NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

EE6405

Natural Language Processing

Dr. Simon Liu
NTU Electrical and Electronic Engineering

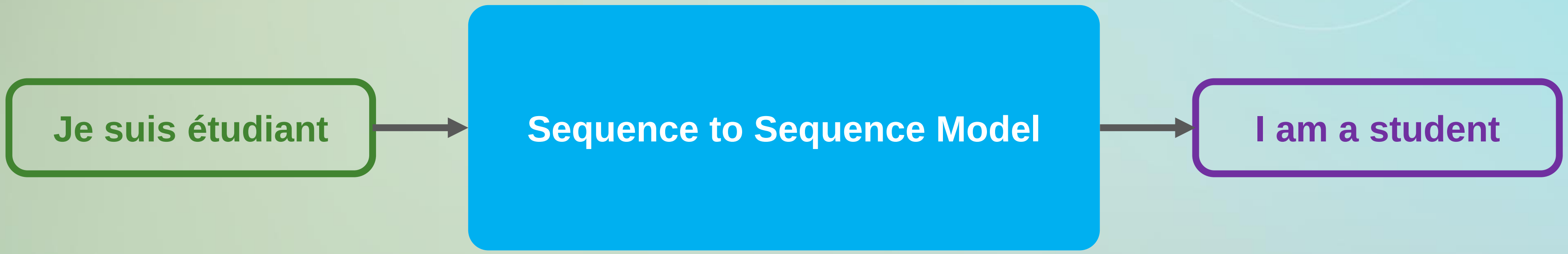
Transformers

Seq2Seq Models

- Sequence-to-sequence (Seq2Seq) models are models that take a sequence of items (words, letters, features of an images…etc) and outputs another sequence of items.

Neural Machine Translation

Sequence to Sequence Model

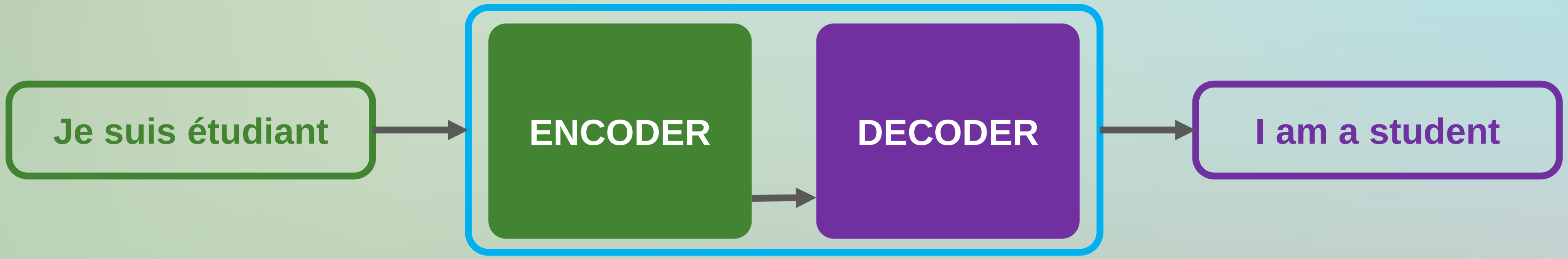


Seq2Seq Models

- Seq2Seq models follow an encoder-decoder architecture.
- The encoder processes the input sequence step by step, typically using RNN or more advanced models like LSTMs or GRUs.
- A context vector that captures the input meaning is generated by the encoder.
- The decoder takes the context vector produced by the encoder and uses it to generate the output sequence.

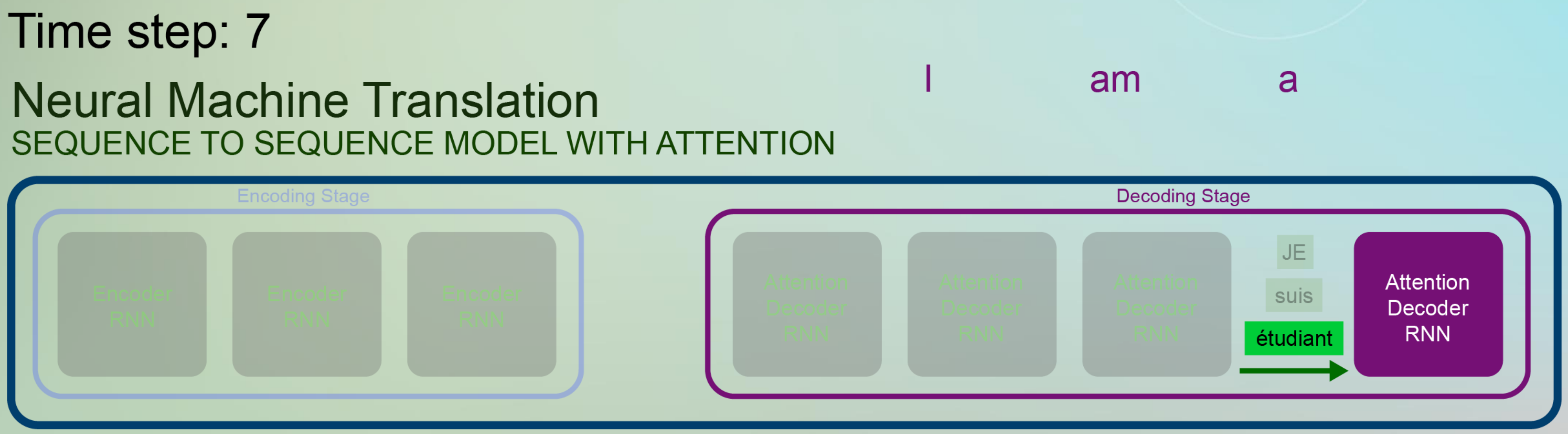
Neural Machine Translation

Sequence to Sequence Model



Attention Mechanism

- Seq2Seq models often experience an information bottleneck in the context vectors.
- As the length of the input sequence increases, it becomes increasingly difficult for the context vector to retain all the relevant information, leading to a degradation in the performance of the model.
- LSTMs and GRUs are not sufficient to capture dependencies in very long sequences.
- Attention mechanism resolves this problem by allowing the model to focus on the relevant parts of an input sequence.



