**TRANSFORMERS**

**\*Transformers vs. Recurrent and Convolutional Architectures**

Language

Traditional Models: RNNs, LSTMs, and GRUs process sequences step-by-step, and often not suitable for long-range sentences.

Transformers: Use self-attention, allowing every element to interact with every other element, solving the issue with long-range sentences from the start.

Vision

CNNs:it processes the image locally then builds up a global understanding by identifying features like edges and corners.

Transformers (ViT): Use self-attention to model global information from the start. They divide images into patches and apply self-attention, efficiently processing large datasets and improves the accuracy.

Multimodal Tasks

Traditional : Specialized models for each data type made integration

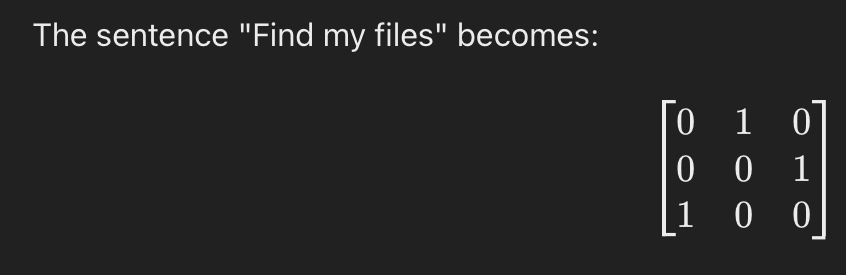
challenging.

Transformers: they provide a unified architecture for multiple data types. Cross-attention mechanisms allow combining information from different sources.

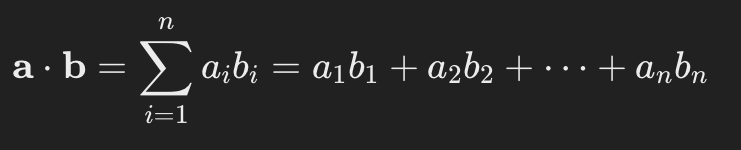
**\*Terminologies**

**1.one-hot encoding**

Computers process numerical data, but many inputs, like text and images, are not naturally numeric. One-hot encoding converts categorical variables into a numerical form suitable for ML.

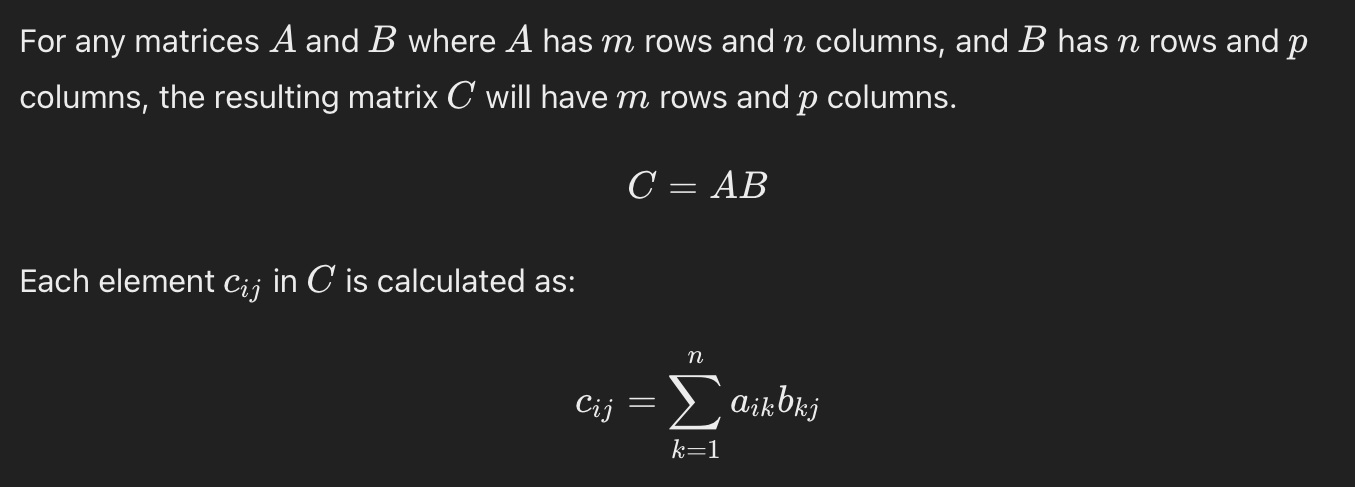
For ex in NLP-

**2.dot product**

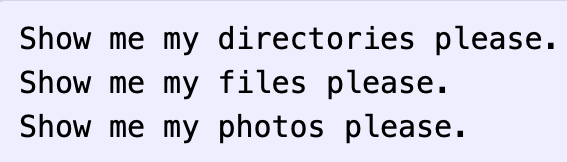
Dot product of two vectors is -

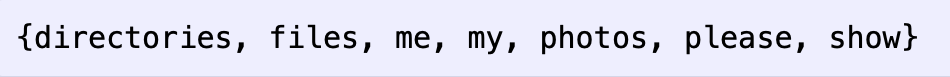
Geometrically in euclidian space-

Dot product of one hot vector with itself is 1 and with any other vector is 0.

