EMBEDDED LEARNING

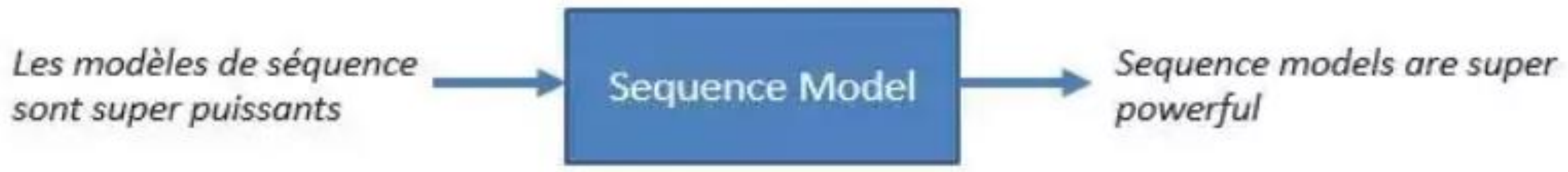
SESSION 4: ATTENTION MECHANISM
28/01/2025

By:
Priyanka Saxena,
Assistant Professor, CSE, KMIT.

SEQ TO SEQ MODEL

- Seq2Seq models are a type of neural network architecture used for transforming one sequence into another, such as translating a sentence from one language to another.
- They are particularly useful in tasks like machine translation, text summarization, and conversational agents.

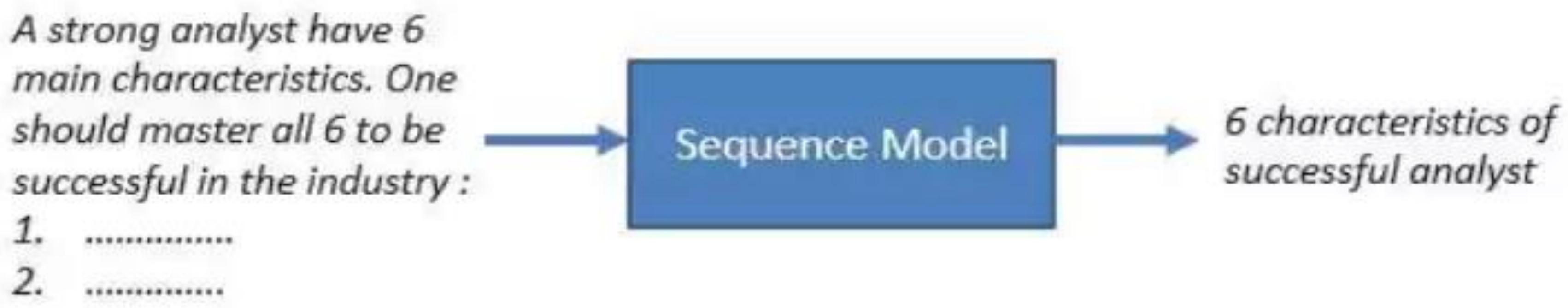
Machine Language Translation



Les modèles de séquence sont super puissants

Sequence models are super powerful

Text Summarization



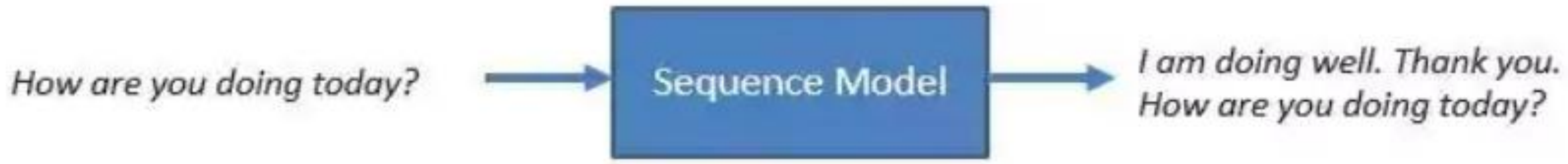
A strong analyst have 6 main characteristics. One should master all 6 to be successful in the industry :

1. ....

2. ....

6 characteristics of successful analyst

Chatbot

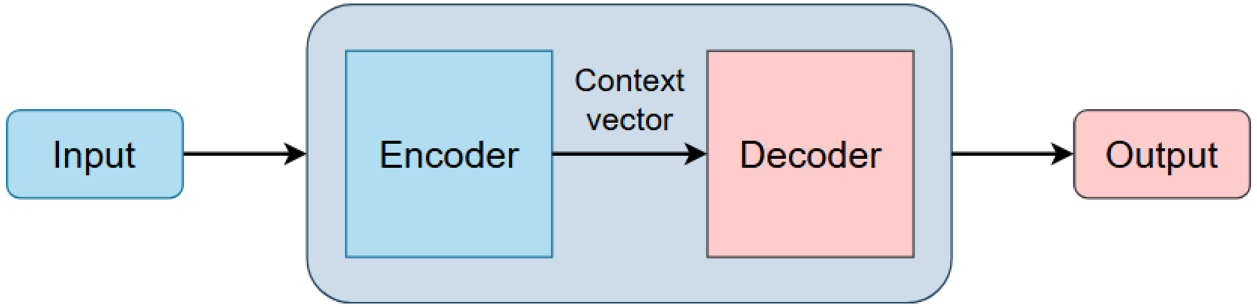


How are you doing today?

