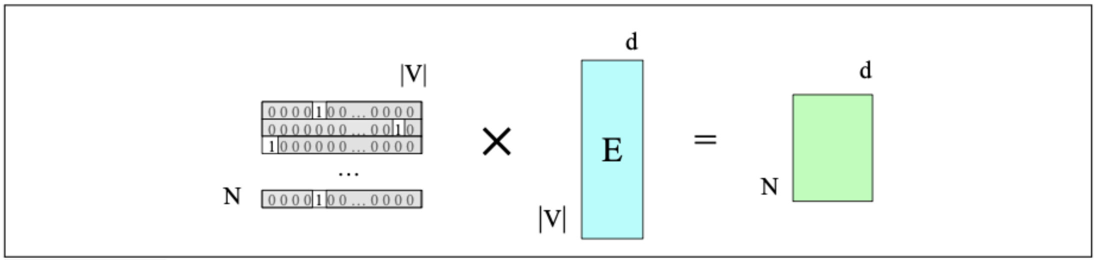
```
[0 0 0 0 1 0 0 ... 0 0 0 0]
 1 2 3 4 5 6 7 ...   ... |V|
```

Selecting the token embedding



Multiplying **E** by a one-hot vector that has only one non-zero element x_i = 1 simply selects the relevant row vector for word *i*, resulting in the embedding for word *i*.

Storing all of the N input tokens



To represent the entire token sequence, we multiply all **N** one-hot vectors with **E**.

Positions

- While the order in which the N tokens are inserted represents word order, this is not sufficient.
- Recall that attention heads can move tokens around.
- We wish to associate with each word the order in which it appeared in the text.
- We will create position embeddings.
- For example, just as we have an embedding for the word *fish*, we will have an embedding for position 3.

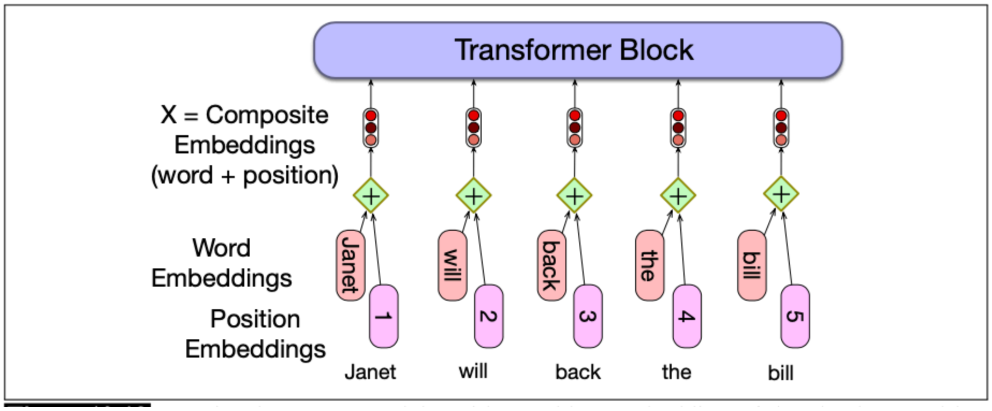
Positions

- The positions are absolute.
- We do not simply use integers.
- To be consistent, we use vectors of the same dimensionality as the token embeddings
- The vectors are initialized with random numbers.

Positions

- The positional embeddings are learned along with other parameters during training.
- Recall that token embeddings were learned at some point in time.
- To produce an input embedding that captures positional information, we just add the word embedding for each input to its corresponding positional embedding.

Positions



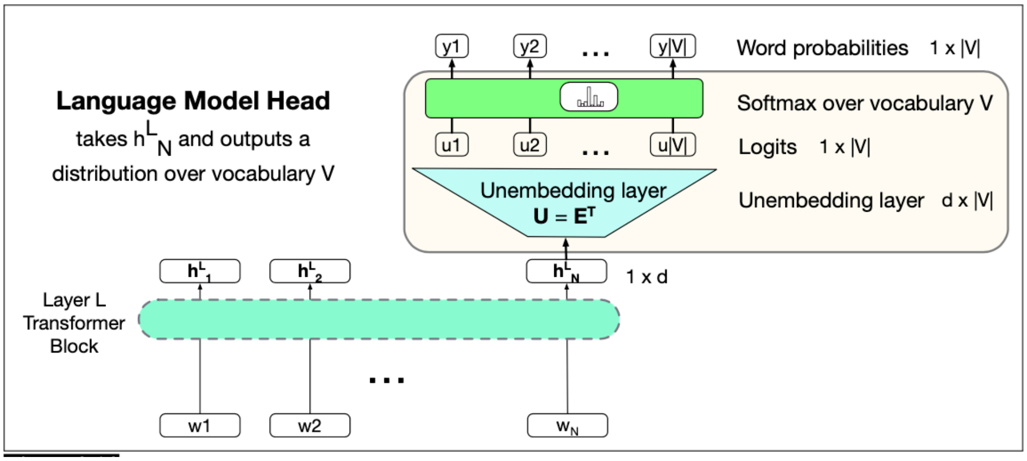
Combining word embeddings with positions.

