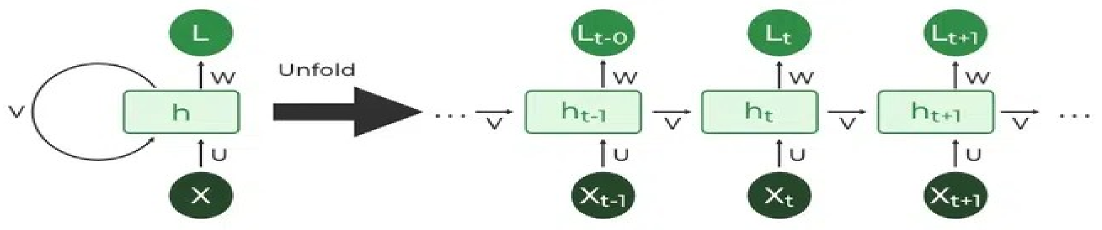
UNIT FOUR NOTES(PART I)

- **RECURRENT NEURAL NETWORK**
- **LONG SHORT TERM MEMORY**
- **ATTENTION MECHANISM**
- **TRANSFORMER BASED MODEL**
- **SELF ATTENTION**
- **MULTI HEADED ATTENTION**

RECURRENT NEURAL NETWORK

Recurrent Neural Networks (RNNs) work a bit different from regular neural networks. In neural network the information flows in one direction from input to output. However in RNN information is fed back into the system after each step. Think of it like reading a sentence, when you're trying to predict the next word you don't just look at the current word but also need to remember the words that came before to make accurate guess.

RNNs allow the network to "remember" past information by feeding the output from one step into next step. This helps the network understand the context of what has already happened and make better predictions based on that. For example when predicting the next word in a sentence the RNN uses the previous words to help decide what word is most likely to come next.

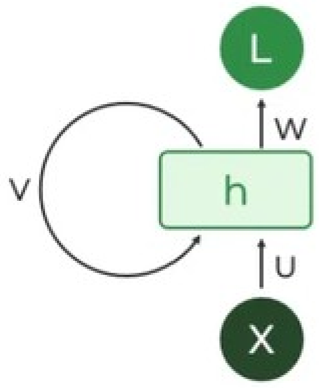


Basic architecture of RNN and the feedback loop mechanism where the output is passed back as input for the next time step.

Key Components of RNNs

1. Recurrent Neurons

The fundamental processing unit in RNN is a **Recurrent Unit.** Recurrent units hold a hidden state that maintains information about previous inputs in a sequence. Recurrent units can "remember" information from prior steps by feeding back their hidden state, allowing them to capture dependencies across time.

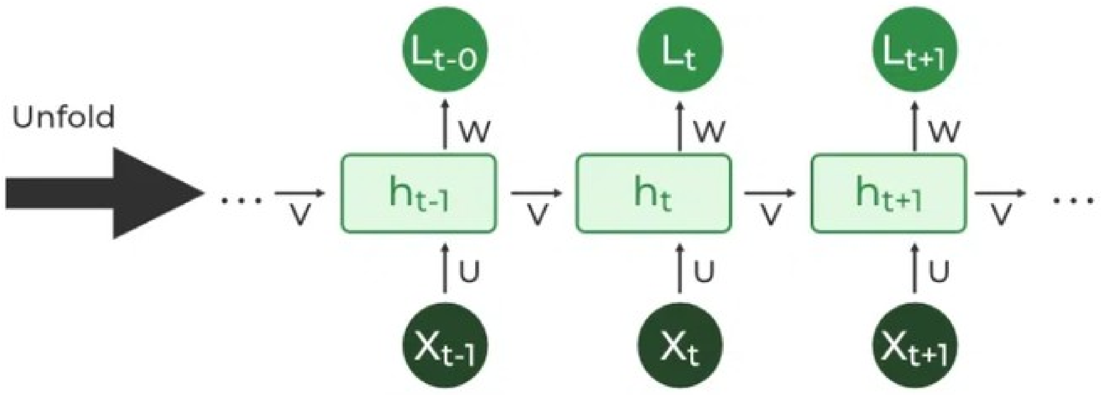


Recurrent Neuron

2. RNN Unfolding

RNN unfolding or unrolling is the process of expanding the recurrent structure over time steps. During unfolding each step of the sequence is represented as a separate layer in a series illustrating how information flows across each time step.