I am doing well. Thank you. How are you doing today?

SEQ TO SEQ MODEL

- A seq-to-seq model consists of following key components:
  - Encoder
  - Context vector
  - Decoder



Components of seq2seq models

Encoder:

- The encoder processes the input sequence and compresses it into a fixed-size context vector.
- It typically consists of recurrent layers (RNN, LSTM, or GRU) that encode information from the input sequence step by step.

SEQ TO SEQ MODEL

Context Vector:

- The context vector is a single vector that summarizes the entire input sequence.
- This vector is passed from the encoder to the decoder.

SEQ TO SEQ MODEL

Decoder:

- The decoder generates the output sequence from the context vector.
- Similar to the encoder, it also consists of recurrent layers.
- It produces one output token at a time until the end of the sequence is reached.

SEQ TO SEQ MODEL - ISSUES

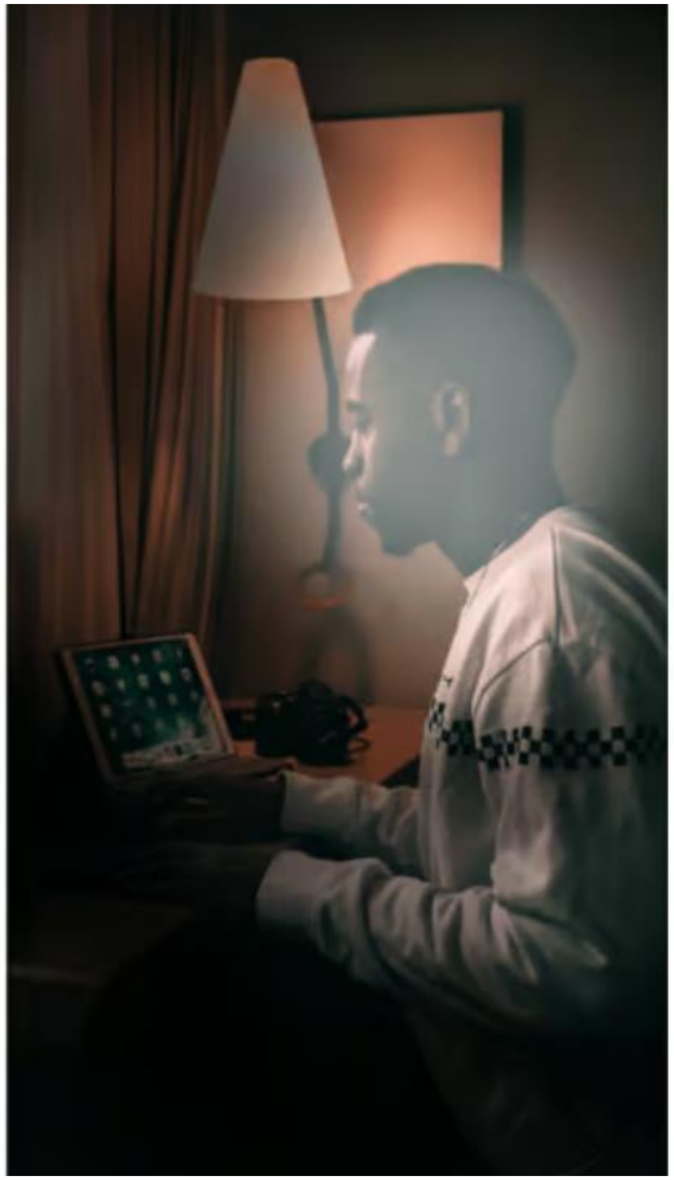
- The deeper a neural network is, the **harder it is to train**. For recurrent neural networks, the longer the sequence is, the deeper the neural network is along the time dimension.
- This results in **vanishing gradients**, where the gradient signal from the objective that the recurrent neural network learns from disappears as it travels backward.
- Even with RNNs specifically made to help prevent vanishing gradients, such as the **LSTM**, this is still a fundamental problem.
- **Bottleneck problem** refers to the issue where the entire input sequence needs to be compressed into a single fixed-size context vector. This context vector is then used by the decoder to generate the output sequence.
- Limited Parallelism

Difficulty with Long Sequences

- The bottleneck issue is particularly pronounced with long input sequences.
- The model might struggle to retain and use all relevant information from the beginning to the end of the sequence.
- SOLUTION : ATTENTION MECHANISM!!!!!

ATTENTION MECHANISM – EXPLAINED SIMPLE!!

- Humans do not tend to pay attention sequentially.
- We don't typically start looking at a picture from a corner—our focus tends to go to the important features first.
- For example, when looking at a picture, we often look at the objects in the middle.
- Humans pay attention in two ways, that is, conscious attention and unconscious attention.
- We would like our neural networks to model the behavior of conscious attention so they can pay attention to the important parts of the input first.
- This is done by **attention**, more commonly known as the **attention mechanism**. It is like blurring out the less important features.