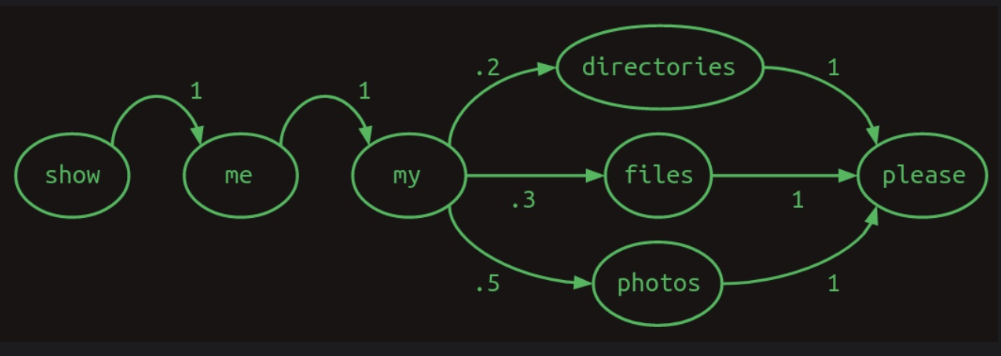
**3.matrix multiplication as dot product**

**4.First Order Sequence Model**

To develop an NN interface with -

The vocabulary is -

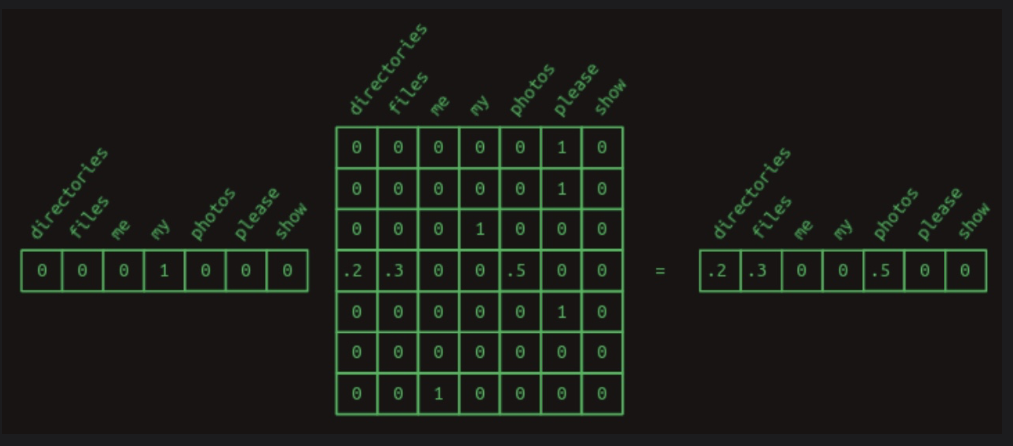
Markov Chain Transition Model

A first-order Markov chain represents word sequences where the probability of each word depends only on the previous word.

Now Markov chains can be represented as matrices too.



Above is a transition matrix.

To find transition probabilities of a word we multiply its one hot vector by the transition matrix.

**5.Second Order Sequence Model**

Predicting the next word is based on how hard is the current word.

For Ex-

1. Check whether the battery ran down please.
2. Check whether the program ran please.

A first-order Markov model uses the current word to predict the next. For instance, after "ran," both "down" and "please" are possible.

A second-order Markov model uses the two most recent words for predictions.

Knowing "battery ran" leads to "down," while "program ran" leads to "please," eliminating uncertainty.

This has N(2) rows for each pair of words.

**Second order Sequence model with skips**

It works well with longer dependencies by considering the most recent word combined with each previous word.