This unrolling enables backpropagation through time (BPTT) a learning process where errors are propagated across time steps to adjust the network's weights enhancing the RNN's ability to learn dependencies within sequential data.



RNN Unfolding

Recurrent Neural Network Architecture
RNNs share similarities in input and output structures with other deep learning architectures but differ significantly in how information flows from input to output. Unlike traditional deep neural networks, where each dense layer has distinct weight matrices, RNNs use shared weights across time steps, allowing them to remember information over sequences.

In RNNs, the hidden state Hiis calculated for every input Xi to retain sequential dependencies. The computations follow these core formulas:

1. Hidden State Calculation:

$h=\sigma(U\cdot X+W\cdot h_{t-1}+B)$

Here, hh represents the current hidden state, UU and WW are weight matrices, and BB is the bias.

2. Output Calculation:

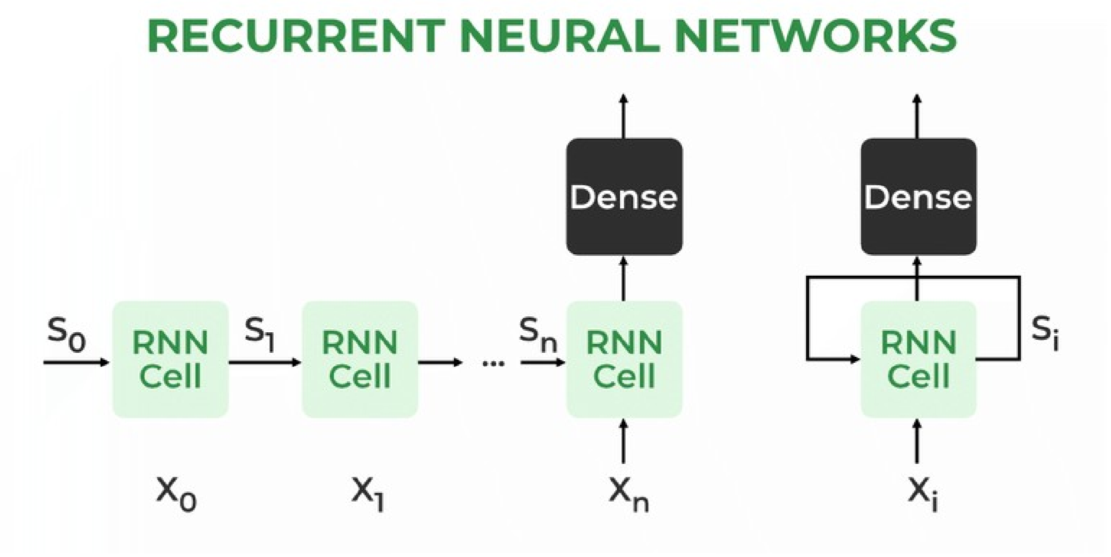
$Y=O(V\cdot h+C)$

The output YY is calculated by applying OO, an activation function, to the weighted hidden state, where VV and CC represent weights and bias.

3. Overall Function:

$$Y=f(X,h,W,U,V,B,C)\,Y=f(X,h,W,U,V,B,C)$$

This function defines the entire RNN operation, where the state matrix SS holds each element si representing the network's state at each time step i



Recurrent Neural Architecture

How does RNN work?
At each time step RNNs process units with a fixed activation function. These units have an internal hidden state that acts as memory that retains information from previous time steps. This memory allows the network to store past knowledge and adapt based on new inputs.