Applications of Attention Mechanism

The attention mechanism is one of the most important concepts in deep learning. It has several applications. Some of them are given below:

- Image caption generation
- Image-based analysis
- Action recognition
- Text classification
- Machine translation
- Speech recognition
- Recommendation systems

ATTENTION MECHANISM

- The attention mechanism is crucial in many modern neural network architectures, especially in tasks involving sequences like natural language processing and computer vision.
- Its primary function is to allow the model to dynamically focus on different parts of the input sequence while processing the output sequence.

Key components – Q,K and V

- **Q (Query):** The word (or token) for which we are making the prediction.
- **K (Key):** Represents each word (or token) in the input sequence.
- **V (Value):** Represents the associated information or embedding of each word.

Analogy

- Q: “What is the next word in this sentence?”
- K: Each word in the sentence tries to provide an answer.
- V: The actual word representations (values) that provide the relevant information.

Example

Let's consider the sentence: "The dog barked loudly."

- **Query (Q)**: Suppose the current word being predicted is "barked."
- **Keys (K)**: Every word in the sentence ["The", "dog", "barked", "loudly"] will have a key.
- **Values (V)**: These are the word embeddings of each word, representing the information the model uses.

Example

- The query "barked" will compare its query vector with the key vectors of all the words.
- Based on the similarity, attention scores will be computed, and the values (word embeddings) will be weighted according to those scores to compute the final output.

ATTENTION MECHANISM- HOW IT WORKS?

- In the simplest terms, the attention mechanism works by calculating the weighted sum.
- This weighted sum tells the model about the importance of features.
- The process of calculating these weights is crucial and may vary for different applications.
- Here, we explain the attention mechanism in a **general sense** which is the same for most of the applications.

ATTENTION MECHANISM- HOW IT WORKS?

The working of the attention mechanism is divided into two parts:

- Calculating the **attention distribution** from the input.
- Calculating the **context vector** using the attention distribution.

ATTENTION MECHANISM- HOW IT WORKS?

- The **attention distribution** is what gives weightage to different inputs.
- Before calculating this, we first encode our inputs using a neural network. This encoded representation of the input is called **keys (*K*)**.
- Another thing used to calculate attention distribution is a **query** (*q*). It is a task-related representation that can be a vector or a matrix depending upon the task.

ATTENTION MECHANISM- HOW IT WORKS?

- A neural network computes the correlation between the keys and the query by using the **score function** *f*.
- The score function explains how keys and queries are related by giving us an **energy score OR attention score**.