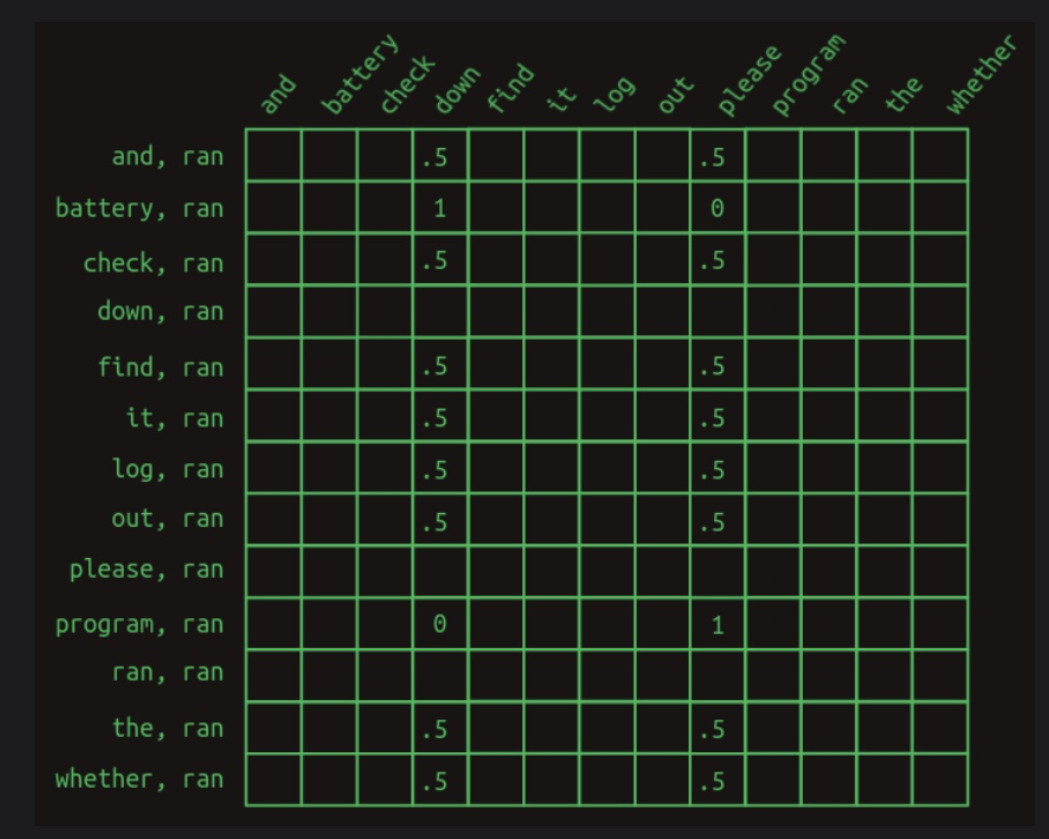
Ex-

1.Check the program log and find out whether it ran please.

2.Check the battery log and find out whether it ran down please.

To predict the word after "ran," we can consider combinations like "battery ran" and "program ran."

Skip Transition Matrix

This matrix shows rows for each combination of "ran" with preceding words, focusing on relevant transitions. Instead of probabilities, values represent “votes.”

In the above ex,

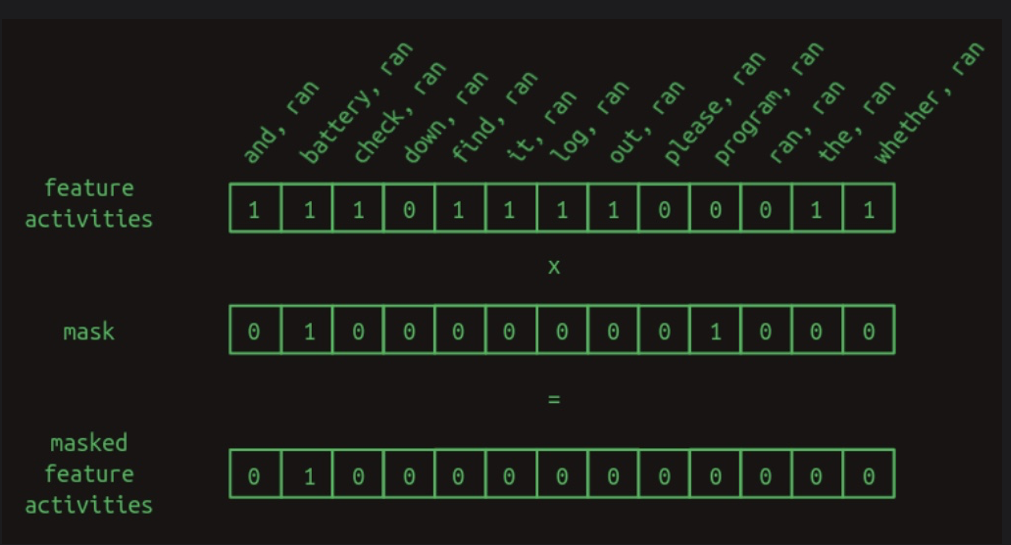
"battery ran" strongly votes for "down."

"program ran" strongly votes for "please."

Summing votes across all features determines the next word, handling even deep dependencies effectively.

**6.Masking**

To enhance model prediction , we focus on information feature and eliminate the noise.

Procedure given in below diagram-

To improve precision, we apply a mask to ignore noise:

The mask vector is filled with ones, except for positions we want to ignore, which are set to zero.

After masking the next word predictions become stronger .

For below ex-

1.Check the program log and find out whether it ran please.

2.Check the battery log and find out whether it ran down please.

After masking,

"battery, ran" predicts "down"

"program, ran" predicts "please"

**\*Transformer Core**

**\*Embeddings**

The Problem is that transformers require immense computational cost due to large transition matrices. A 50,000-word vocabulary results in over 100 trillion elements, which is impractical.