Updating the Hidden State in RNNs

The current hidden state h_t depends on the previous state h_{t-1} and the current input xtxt, and is calculated using the following relations:

1. **State Update**:

$$h_t=f(h_{t-1},x_t)$$

where:

- h_t is the current state
- h_{t-1} is the previous state
- x_t is the input at the current time step

2. Activation Function Application:

$$h_t=\tanh(W_{hh}\cdot h_{t-1}+W_{xh}\cdot x_t)$$

Here, $W_{hh}$is the weight matrix for the recurrent neuron, and W_{xh} is the weight matrix for the input neuron.

3. Output Calculation:

$$y_t=W_{hy}\cdot h_t$$

where $y_t$is the output and W_{hy} is the weight at the output layer.

*T*hese parameters are updated using backpropagation. However, since RNN works on sequential data here we use an updated backpropagation which is known as backpropagation through time.

LONG SHORT TERM MEMORY

Long Short-Term Memory (LSTM) is an enhanced version of the Recurrent Neural Network (RNN) designed by Hochreiter & Schmidhuber. LSTMs can capture long-term dependencies in sequential data making them ideal for tasks like language translation, speech recognition and time series forecasting.

Unlike traditional RNNs which use a single hidden state passed through time LSTMs introduce a memory cell that holds information over extended periods addressing the challenge of learning long-term dependencies.

Problem with Long-Term Dependencies in RNN

Recurrent Neural Networks (RNNs) are designed to handle sequential data by maintaining a hidden state that captures information from previous time steps. However they often face challenges in learning long-term dependencies where information from distant time steps becomes crucial for making accurate predictions for current state. This problem is known as the vanishing gradient or exploding gradient problem.

Vanishing Gradient: When training a model over time, the gradients (which help the model learn) can shrink as they pass through many steps. This makes it hard for the model to learn long-term patterns since earlier information becomes almost irrelevant.

Exploding Gradient: Sometimes, gradients can grow too large, causing instability. This makes it difficult for the model to learn properly, as the updates to the model become erratic and unpredictable.

Both of these issues make it challenging for standard RNNs to effectively capture long-term dependencies in sequential data.

LSTM Architecture

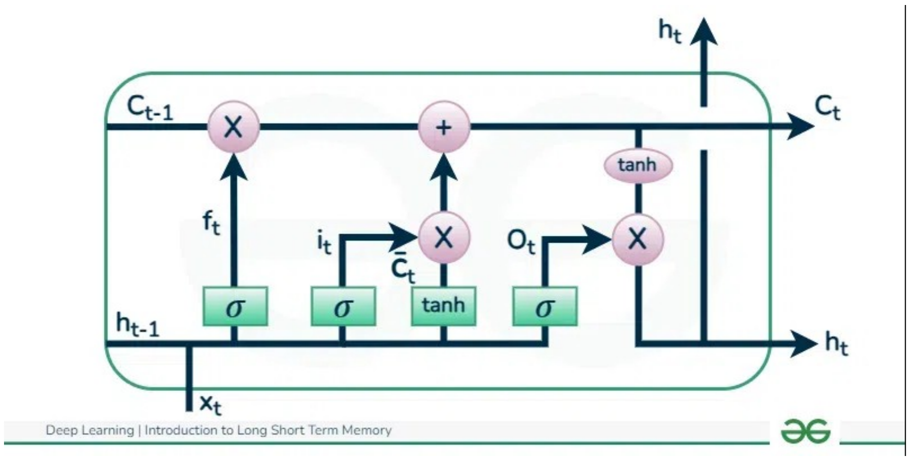
LSTM architectures involves the memory cell which is controlled by three gates: the input gate, the forget gate and the output gate. These gates decide what information to add to, remove from and output from the memory cell.