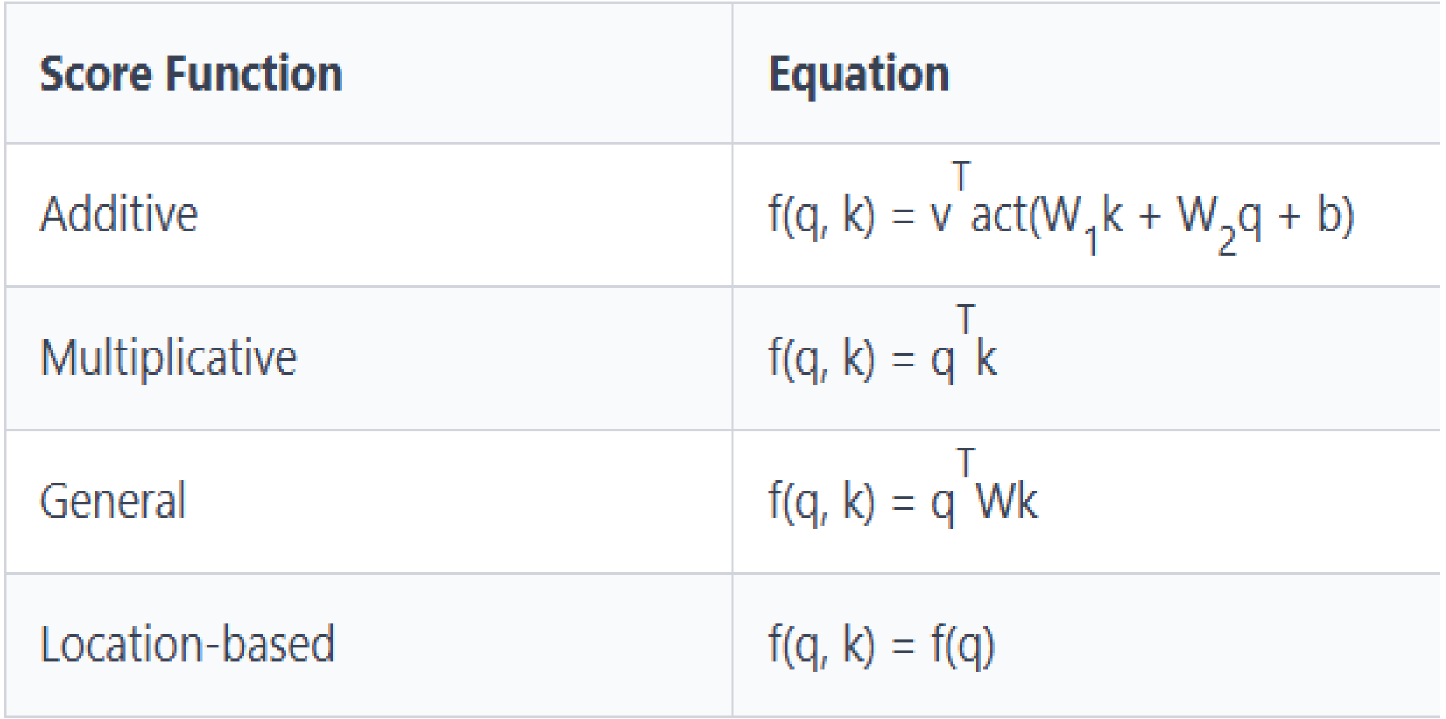
$$e=f(q,K)$$

Score Functions

- The selection of a good score function is really important.
- The two of the most common score functions are **additive attention** and **multiplicative attention**.
- Some of these score functions incorporate learnable parameters.

Score Functions

- The table below summarizes some of the common score functions.



Score Function	Equation
Additive	$f(q, k) = v^T act(W_1k + W_2q + b)$
Multiplicative	$f(q, k) = q^T k$
General	$f(q, k) = q^T Wk$
Location-based	$f(q, k) = f(q)$

- Here, *v*,*W*,*W*1,*W*2,*b* are the learnable parameters and *act* is the activation function.

Attention Weights

- The energy or attention scores are used to compute α attention weights by using a distribution function g.
- This distribution function g varies according to the application in which we are using the attention mechanism.

$$\alpha = g(e)$$

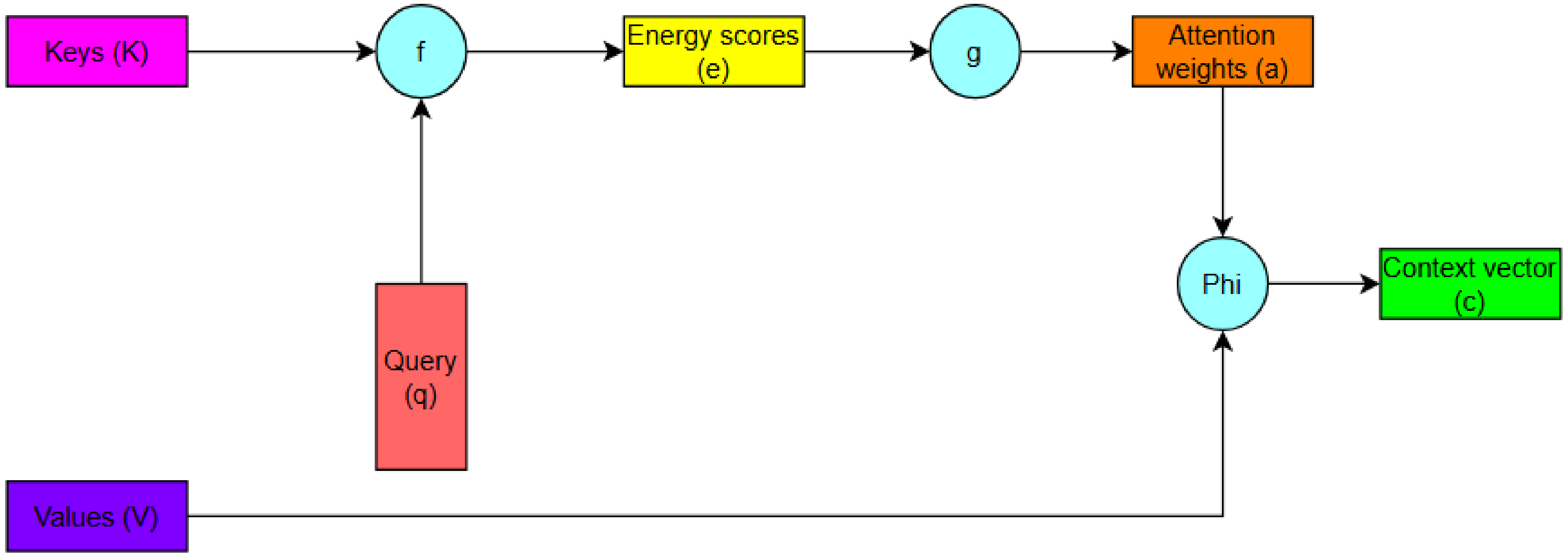
Values & Context Vector

- Now, we calculate the **context vector** *c*.
- Before that, we need another vector to introduce a new vector representation *V* called **values**.
- This vector has a one-on-one correspondence with the keys *K*. In some cases, the vector *V* is the same as *K*.
- The context vector *c* is calculated as:

$$c=\phi(\alpha,V)$$

- Here, ϕ is a function that returns a single vector which we refer to as the context vector.
- In most cases, ϕ performs the weighted sum. Our neural networks will now use this context vector to make predictions.