Attention Mechanism

- The general attention mechanism makes use of three main components namely the queries ($\boldsymbol{Q}$), the keys ($\boldsymbol{K}$) and the values ($\boldsymbol{V}$).
- The general attention mechanism then performs the following computations:
- Each query vector, $q = s_{t-1}$, is matched against a database of keys to compute a score value. This matching operation is computed as the dot product of the specific query under consideration with each key vector, k_i.

$$\boxed{e_{q,k_i} = q \cdot k_i}$$

- The scores are passed through a softmax operation to generate the weights:

$$\boxed{\alpha_{q,k_i} = softmax(e_{q,k_i})}$$

- The generalised attention is then computed by a weighted sum of the value vectors, v_{k_i}, where each value vector is paired with a corresponding key:

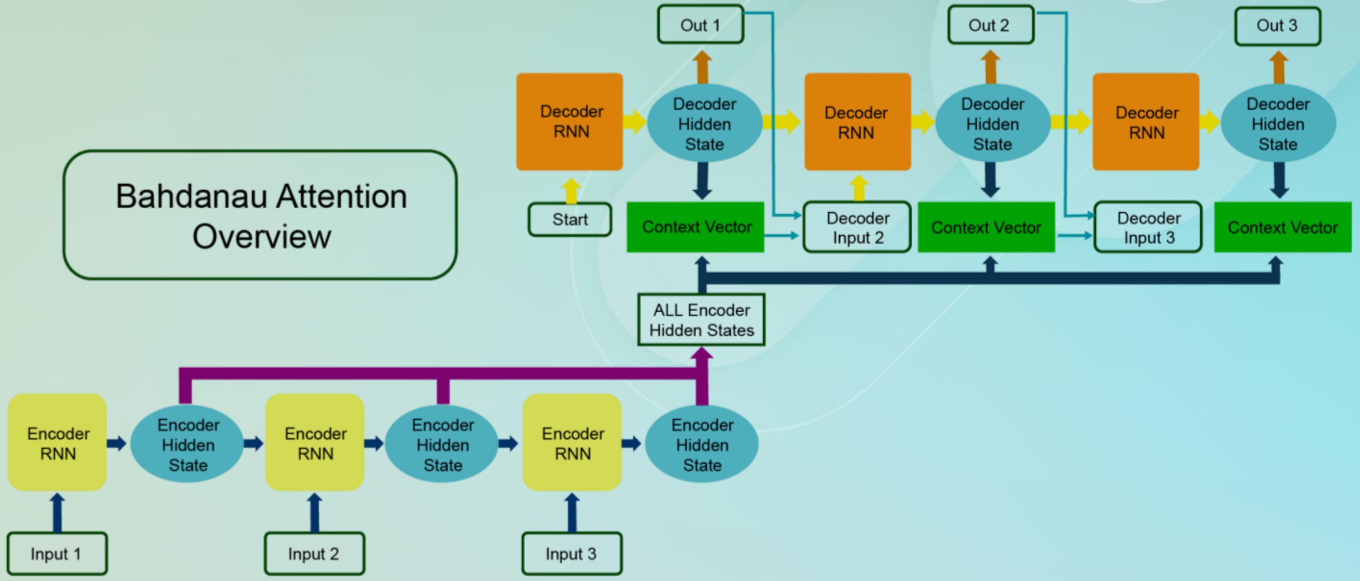
$$\boxed{\boldsymbol{attention(q, K, V) = \sum_i \alpha_{q,k_i} v_{k_i}}}$$

Attention Mechanisms

- When the generalized attention mechanism is presented with a sequence of words:
  - The query vector attributed to some specific word in the sequence is scored against each key in the database.
  - In doing so, it captures how the word under consideration relates to the others in the sequence.
  - The values are then scaled according to the attention weights (computed from the scores) to retain focus on those words relevant to the query.
  - An attention output for the word under consideration is then produced.

Attention Mechanisms

- Scores are computed using each of encoder hidden state (keys) and the current decoder hidden state (query).
- The scores are passed through a softmax function to obtain normalised attention weights.
- The attention weights obtained from the softmax operation are used to calculate a weighted sum of the encoder hidden states.
- This weighted sum is then used to provide context information to the decoder for the current time step.
- This sum is concatenated with the input to the decoder at the current time step. The concatenated vector is then used as input to the decoder's recurrent unit.



Transformer Models

- Transformers are a type of deep learning architecture introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017.
- RNNs, LSTMs and GRUs all face challenges when dealing with long-range dependencies in sequences.
- We have seen that attention mechanism can help address the long-range issue.
- Given that attention gives us access to any state, do we still need the underlying RNNs (or LSTMs or GRUs)? Or can we build something simpler based on the attention mechanism?