- **Input gate**: Controls what information is added to the memory cell.
- **Forget gate**: Determines what information is removed from the memory cell.
- **Output gate**: Controls what information is output from the memory cell.

This allows LSTM networks to selectively retain or discard information as it flows through the network which allows them to learn long-term dependencies. The network has a hidden state which is like its short-term memory. This memory is updated using the current input, the previous hidden state and the current state of the memory cell.

Working of LSTM

LSTM architecture has a chain structure that contains four neural networks and different memory blocks called **cells**.



Information is retained by the cells and the memory manipulations are done by the **gates.** There are three gates –

Forget Gate

The information that is no longer useful in the cell state is removed with the forget gate. Two inputs *xt* (input at the particular time) and *ht-1* (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.

The equation for the forget gate is:

ft=σ(Wf⋅[ht−1,xt]+bf) $f_t=\sigma(W_f \cdot [h_{t-1},x_t]+b_f)$
where:

- W_f represents the weight matrix associated with the forget gate.
- [h_t-1, x_t] denotes the concatenation of the current input and the previous hidden state.
- b_f is the bias with the forget gate.
- σ is the sigmoid activation function.

Input gate

The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs *ht-1* and *xt*. . Then, a vector is created using *tanh* function that gives an output from -1 to +1, which contains all the possible values from ht-1 and *xt*. At last, the values of the vector and the regulated values are multiplied to obtain the useful information. The equation for the input gate is:

$$_t=\sigma(W_i \cdot [h_{t-1},x_t]+b_i)$$

C^t=tanh(Wc⋅[ht−1,xt]+bc) $\hat{C}_t=tanh(W_c \cdot [h_{t-1},x_t]+b_c)$

We multiply the previous state by ft, disregarding the information we had previously chosen to ignore. Next, we include it∗Ct. This represents the updated candidate values, adjusted for the amount that we chose to update each state value.

$$C_t = f_t \odot C_{t-1} + i_t \odot \hat{C}_t$$

where

- ⊙ denotes element-wise multiplication
- tanh is tanh activation function

Output gate

The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs ht−1h_{t-1} and xtx_t. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.

Applications of LSTM

Some of the famous applications of LSTM includes:

- **Language Modeling**: Used in tasks like language modeling, machine translation and text summarization. These networks learn the dependencies between words in a sentence to generate coherent and grammatically correct sentences.
- **Speech Recognition**: Used in transcribing speech to text and recognizing spoken commands. By learning speech patterns they can match spoken words to corresponding text.
- **Time Series Forecasting**: Used for predicting stock prices, weather and energy consumption. They learn patterns in time series data to predict future events.
- **Anomaly Detection**: Used for detecting fraud or network intrusions. These networks can identify patterns in data that deviate drastically and flag them as potential anomalies.
- **Recommender Systems**: In recommendation tasks like suggesting movies, music and books. They learn user behavior patterns to provide personalized suggestions.
- **Video Analysis**: Applied in tasks such as object detection, activity recognition and action classification. When combined with Convolutional Neural Networks (CNNs) they help analyze video data and extract useful information.

LTSM vs RNN

Feature	LSTM (Long Short-term Memory)	RNN (Recurrent Neural Network)
Memory	Has a special memory unit that	Does not have a memory unit

Feature	LSTM (Long Short-term Memory)	RNN (Recurrent Neural Network)
	allows it to learn long-term dependencies in sequential data	
Directionality	Can be trained to process sequential data in both forward and backward directions	Can only be trained to process sequential data in one direction
Training	More difficult to train than RNN due to the complexity of the gates and memory unit	Easier to train than LSTM
Long-term dependency learning	Yes	Limited
Ability to learn sequential data	Yes	Yes
Applications	Machine translation, speech recognition, text summarization, natural language processing, time series forecasting	Natural language processing, machine translation, speech recognition, image processing, video processing

ATTENTION MECHANISM

An attention mechanism is an Encoder-Decoder kind of neural network architecture that allows the model to focus on specific sections of the input while executing a task. It dynamically assigns weights to different elements in the input, indicating their relative importance or relevance. By incorporating attention, the model can selectively attend to and process the most relevant information, capturing dependencies and relationships within the data. This mechanism is particularly valuable in tasks involving sequential or structured data, such as natural language processing or computer vision, as it enables the model to effectively handle long-range dependencies and improve performance by selectively attending to important features or contexts.