The Solution: Embeddings

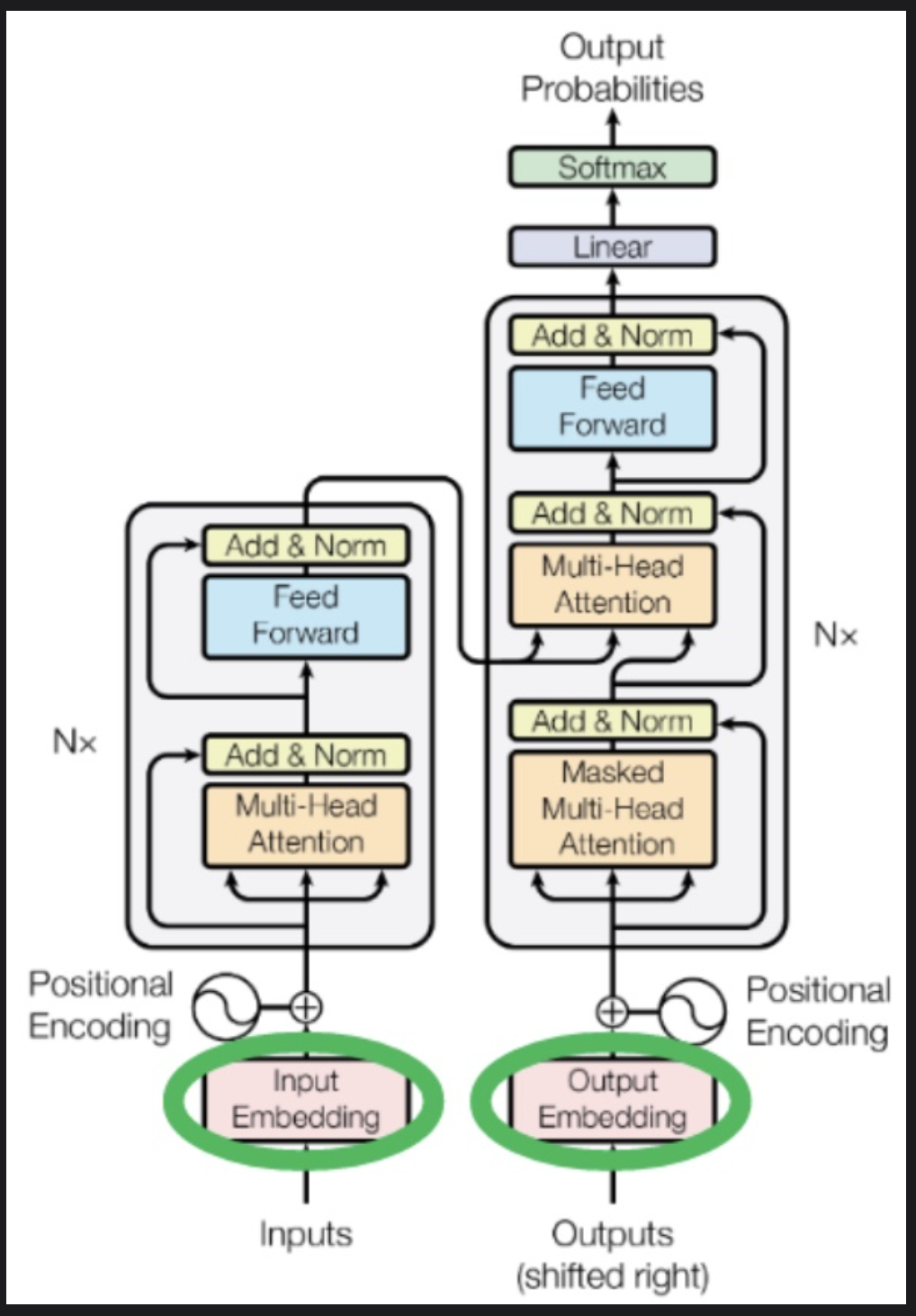
Embeddings reduce dimensionality by projecting high-dimensional one-hot vectors into a lower-dimensional vectors.

Working-

One-Hot Encoding: Each word is an N-dimensional vector with one element set to 1.

Projection: These vectors are projected into a lower-dimensional space via matrix multiplication.

Grouping: Similar words are grouped together, making models more efficient and needing less training data.

Architecture diagram used in the reference paper-

**\*Positional Encoding**

Unlike recurrent and CNNs, transformers don't inherently model word

positions.

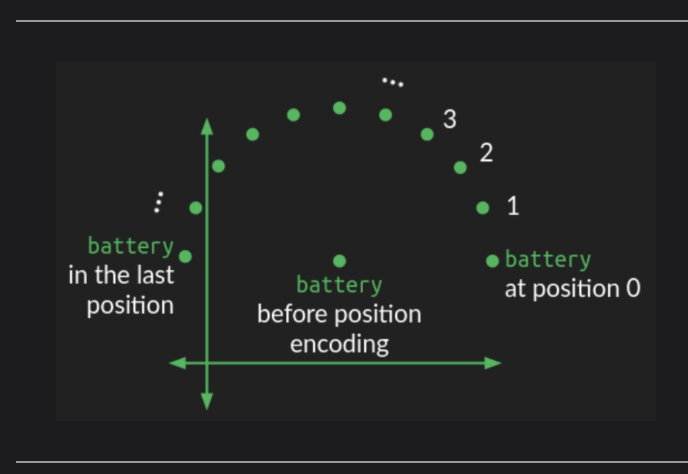
The Solution:

Positional embeddings add spatial information to the transformer,

addressing the position of words in a sequence.

In the original transformer to position information we used to add circular wiggle by using sinusoidal positional embeddings.

In advanced methods we use Rotary Position Embeddings (RoPE).

The diagram below shows-positional encoding introduces a circular wiggle, in addition of sinusoidal positional embeddings.

In the above diagram position of the word is at centre.Movement of each word by same distance creates a circle.