Transformer Models

- Transformers are sequence-to sequence encoder-decoder models.
- Uses attention mechanisms to weigh the importance of different elements in the input sequence.
- Transformer models have the following key components:

Input Embeddings

convert input tokens into embeddings

Positional Encoding

add positional information to embeddings

Multi-Head Self-Attention

compute attention scores across different positions

Transformer Models

Encoder Layers:

- **Multi-Head Self-Attention:** Compute attention scores across different positions
- **Feed-Forward Neural Network**: Processes the output from the self-attention mechanism
- **Residual Connection & Layer Normalisations**: Applied after self-attention and after the feed-forward network

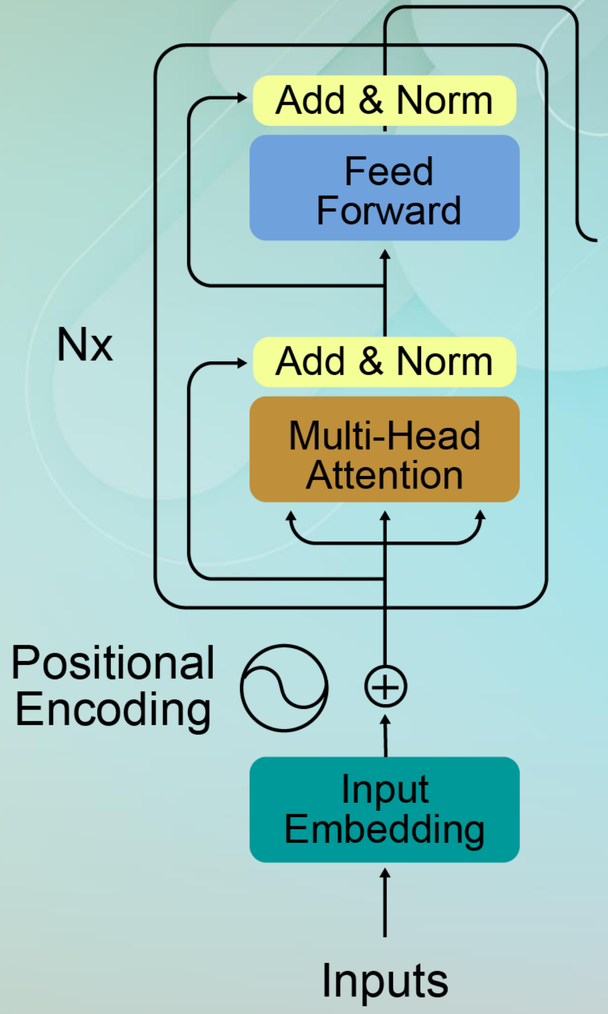
Decoder Layers:

- **Masked Self-Attention**: Prevents positions from attending to subsequent positions.
- **Encoder-Decoder Attention**: Focuses on relevant parts of the input sequence.

Transformer Models – Encoder

The input to encoder setup is as follows:

- The input is first converted into an embedding.
- This is followed by a positional encoding layer.
- The encoder layer consist of multi-head self-attention.
- Followed by a feedforward neural network.
- Normalisation is done in the Add & Norm steps.
- This encoder layer can be stacked multiple times depending on complexity of the task
- In the original paper, the encoder layer is stacked 6 times.



Transformer Models – Positional Encodings

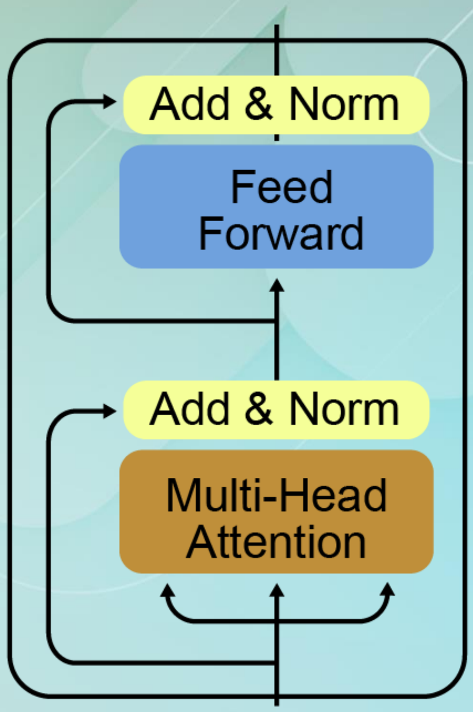
- Since Transformer models do not inherently process sequences in order (like RNNs or LSTMs), positional encodings are added to input embeddings to give the model information about the position of each token in the sequence.
- Positional encodings ensure that each position in the sequence has a unique representation.
- Positional encodings are added to the token embeddings. This combination allows the model to process both the content of the tokens and their positions in the sequence simultaneously.
- In the original paper, positional encodings are created using sinusoidal functions of different frequencies, ensuring that each position generates a unique encoding and that these encodings are consistent across different sequence lengths.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

Transformer Models

- The complete transformer block has two sublayers:
  - **Multi-head Self-Attention:** Processes input by attending to different positions of the sequence simultaneously in multiple representational subspaces.
  - **Feedforward neural network:** Applies two linear transformations with an activation function in between to each position, enhancing the representation independently for each position.
- Each of these two sublayers also has:
  - **Residual connection:** Facilitates deeper model architectures and mitigating the vanishing gradient problem.
  - **Layer Normalisation:** Normalises the output of each sublayer post-residual connection, stabilizing and accelerating the training process.