Recurrent models of visual attention use reinforcement learning to focus attention on key areas of the image. A recurrent neural network governs the peek network, which dynamically selects particular locations for exploration over time. In classification tasks, this method outperforms convolutional neural networks. Additionally, this framework goes beyond image identification and may be used for a variety of visual reinforcement learning applications, such as helping robots choose behaviours to accomplish particular goals. Although the most basic use of this strategy is supervised learning, the use of reinforcement learning permits more adaptable and flexible decision-making based on feedback from past glances and rewards earned throughout the learning process.

The application of attention mechanisms to image captioning has substantially enhanced the quality and accuracy of generated captions. By incorporating attention, the model learns to focus on pertinent image regions while creating each caption word. The model can synchronize the visual and textual modalities by paying attention to various areas of the image at each time step thanks to the attention mechanism. By focusing on important objects or areas in the image, the model is able to produce captions that are more detailed and contextually appropriate. The attention-based image captioning models have proven to perform better at catching minute details, managing complicated scenes, and delivering cohesive and educational captions that closely match the visual material.

The attention mechanism is a technique used in machine learning and natural language processing to increase model accuracy by focusing on relevant data. It enables the model to focus on certain areas of the input data, giving more weight to crucial features and disregarding unimportant ones. Each input attribute is given a weight based on how important it is to the output in order to accomplish this. The performance of tasks requiring the utilization of the attention mechanism has significantly improved in areas including speech recognition, image captioning, and machine translation.

How Attention Mechanism Works

An attention mechanism in a neural network model typically consists of the following steps:

1. **Input Encoding:** The input sequence of data is represented or embedded using a collection of representations. This step transforms the input into a format that can be processed by the attention mechanism.
2. **Query Generation:** A query vector is generated based on the current state or context of the model. This query vector represents the information the model wants to focus on or retrieve from the input.
3. **Key-Value Pair Creation:** The input representations are split into key-value pairs. The keys capture the information that will be used to determine the importance or relevance, while the values contain the actual data or information.
4. **Similarity Computation:** The similarity between the query vector and each key is computed to measure their compatibility or relevance. Different similarity metrics can be used, such as dot product, cosine similarity, or scaled dot product.

5.Attention Weights Calculation: The similarity scores are passed through a softmax function to obtain attention weights. These weights indicate the importance or relevance of each key-value pair.
Attention Weight (α(s,i))=softmax(Similarity Scores(s,i))Attention Weight ($\alpha(s,i)$)=softmax(Similarity Scores(s,i))
6.Weighted Sum: The attention weights are applied to the corresponding values, generating a weighted sum. This step aggregates the relevant information from the input based on their importance determined by the attention mechanism.
7.Context Vector: The weighted sum serves as a context vector, representing the attended or focused information from the input. It captures the relevant context for the current step or task.
8.Integration with the Model: The context vector is combined with the model's current state or hidden representation, providing additional information or context for subsequent steps or layers of the model.
9. Repeat: Steps 2 to 8 are repeated for each step or iteration of the model, allowing the attention mechanism to dynamically focus on different parts of the input sequence or data.

By incorporating an attention mechanism, the model can effectively capture dependencies, emphasize important information, and adaptively focus on different elements of the input, leading to improved performance in tasks such as machine translation, text summarization, or image recognition.

TRANSFORMER BASED MODEL

Transformer is a neural network architecture used for performing machine learning tasks particularly in natural language processing (NLP) and computer vision. In 2017 Vaswani et al. published a paper *" Attention is All You Need"* in which the transformers architecture was introduced. The article explores the architecture, workings and applications of transformers.