The Language Modelling Head

- Language models are word predictors.
- For example, if the preceding context is "Thanks for all the" and we want to know how likely the next word is "fish" we would compute:

  P(*fish*|*Thanks for all the*)
- The language modeling head takes the output of the final transformer layer.
- It looks at the last token N.
- Predicts the upcoming word at position N + 1.

The Language Modelling Head



Language Model Head

takes h^L_N and outputs a distribution over vocabulary V

y1
y2
...
y|V|
Word probabilities 1 x |V|
Softmax over vocabulary V
u1
u2
...
u|V|
Logits 1 x |V|
Unembedding layer
U = E^T
Unembedding layer d x |V|
h^L_1
h^L_2
h^L_N
1 x d
Layer L Transformer Block
...
w1
w2
w_N

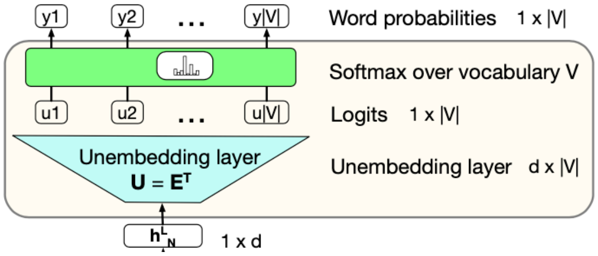
The Language Modelling Head

The unembedding layer takes as input a vector of size d.

It needs to produce an output of size |V|.

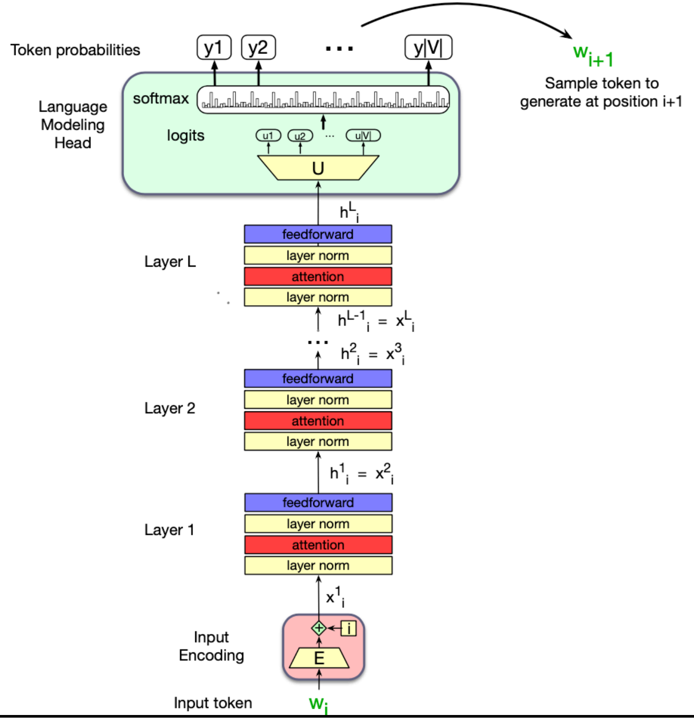
Conveniently, the input embeddings are of size |V|

Let's take the transpose of the input embeddings matrix.

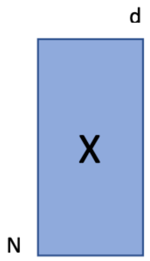


y1
y2
...
y|V|
Word probabilities 1 x |V|
Softmax over vocabulary V
u1
u2
...
u|V|
Logits 1 x |V|
Unembedding layer
U = E^T
Unembedding layer d x |V|
h^L_N
1 x d

Encoder Architecture



Parallelizing Self-Attention



- So far, we computed a single output at a single time step *i*.
- Each output, y_i, is computed independently.
- The calculation can be parallelized.
- We pack the input embeddings of the *N* tokens of the input sequence into a **single** matrix **X** $\in \mathrm{R}^{N\times d}$
- Each row of **X** is the embedding of **one** token of the input.
- Transformers for large language models can have an input length *N* = 1024, 2048, or 4096 tokens.
- **X** has between 1K and 4K rows, each of the dimensionality of the embedding *d*.