Transformer Models – Self Attention

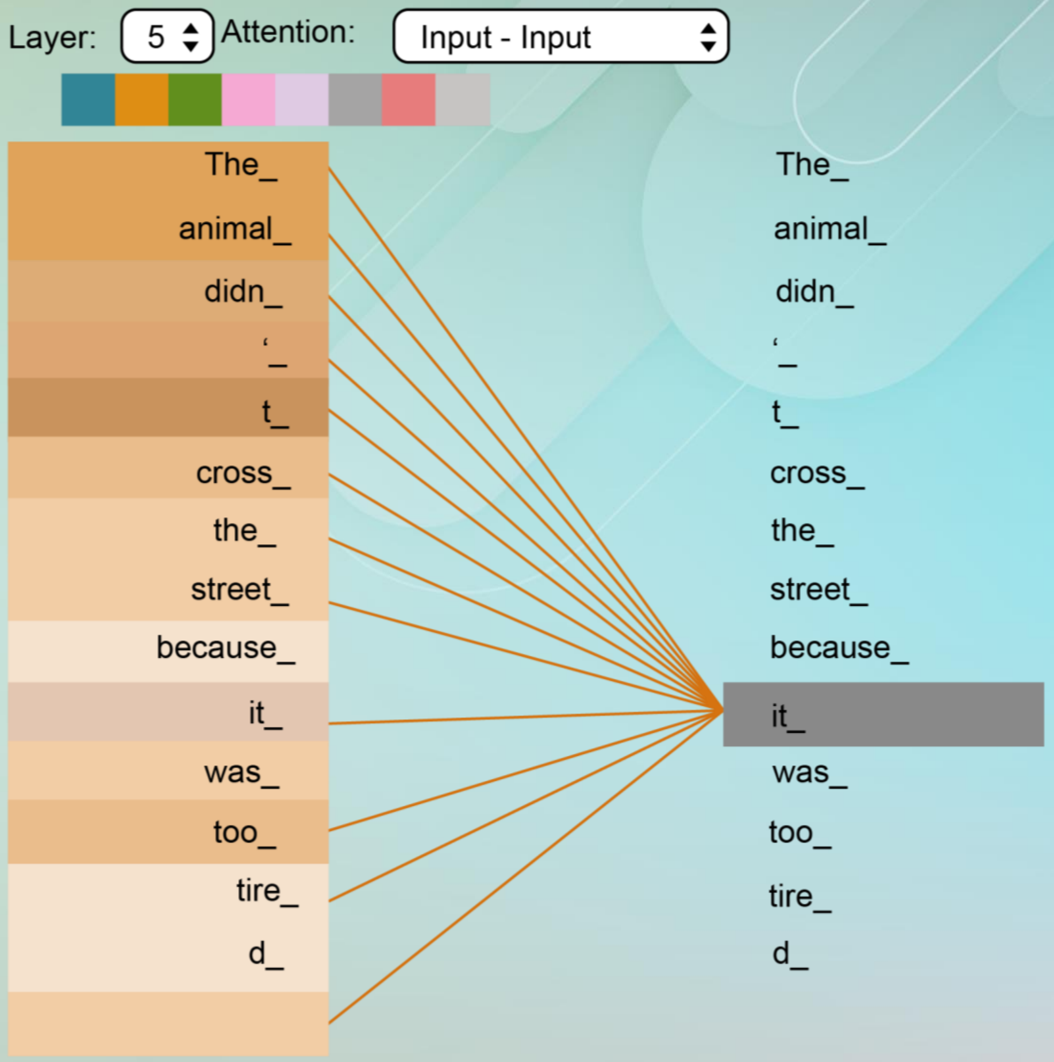
- Self-attention operates on a single sequence, allowing each position in the sequence to attend to all other positions within the same sequence.
- This contrasts with general attention mechanisms, which often involve attending to a different sequence (**like in seq2seq models where the decoder attends to the encoder's output**).
- Specifically, key, query, and value all come directly from the same input sequence.
- Each query is compared to all keys (including itself) using a dot product operation to compute attention scores, indicating the relevance of each key to the query.
- The scores can be scaled by a factor, typically the square root of the dimension of the key vectors, to control the magnitude of the scores.
- These scores are then normalised (typically using a softmax function) and used to create a weighted sum of the values.

Transformer Models

This self-attention allows the model to register the following:

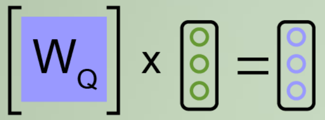
- **The animal didn't cross the street because it was too tired.**
- Self-attention allows the model to associate the "it" in the sentence with the word "animal".

The association of "it" with the other words in the sentence.



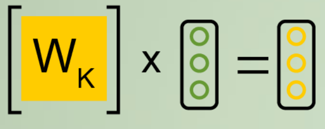
Transformer Models

Each vector receives three representations ("roles")



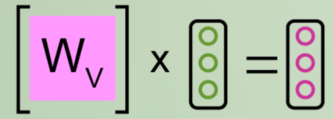
Query: vector from which the attention is looking

"Hey there, do you have this information?"



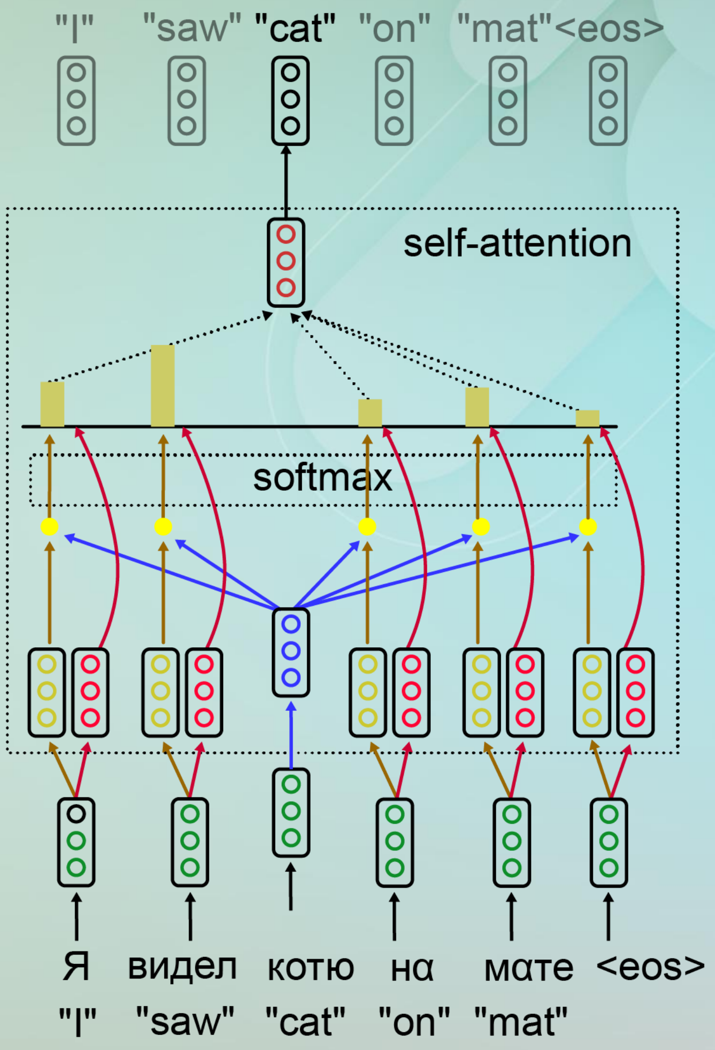
Key: vector at which the query looks to compute weights

"Hi, I have this information — give me a large weight!"