ATTENTION MECHANISM – WORKING!



The working of attention mechanism

ATTENTION MECHANISM – WORKING!

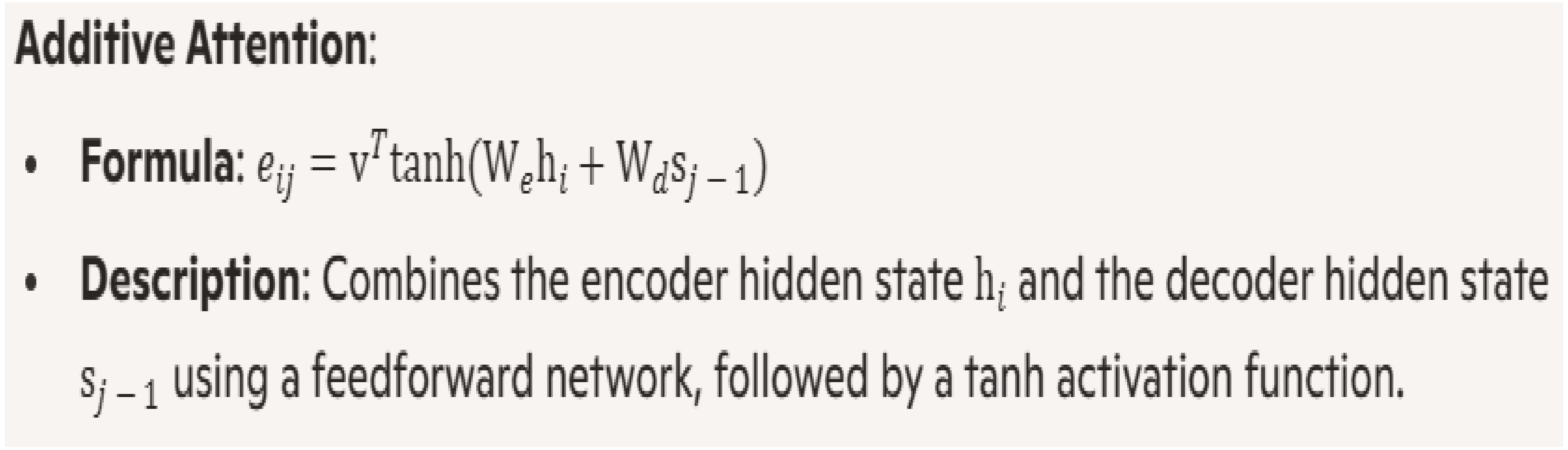
- In a nutshell, the attention mechanism computes a set of attention weights that determine the importance of different parts of the input sequence for generating each output token.
- These weights are used to create a weighted sum of the input features, which forms a dynamic context vector for each decoding step.

Summary of Steps involved in Attention Mechanism

- **Calculate Attention Scores:** For each output step, calculate the relevance (attention scores) of each input token based on the current state of the decoder.
- **Compute Attention Weights:** Normalize the attention scores to get attention weights (usually done using a softmax function).
- **Weighted Sum of Inputs:** Use the attention weights to compute a weighted sum of the encoder's hidden states. This results in a dynamic context vector.
- **Generate Output:** The decoder uses this dynamic context vector to generate the next output token.

TYPES OF ATTENTION MECHANISM

- **Additive (Bahdanau) Attention**
- **Introduced by**: Bahdanau et al., 2014
- **Description**: Computes the attention scores by combining the encoder hidden states and the decoder hidden state using a feedforward network. The combined vector is then passed through a tanh activation function.
- **Usage**: Common in machine translation and other sequence generation tasks.



Additive Attention:

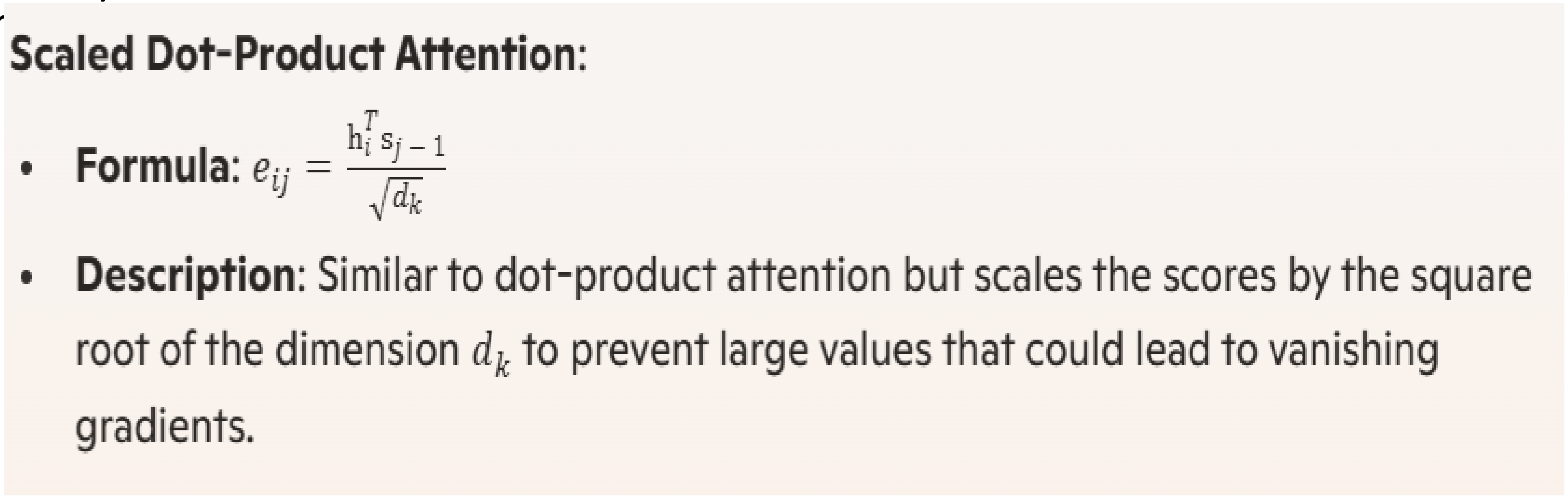
- **Formula:** $e_{ij} = \mathrm{v}^T \tanh(\mathrm{W}_e \mathrm{h}_i + \mathrm{W}_d \mathrm{s}_{j-1})$
- **Description:** Combines the encoder hidden state h_i and the decoder hidden state s_{j-1} using a feedforward network, followed by a tanh activation function.

TYPES OF ATTENTION MECHANISM

Multiplicative (Dot-Product) Attention

- **Introduced by**: Luong et al., 2015
- **Description**: Calculates attention scores by taking the dot product of the encoder hidden states and the decoder hidden state. It's computationally more efficient than additive attention.
- **Variants**:

  Scaled Dot-Product Attention: Scales the dot product by the square root of the dimension of the queries to prevent excessively large values.
- **Usage**: Widely used in various NLP tasks and the foundation of the attention mechanism in the Transfor



Scaled Dot-Product Attention:

- **Formula:** $e_{ij} = \frac{\mathrm{h}_i^T \mathrm{s}_{j-1}}{\sqrt{d_k}}$
- **Description:** Similar to dot-product attention but scales the scores by the square root of the dimension d_k to prevent large values that could lead to vanishing gradients.

TYPES OF ATTENTION MECHANISM

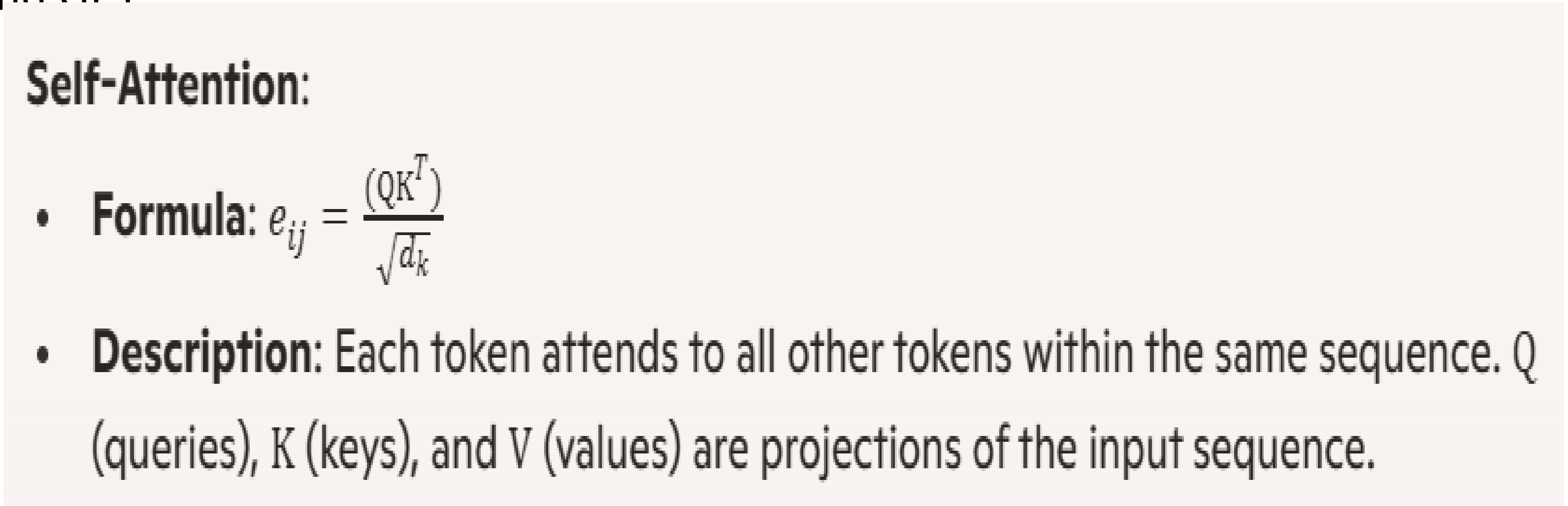
Single Headed Attention:

- In single-headed attention, the model only uses one set of **Q**, **K**, and **V** vectors to compute attention scores. Essentially, it's like attending to the entire sequence with a single focus point.
- This can work for simple tasks where one view of the data is enough. However, it has limitations when more complex relationships need to be captured between the tokens in the sequence.