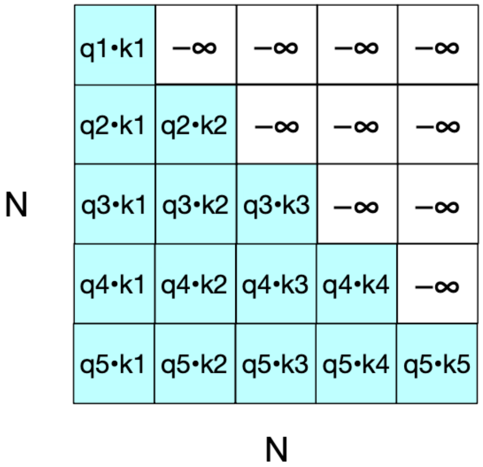
Parallelizing Self-Attention

- We multiply **X** by the key, query, and value matrices.
- They all are of size *d* x *d*.
- This produces matrices $Q \in \mathrm{R}^{N\times d}$, $K \in \mathrm{R}^{N\times d}$, and $V \in \mathrm{R}^{N\times d}$
- And the query, key, and value vectors:

  $Q=XW^Q$; $K=XW^K$; $V=XW^V$
- Given these matrices we can compute all the requisite query-key comparisons simultaneously by multiplying Q and K^T in a single matrix multiplication.
- The product is of shape *N* × *N*.

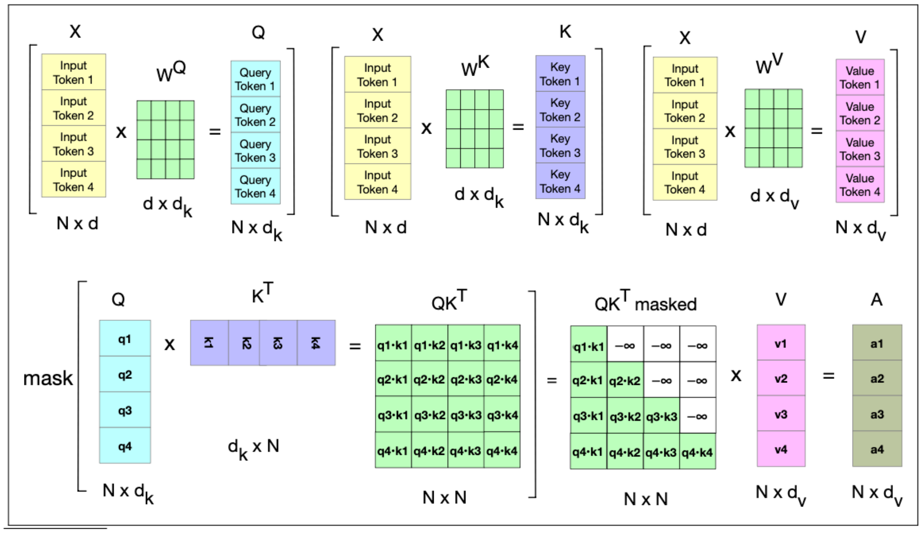
Masking out the Future

- The self-attention computation has a problem: the calculation in QK^T results in a score for each query value to every key value, *including those that follow the query*.
- This is inappropriate in the setting of language modeling: guessing the next word is pretty simple if you already know it!
- Hence, the upper-triangle portion of the comparisons matrix set to −∞.
- Softmax will turn them into zeros



q1•k1	−∞	−∞	−∞	−∞
q2•k1	q2•k2	−∞	−∞	−∞
q3•k1	q3•k2	q3•k3	−∞	−∞
q4•k1	q4•k2	q4•k3	q4•k4	−∞
q5•k1	q5•k2	q5•k3	q5•k4	q5•k5

Image source: Speech and Language Processing, Jursafky and Martin, Jan. 12, 2022 draft


X
Input Token 1
Input Token 2
Input Token 3
Input Token 4
N x d
x
W^Q
d x d_k
=
Q
Query Token 1
Query Token 2
Query Token 3
Query Token 4
N x d_k
X
Input Token 1
Input Token 2
Input Token 3
Input Token 4
N x d
x
W^K
d x d_k
=
K
Key Token 1
Key Token 2
Key Token 3
Key Token 4
N x d_k
X
Input Token 1
Input Token 2
Input Token 3
Input Token 4
N x d
x
W^V
d x d_v
=
V
Value Token 1
Value Token 2
Value Token 3
Value Token 4
N x d_v
mask
Q
q1
q2
q3
q4
N x d_k
x
K^T
k1
k2
k3
k4
d_k x N
=
QK^T
q1•k1 q1•k2 q1•k3 q1•k4
q2•k1 q2•k2 q2•k3 q2•k4
q3•k1 q3•k2 q3•k3 q3•k4
q4•k1 q4•k2 q4•k3 q4•k4
N x N
=
QK^T masked
q1•k1 −∞ −∞ −∞
q2•k1 q2•k2 −∞ −∞
q3•k1 q3•k2 q3•k3 −∞
q4•k1 q4•k2 q4•k3 q4•k4
N x N
x
V
v1
v2
v3
v4
N x d_v
=
A
a1
a2
a3
a4
N x d_v