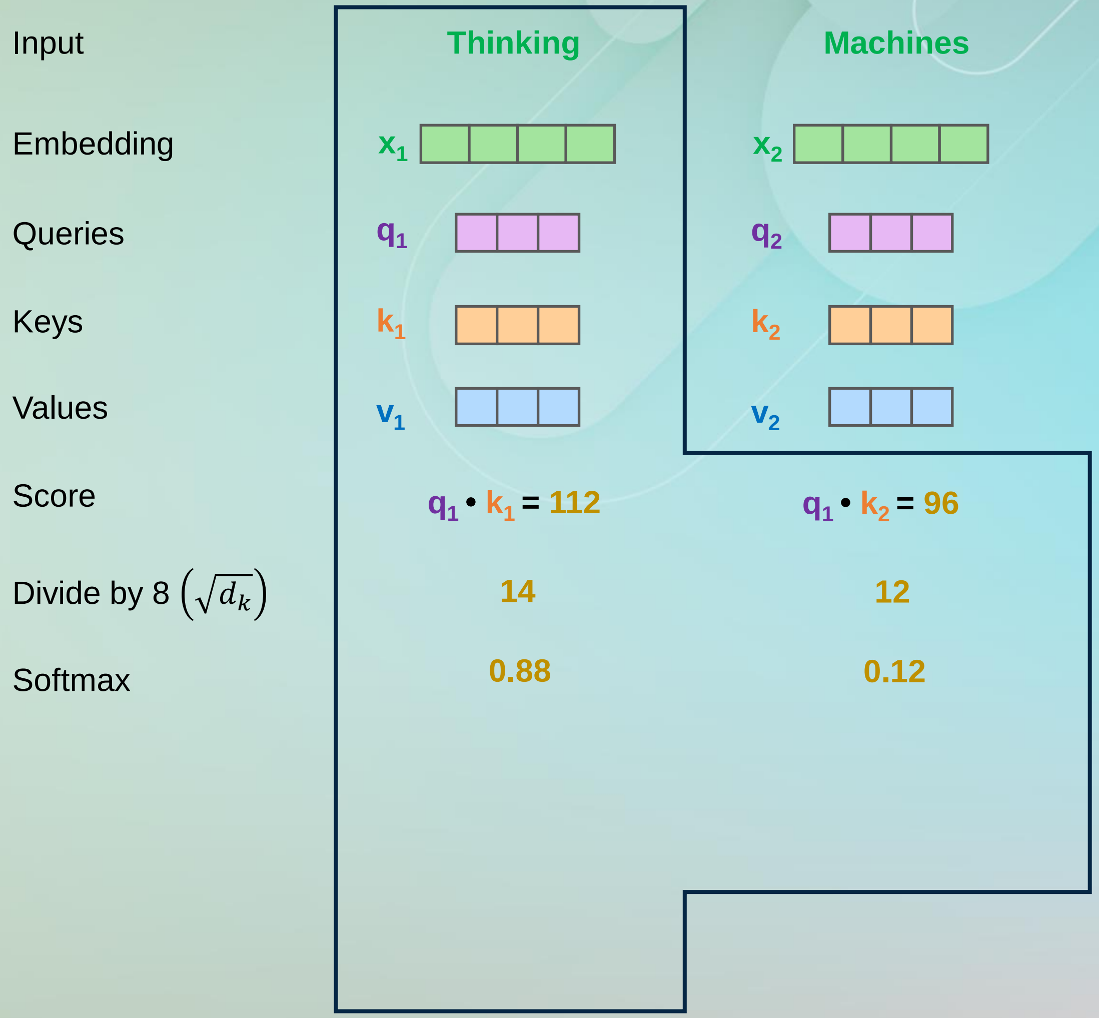
Value: their weighted sum is attention output

"Here' the information I have!"