TYPES OF ATTENTION MECHANISM

Self-Attention

- **Description**: Allows the model to relate different positions of a single sequence to compute its representation. Each token in the sequence attends to every other token.
- **Key Feature**: Enables parallelization and captures long-range dependencies more effectively.
- **Usage**: Integral to the Transformer model, which is the backbone of models like BERT and GPT



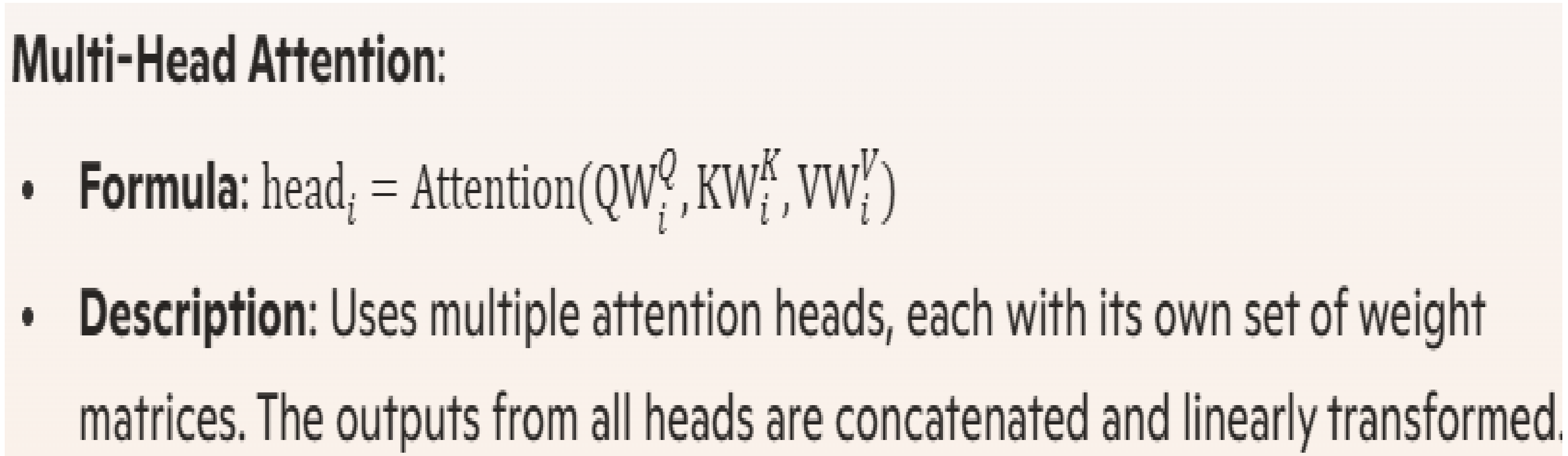
Self-Attention:

- **Formula:** $e_{ij} = \frac{(QK^T)}{\sqrt{d_k}}$
- **Description:** Each token attends to all other tokens within the same sequence. Q (queries), K (keys), and V (values) are projections of the input sequence.

TYPES OF ATTENTION MECHANISM

Multi-Head Attention

- **Introduced by**: Vaswani et al., 2017 (Transformer model)
- Multi-headed attention allows the model to attend to different parts of the sequence with multiple sets of **Q**, **K**, and **V**. This means that the model has multiple "attention heads" that can each focus on different aspects of the sequence.
- Extends self-attention by using multiple attention heads. Each head independently computes attention, and their outputs are concatenated and linearly transformed.
- **Advantage**: Allows the model to focus on different parts of the sequence simultaneously, capturing more diverse information and complex relationships in the sequence.
- **Usage**: A key component of the Transformer architecture.



Multi-Head Attention:

- **Formula:** $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$
- **Description:** Uses multiple attention heads, each with its own set of weight matrices. The outputs from all heads are concatenated and linearly transformed.

SELF ATTENTION PROCESS

- In **self-attention**, each word (or token) in the sequence is compared with all other words in the sequence, including itself, to determine the relevance (or "attention") between them.
- The key idea is that **each token computes attention scores with every other token** in the sequence, allowing it to aggregate information from the entire sequence.
- This is unlike traditional RNNs or LSTMs, where each word’s representation is updated sequentially based on previous words.

SELF ATTENTION PROCESS

Example Sentence:

Input Sentence: "The cat sat on the mat."