Need For Transformers Model in Machine Learning

Transformer Architecture is a model that uses **self-attention** to transform one whole sentence into a single sentence. This is useful where older models work step by step and it helps overcome the challenges seen in models like RNNs and LSTMs. Traditional models like RNNs (Recurrent Neural Networks) suffer from the **vanishing gradient** problem which leads to long-term memory loss. RNNs process text sequentially meaning they analyze words one at a time.

For example, in the sentence: "XYZ went to France in 2019 when there were no cases of COVID and there he met the president of that country" the word "that country" refers to "France".

However RNN would struggle to link "that country" to "France" since it processes each word in sequence leading to losing context over long sentences. This limitation prevents RNNs from understanding the full meaning of the sentence.

While adding more memory cells in LSTMs (Long Short-Term Memory networks) helped address the vanishing gradient issue they still process words one by one. This sequential processing means LSTMs can't analyze an entire sentence at once.

For instance the word "point" has different meanings in these two sentences:

- "The needle has a sharp point." (Point = Tip)
- "It is not polite to point at people." (Point = Gesture)

Traditional models struggle with this context dependence, whereas, Transformer model through its self-attention mechanism, processes the entire sentence in parallel addressing these issues and making it significantly more effective at understanding context.

Architecture and Working of Transformers

1. Positional Encoding

Unlike RNNs transformers lack an inherent understanding of word order since they process data in parallel. To solve this Positional Encodings are added to token embeddings providing information about the position of each token within a sequence.

2. Position-wise Feed-Forward Networks

The Feed-Forward Networks consist of two linear transformations with a ReLU activation. It is applied independently to each position in the sequence.

Mathematically:

$$\text{FFN}(x)=\max(0,xW1+b1)W2+b2$$

This transformation helps refine the encoded representation at each position.

3. Attention Mechanism

The attention mechanism allows transformers to determine **which words in a sentence are most relevant** to each other. This is done using a **scaled dot-product attention** approach:

1. Each word in a sequence is **mapped to three vectors**:

- **Query (Q)**
- **Key (K)**
- **Value (V)**

2. Attention scores are computed as: Attention(Q,K,V)=softmax(($d_k QK_T$)V
3. These scores determine how much attention each word should pay to others.

Multi-Head Attention

Instead of using a single attention mechanism transformers apply **multi-head attention** where multiple attention layers run in parallel. This enables the model to **capture different types of relationships within the input**.

4. Encoder-Decoder Architecture

The encoder-decoder structure is key to transformer models. The encoder processes the input sequence into a vector, while the decoder converts this vector back into a sequence. Each encoder and decoder layer includes **self-attention** and **feed-forward layers**. In the decoder, an encoder-decoder attention layer is added to focus on relevant parts of the input.

> *For example, a French sentence **"Je suis étudiant"** is translated into **"I am a student"** in English.*

The **encoder** consists of multiple layers (typically **6 layers**). Each layer has **two main components**:

- **Self-Attention Mechanism** – Helps the model understand word relationships.
- **Feed-Forward Neural Network** – Further transforms the representation.

The **decoder** also consists of **6 layers**, but with an additional **encoder-decoder attention** mechanism. This allows the decoder to focus on **relevant parts of the input sentence** while generating output.