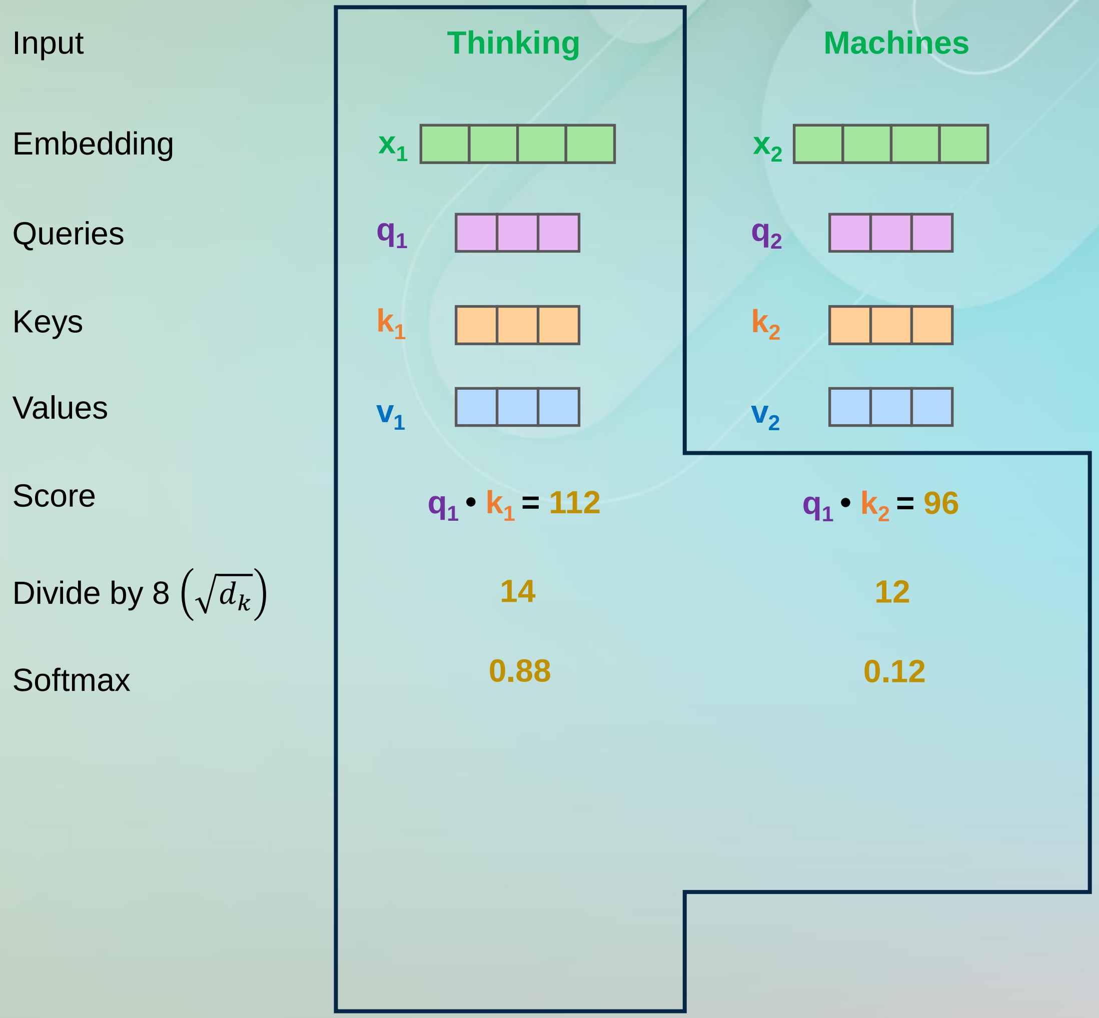
Transformer Models

- To calculate the self-attention for the first word in this example, “Thinking”.
- We need to score each word of the input sentence against this word.
- The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.
- The score is calculated by taking the dot product of the query vector with the key vector of the respective word that we are scoring.