1. **Tokenization**: First, we tokenize the sentence into words (tokens).
2. **Tokens**: ["The", "cat", "sat", "on", "the", "mat"]
3. **Embedding**: Each word is mapped to a vector (embedding). For simplicity, let's assume that the words are represented by 3-dimensional vectors:

"The" → [0.1, 0.2, 0.3]

"cat" → [0.4, 0.5, 0.6]

"sat" → [0.7, 0.8, 0.9]

"on" → [1.0, 1.1, 1.2]

"the" → [1.3, 1.4, 1.5]

"mat" → [1.6, 1.7, 1.8]

- **Query, Key, and Value Vectors**: For each token, we create three vectors: the **Query (Q)**, **Key (K)**, and **Value (V)**. These vectors are typically created by multiplying the token embeddings by learned weight matrices.

SELF ATTENTION PROCESS

- Query: Represents the current word (token) and asks for "what to look for" in the other words.
- Key: Represents all words in the sequence and acts as a reference point for comparison.
- Value: Contains the actual information associated with each token that will be used to compute the output.

SELF ATTENTION PROCESS

In the self-attention mechanism, for each word:

- The **Query** vector is compared with all **Key** vectors.
- The similarity between the Query and the Key determines how much attention should be given to each token's Value.

Calculating Attention Scores:

- For each Query vector, calculate the **dot product** between the Query and each of the Key vectors.
- This measures how much focus or "attention" a word should have on other words. The result is a vector of attention scores.

SELF ATTENTION PROCESS

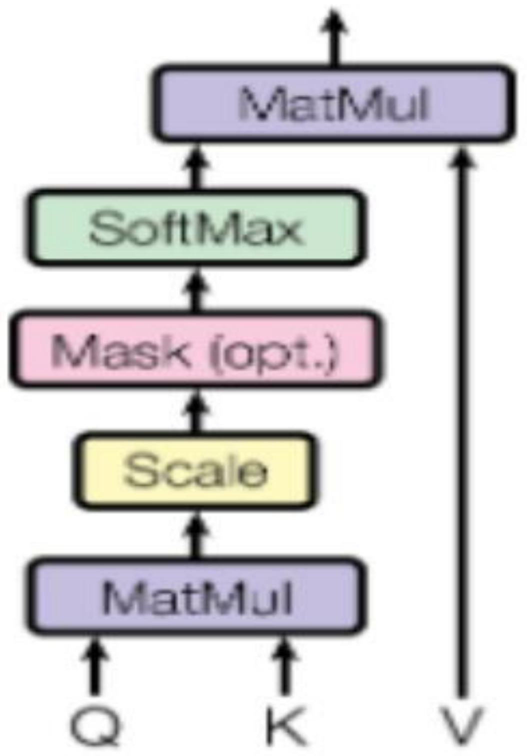
- **Dot product** for each Query and all Keys:
  - Attention score between "The" and "The", "The" and "cat", "The" and "sat", etc.
  - For example:

$$\text{Attention score}\ (\text{The}, \text{The}) = Q_{\text{The}} \cdot K_{\text{The}}$$

$$\text{Attention score}\ (\text{The}, \text{cat}) = Q_{\text{The}} \cdot K_{\text{cat}}$$

This calculation is repeated for all tokens with respect to all others.

SELF ATTENTION PROCESS



SELF ATTENTION PROCESS

- **Scaling the Scores**: If the vectors are high-dimensional (which they often are), the dot product can be very large.
- To prevent the model from becoming unstable, the dot product is often **scaled** by dividing it by the square root of the dimension of the Key vectors $\sqrt{d_k}$.
- This scaling helps to stabilize gradients during training and avoid excessively large values in the **softmax** step.

SELF ATTENTION PROCESS

- **Softmax**: Once the scores are computed, a **softmax** function is applied to the scores to normalize them. This turns the raw scores into probabilities that sum up to 1.
- Attention Weights=softmax(Attention Scores)
- These attention weights indicate the degree of focus or importance the model places on each word in the sequence when computing the output for a given word.

SELF ATTENTION PROCESS

- **Weighted Sum of Values**: Now that we have the attention weights, we use them to calculate a weighted sum of the **Value** vectors. This produces the output of the self-attention mechanism.
- For each token, the weighted sum of the Values is calculated using the attention weights:
- Output=$\sum(Attention\ weight_i)\,.\,v_i$
- This output is the **contextual representation** of the word, taking into account the entire sequence or the **context vector.**

MODELS THAT USE ATTENTION MECHANISM

- Attention is used extensively in Natural Language Processing (NLP). However, it is not limited to NLP, it is also used in computer vision.
- Some famous models that use the attention mechanism are as follows:
  - **GPT-3:** Generative Pretrained Transformers version 3, GPT-3, is the transformer-based model that is trained to not only generate text, but to summarize large documents. It is also capable of responding to conversational tasks. Another surprising feature is that it can write code from comments.
  - **The show, attend, and tell model:** This model was trained to generate captions for images.
  - **ViT:** Vision Transformers, were introduced by Google's research team. This model outperformed the conventional ResNets for computer vision tasks.

THANK YOU!