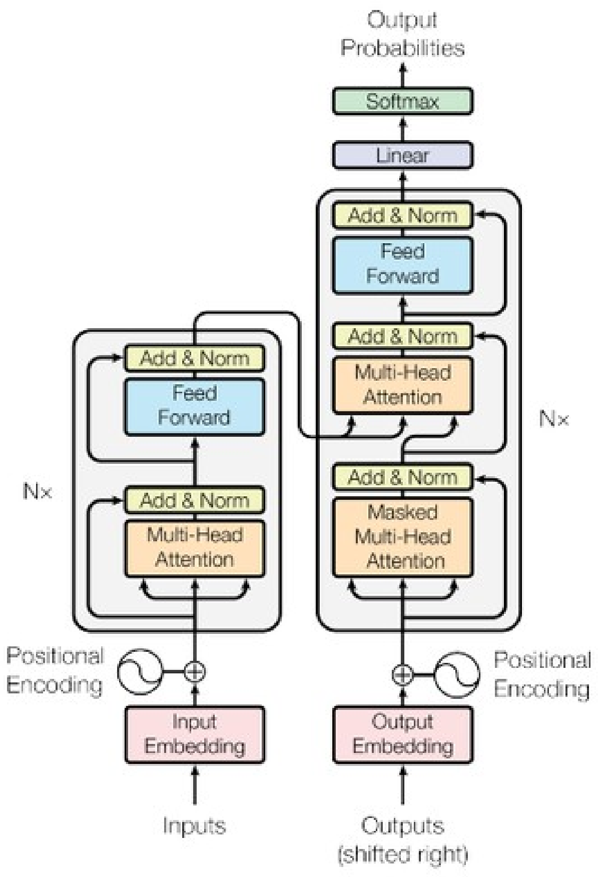


Figure 1: The Transformer - model architecture.

For instance in the sentence ***"The cat didn't chase the mouse, because it was not hungry",*** the word 'it' refers to 'cat'. The self-attention mechanism helps the model correctly associate 'it' with 'cat' ensuring an accurate understanding of sentence structure.

Applications of Transformers

Some of the applications of transformers are:

1. **NLP Tasks**: Transformers are used for machine translation, text summarization, named entity recognition and sentiment analysis.
2. **Speech Recognition**: They process audio signals to convert speech into transcribed text.
3. **Computer Vision**: Transformers are applied to image classification, object detection, and image generation.
4. **Recommendation Systems**: They provide personalized recommendations based on user preferences.
5. **Text and Music Generation**: Transformers are used for generating text (e.g., articles) and composing music.

SELF ATTENTION

Self-attention is a key mechanism in transformer-based models, allowing them to weigh different words in a sequence based on their relevance to each other. It helps the model understand contextual relationships within the input.

How Self-Attention Works

Input Representation:
Each word in a sequence is represented as an embedding (vector).

Query, Key, and Value Matrices:
The model transforms the input embeddings into three different representations using weight matrices:

1. **Query (Q):** Represents the current word trying to find relevant words.
2. **Key (K):** Represents all words that could be relevant.
3. **Value (V):** Represents the actual meaning or information stored in each word

Computing Attention Scores:
The attention scores are computed using the formula:

$$\text{Attention}=\text{softmax}(QK_T/\sqrt{D_K})V$$

1. QK_T computes the similarity between words.
2. $\sqrt{D_K}$ scales the scores to avoid large values.
3. **Softmax** normalizes the scores, making them sum to 1.

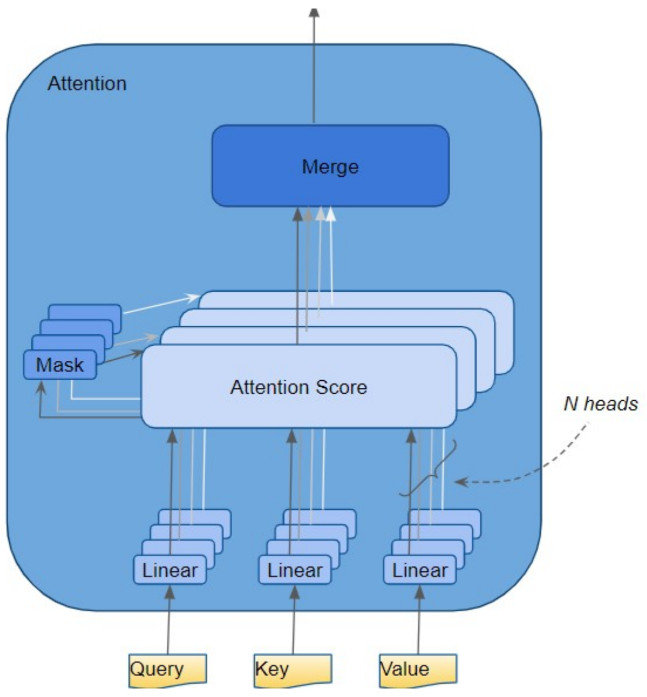
Weighted Sum:
The softmax scores are used to weight the values (VVV), determining how much focus each word should get in the final representation.