Since the circle is 2-D representing a circular wiggle requires modifying two dimensions of the embedding space.If the embedding space has more than 2 dimension then circular wiggle is repeated in all pairs of dimensions and with diff angular frequency.

**\*Decoding Output Words / De-embeddings**

De-Embedding Process

Matrix Multiplication: The embedded vector is multiplied by the

de-embedding matrix.

Vector Output: The resulting vector is deep, with high values for words similar to the embedded word and lower values for others.

Argmax : The word with the highest value is selected recreating a one-hot vector. This is called greedy sequence completion.

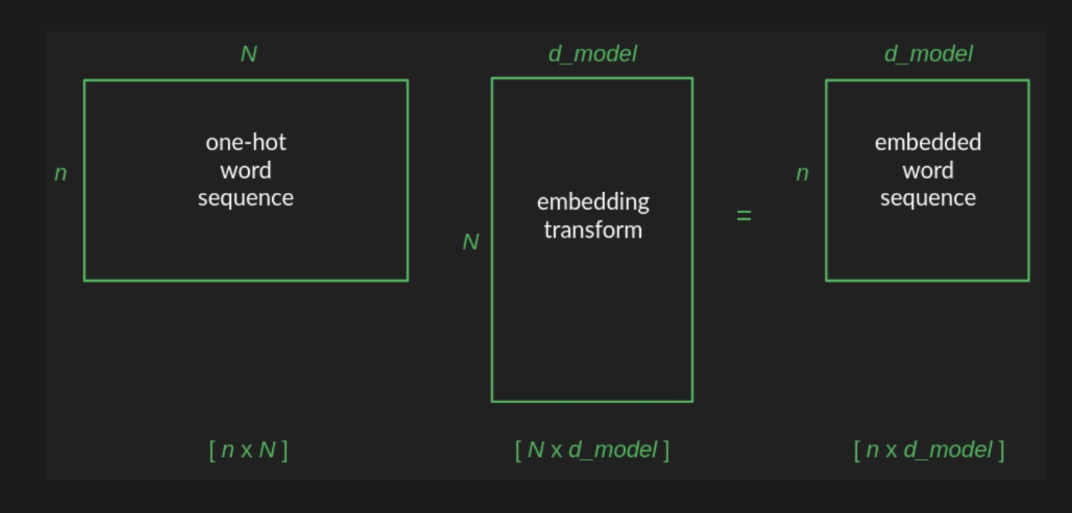
**\*Attention**

Imp terms-

N-Vocabulary size(tens of words) -13

-n - max sequence length -12

d(model): number of dimensions in the embedding space used throughout the mode.-512

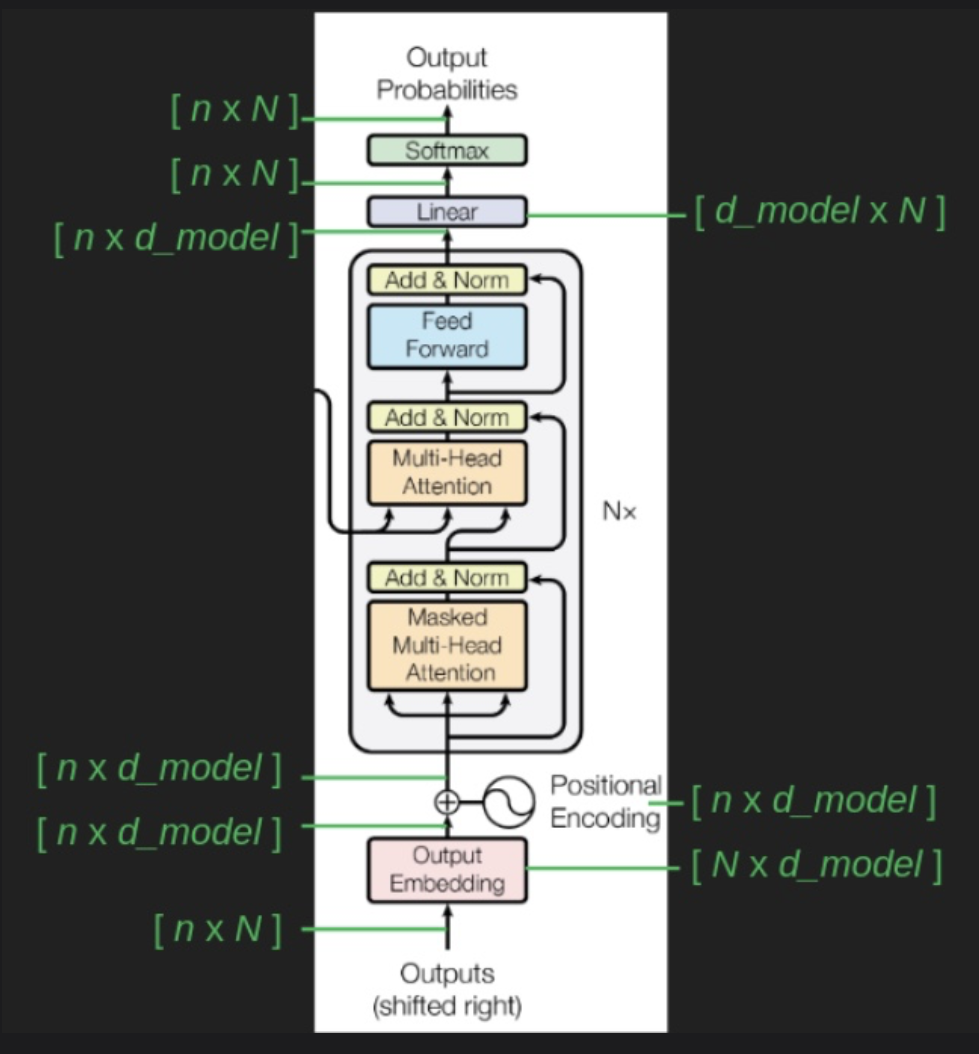
Input is taken from sentence in form of one hot vector [n x N]