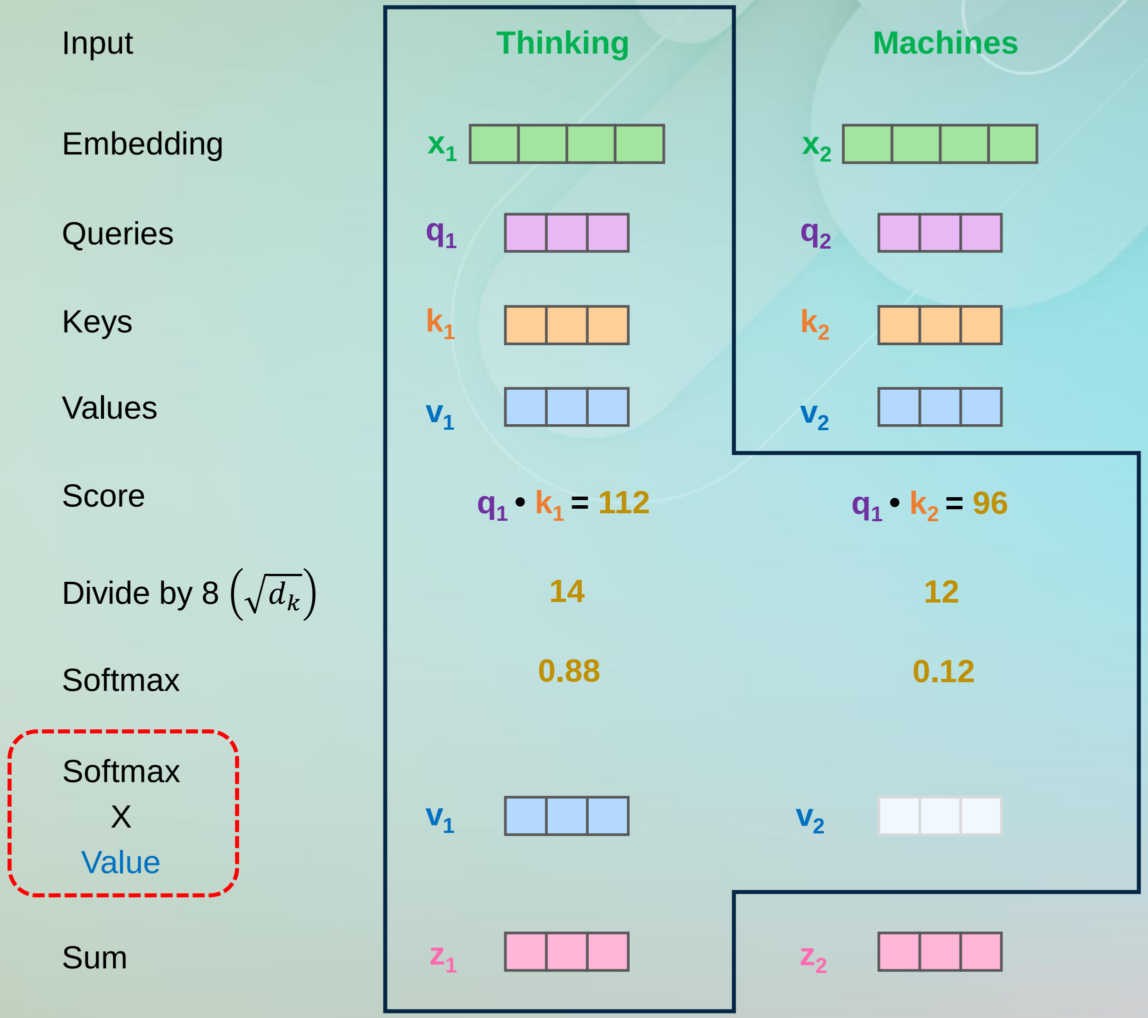
Transformer Models

- The scores are then divided by 8, to allow for more stable gradients.
- The result is passed through a softmax operation.
- Softmax normalises the scores so they're all positive and add up to 1.
- The softmax score determines how much each word will be expressed at this position.



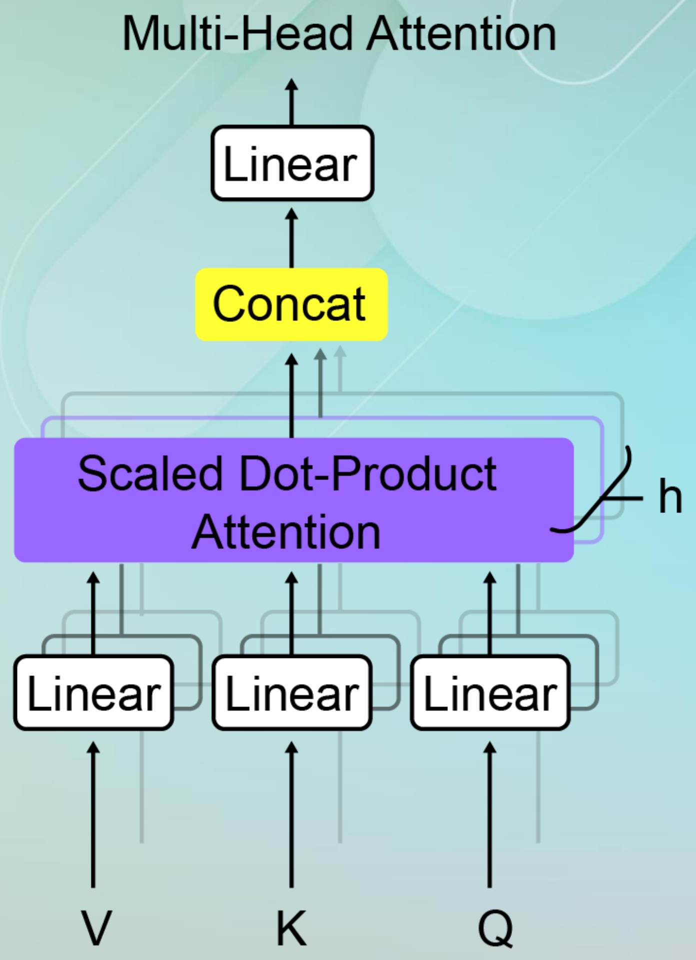
Transformer Models

- Multiply each value vector by the softmax score.
- Keep intact the values of the word(s) we want to focus on and drown-out irrelevant words.
- Sum up the weighted value vectors. This produces the output of the self-attention layer at this position for the first word.



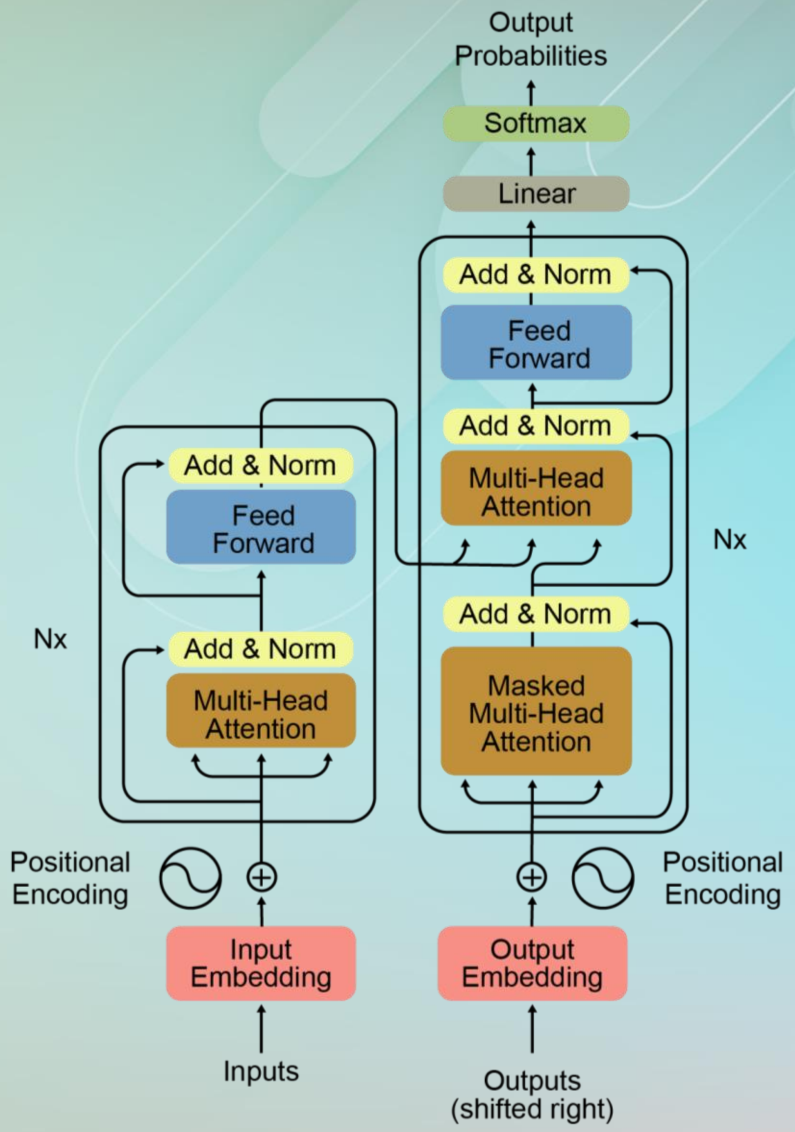
Transformer Models – Multi Head Attention