Why is Self-Attention Important?

- Captures long-range dependencies in a sentence.
- Assigns different importance to words dynamically.
- Enables parallel processing (unlike RNNs).

MULTI HEADED ATTENTION

In the Transformer, the Attention module repeats its computations multiple times in parallel. Each of these is called an Attention Head. The Attention module splits its Query, Key, and Value parameters N-ways and passes each split independently through a separate Head. All of these similar Attention calculations are then combined together to produce a final Attention score. This is called Multi-head attention and gives the Transformer greater power to encode multiple relationships and nuances for each word.



To understand exactly how the data is processed internally, let's walk through the working of the Attention module while we are training the Transformer to solve a translation problem. We'll use one sample of our training data which consists of an input sequence ('You are welcome' in English) and a target sequence ('De nada' in Spanish).

Uses of Multi-Headed Attention (MHA)

Multi-Headed Attention (MHA) is a fundamental component in Transformer models, providing multiple perspectives on input data. It enhances performance in various applications across NLP, computer vision, and more.

Natural Language Processing (NLP)

MHA is widely used in NLP tasks for better contextual understanding.

Machine Translation (e.g., Google Translate)

- Enables models like **Transformer, BERT, and T5** to align words across languages.
- Example: "I love AI" in English → "Me encanta la IA" in Spanish (MHA helps match corresponding words).

Text Summarization (e.g., ChatGPT, BART, Pegasus)

- Helps extract key information by attending to important parts of the input text.
- Example: Given a long article, MHA helps decide which sentences are most relevant.

Question Answering (e.g., BERT, GPT models)

- Improves comprehension by considering different parts of a passage simultaneously.

- Example: Finding the correct answer in a document by attending to both the question and relevant text sections.

Sentiment Analysis

- Captures long-range dependencies (e.g., “not bad at all” means **positive**, not **negative**).
- Helps models understand context better than traditional RNNs or CNNs.

Computer Vision (CV)

MHA is now used in vision models for image understanding.

Vision Transformers (ViTs)

- Instead of CNNs, ViTs use **self-attention** to analyze different regions of an image.
- Example: In image classification, MHA helps focus on important objects in an image.

Object Detection (DETR Model)

- Helps detect multiple objects in an image by attending to different parts simultaneously.
- Example: Identifying a cat and a dog in the same image.

Image Segmentation

- Helps identify and separate objects in an image.
- Example: Self-driving cars use MHA to detect pedestrians, lanes, and vehicles.

Speech Processing

MHA is also useful in speech and audio-related tasks.

Speech Recognition (e.g., Whisper, Wav2Vec2)

- Converts speech into text by attending to different sound frequencies.
- Example: Siri and Google Assistant use MHA to improve speech-to-text accuracy.

Text-to-Speech (TTS)

- Enhances voice synthesis models like Tacotron by capturing the right intonations.

Reinforcement Learning & Robotics

- Used in **game-playing AI** (e.g., AlphaStar, OpenAI Five) to analyze multiple strategies at once.
- Helps **robots** understand their surroundings in real-time decision-making tasks.

Healthcare & Bioinformatics

- Used in **drug discovery** (predicting protein interactions).
- Helps in **medical image analysis** (detecting diseases in scans).