In above by multiplying one hot sequence with embedding we get [n x d\_model]embedded sequence.

After initial embedding positional encoding is additive.

Now the embedded sequence goes through attention layers(shape changes)

Now de-embedding restores it previous shape and output the probabilities.

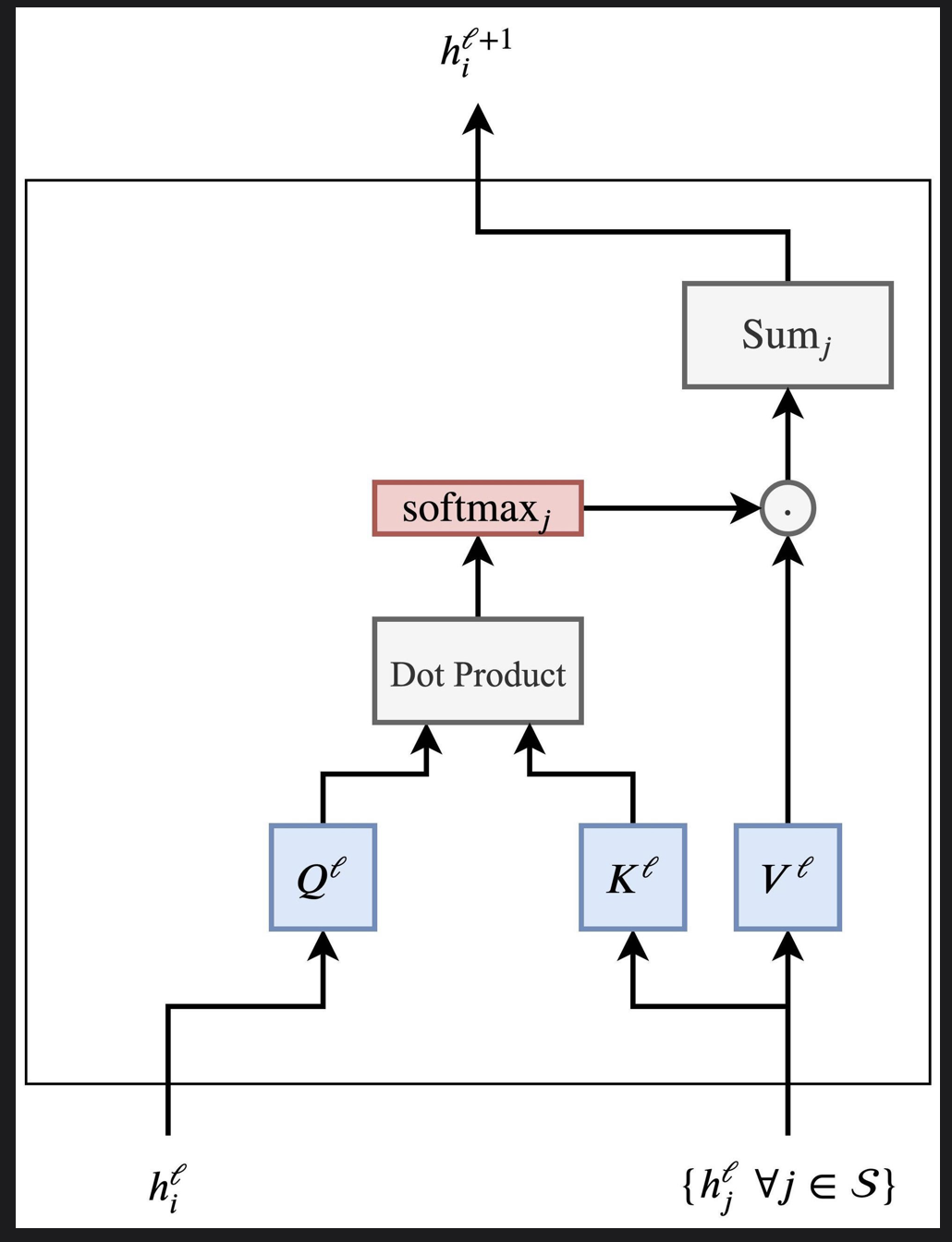


Types of Attention: Additive, Multiplicative (Dot-product), and Scaled

Transformers work on self-attention .