- **Parallel Attention Heads:** Executes several self-attention mechanisms (attention heads) in parallel, allowing the model to capture different relationships in the data simultaneously.
- **Diverse Representations:** Each head can learn distinct aspects of the input, providing a more comprehensive understanding of the sequence.
- **Concatenation and Transformation:** Outputs of all heads are concatenated and linearly transformed to produce the final attention output.
- **Enhanced Capacity:** Increases the model's capacity and expressiveness without significantly raising computational complexity.



Transformer Models – Decoder

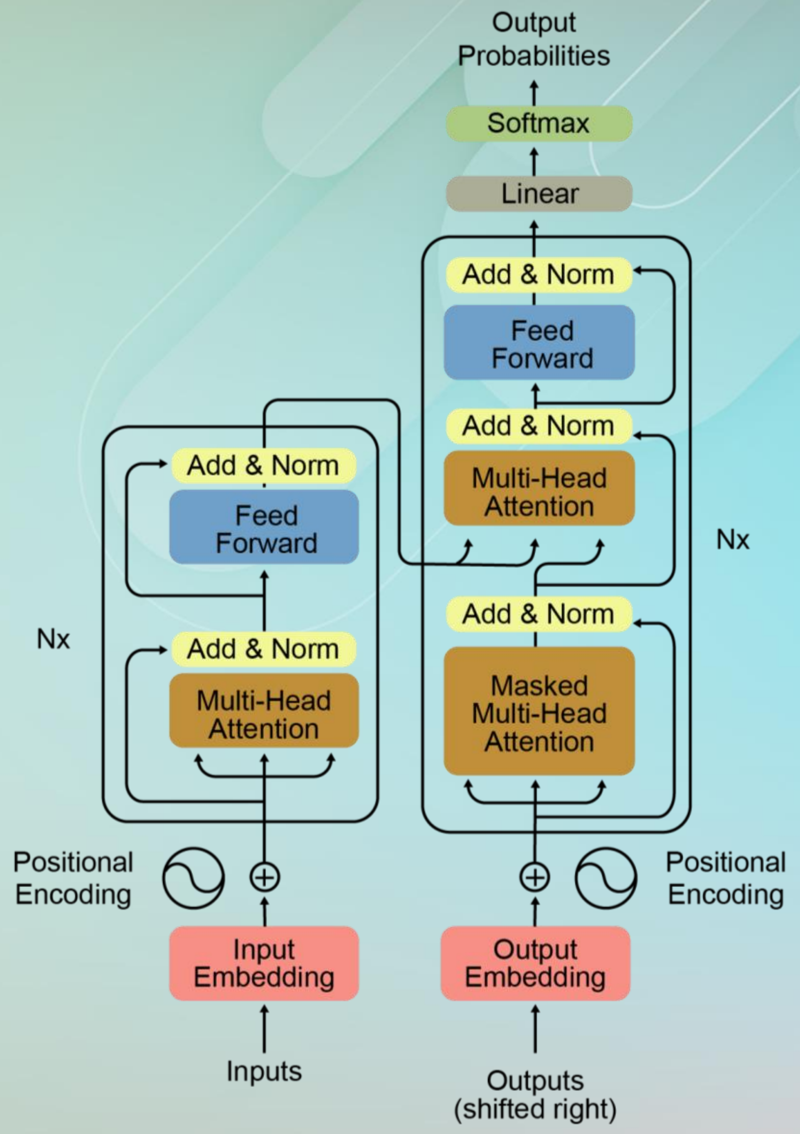
- The decoder in the transformer contains 3 layers:
  - Masked multi-head self-attention.
  - Encoder-decoder attention.
  - Feed-forward neural network.
- Similar to the encoder, each of the sublayers also has:
  - Residual connection
  - Layer Normalisation
- In the original paper, the decoder blocks are also repeated 6 times.
- Finally, the results go through a linear layer and softmax to generate the final output probabilities.



Transformer Models – Decoder

- **Masked multi-head self-attention:**
  - Allows each position in the decoder to attend to all positions up to and including that position in the decoder sequence.
  - The masking prevents positions from attending to subsequent positions, maintaining the auto-regressive property.

- **Encoder-Decoder Attention:**
  - Allows the decoder to focus on different parts of the input sequence.
  - It's similar to multi-head self-attention but the queries come from the previous decoder layer, and the keys and values come from the output of the encoder stack.



Summary

- **Sequence-to-Sequence (Seq2Seq) Model:**
  A neural network architecture designed for converting sequences from one domain to sequences in another domain, commonly used in tasks like machine translation.

- **Attention Mechanisms:**
  - A mechanism that allows neural networks to focus on different parts of input sequences when making predictions, enhancing their ability to capture and utilize relevant information.
  - Resolves the performance issue on long sequences that plagues RNNs and LSTMS.

- **Transformer Models:**
  - Utilise self-attention mechanisms to process entire input sequences simultaneously, providing efficiency and effectiveness in capturing long-range dependencies.
  - Feature an encoder-decoder architecture with multi-head attention, allowing the model to capture a diverse range of information and relationships within the data.
  - Do not rely on recurrence or convolution, making them well-suited for parallel processing and handling sequences with complex structures.