**\*Attention calculation**

Architecture blueprint given from paper-

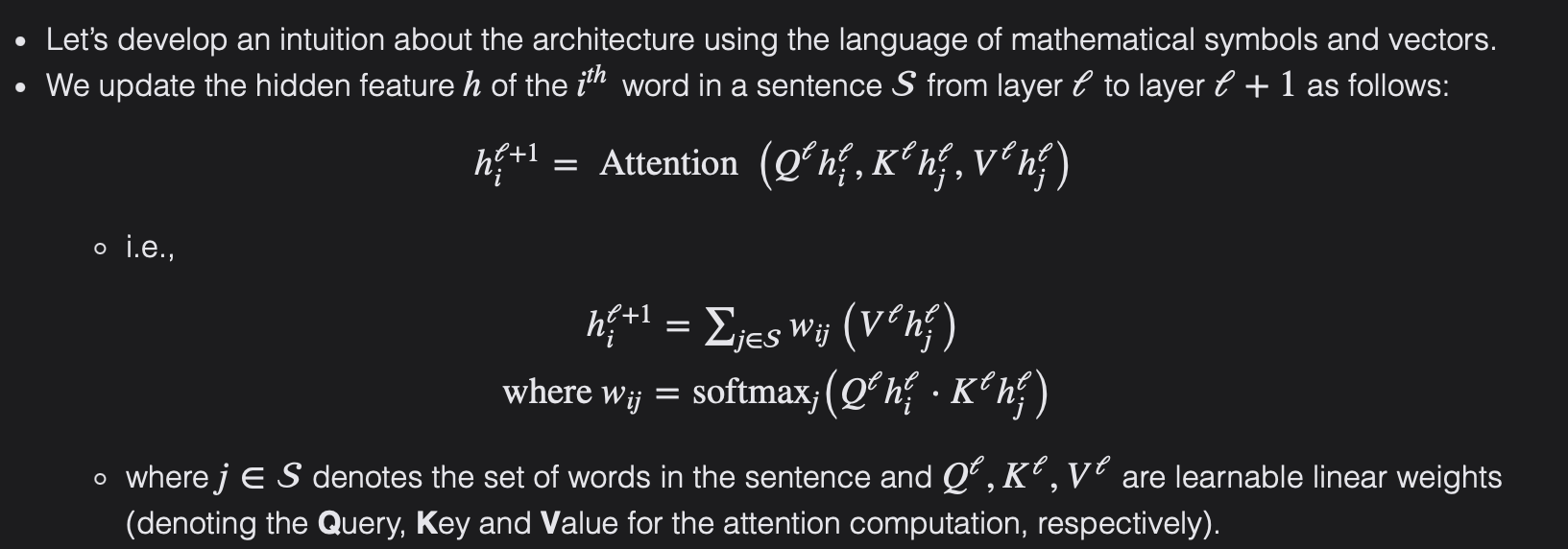


**\*Self attention**

Self-Attention: A mechanism where queries, keys, and values are all derived from the same input sequence.

Parallelization: Unlike RNNs, which process data sequentially, self-attention processes all words in parallel. This leads to faster training and better use of GPUs.

Working

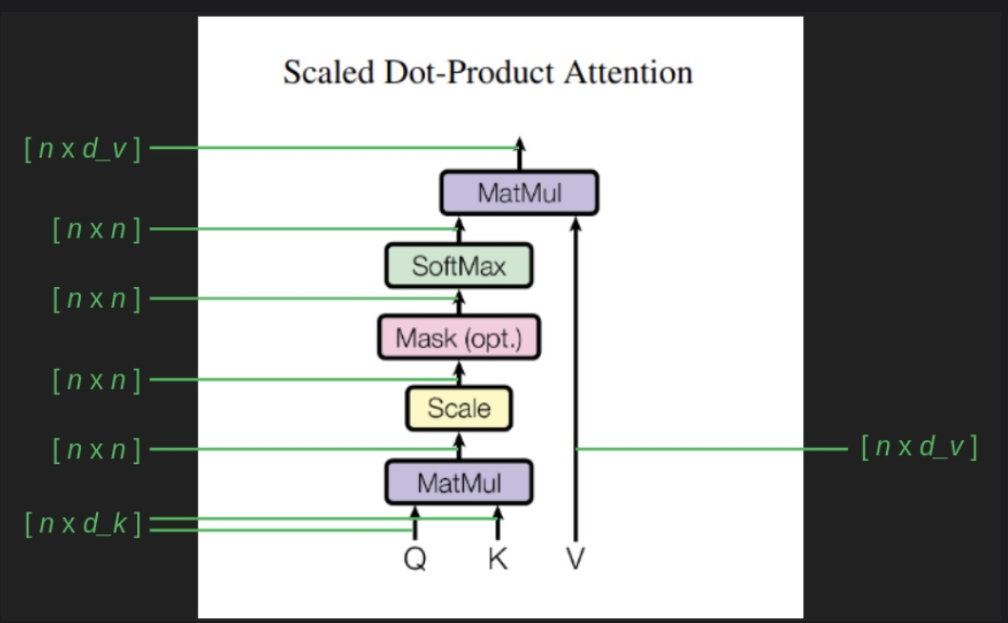
Create query, key, and value vectors from the input.

Using dot-product to measure similarity between queries and keys.

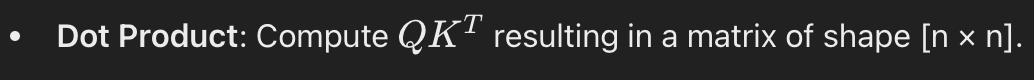
Update Features

**\*Single Head Attention**

Self-attention uses matrices to compute the relationship between words based on their embeddings. The process involves queries and keys into lower-dimensional spaces and using these to focus attention on required words.

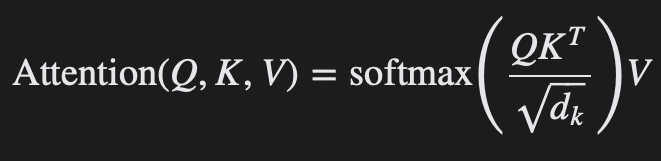


Queries (Q) and Keys (K): Shape [n × d\_k]

Values (V): Shape [n × d\_v]

Divide by sqrt(dk) for scaling.

Apply softmax to get the probabilities.

Formula-

**Calculating Q, K, and V Matrices:**

* Embedding: Each word is embedded into a 512-dimensional vector.
* Self-Attention: Inputs are transformed by Q, K, and V matrices, which are learned projections. Each input vector is split into query, key, and value vectors.
* Multi-Head Attention: Involves multiple sets of Q, K, and V matrices to capture different aspects of the input.

**\*Multi-head attention**

Runs multiple attentions and focuses on different aspect of data.

**\*Cross attention**

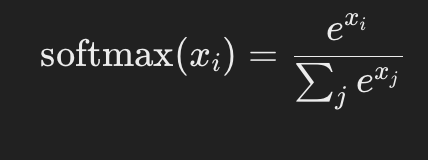
Connects the encoder and decoder by using encoder outputs as keys and values, and decoder outputs as queries.

**\*dropout**

Dropout in sub-layers:applied after in sub-layers and before addition of residual connection & normalization.

In embeddings applied to sum of embedding and positional encodings.

**Skip connections with layer normalization an be implemented.**

Then we use softmax unit go get output as a distribution of probabilities matrix for easier implementation.

**\*Stacking**

Multiple layers provide path to reach better solution as by stacking layers we can predict word easily based on multiple layers.

Helps to mitigate errors.

**\*transformer encoder & decoder**

The Transformer model has two parts: encoder and decoder.

Encoder:

Function: Encodes input sequences into abstract representations.

Components:

Self-Attention Layers: Capture relationships within the input sequence.

Feed-Forward Layers: Further process the attended information.

Residual Connections & Layer Normalization: Enhance training stability and performance.

Decoder:

Function: Decodes the encoded representation to generate the output sequence.

Components:

Masked Self-Attention: Ensures autoregressive generation by attending only to previous tokens.

Cross-Attention: Attends to the encoder's output.

Feed-Forward Layers: Further process the attended information.

Residual Connections & Layer Normalization: Similar to the encoder.

Encoder process the entire input sequence and decoder processes the output receieved from encoder.

Process-

Input sentece is tokenized and converted into embeddings.

Then,

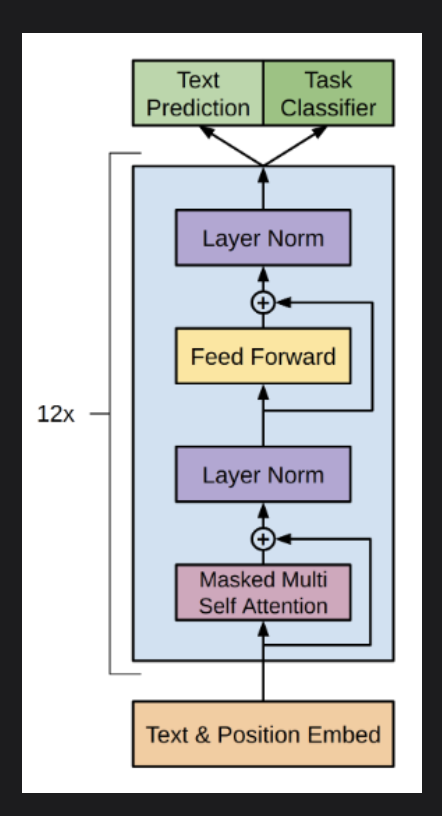
Self-Attention: Computes attention scores and context vectors.

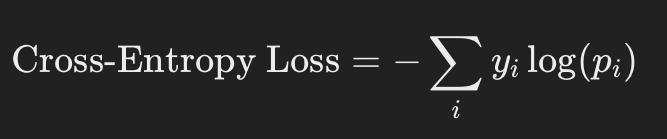
Feed-Forward: Applies linear transformations with non-linear activation.

Then establish residual connection and normalization .

Then decoder receives the input from encoder.

Finally decoder make the prediction.



**\*Loss function**

The encoder & decoder are jointly trained to minimise cross-entropy loss.

the cross-entropy loss “pulls” the predicted probability of the correct class towards 1 during training.by calculating gradients w.r.t. the weights; with the model’s sigmoid/softmax output serving as the prediction.

The entire transformer setup -