A0597203 AI Business Applications

Introduction to Deep Learning

AI Business Applications

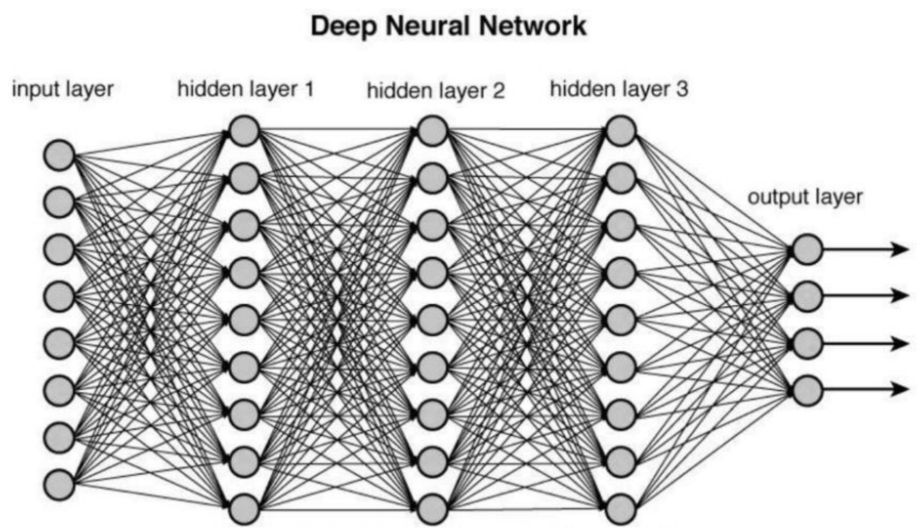
Introduction to Deep Learning

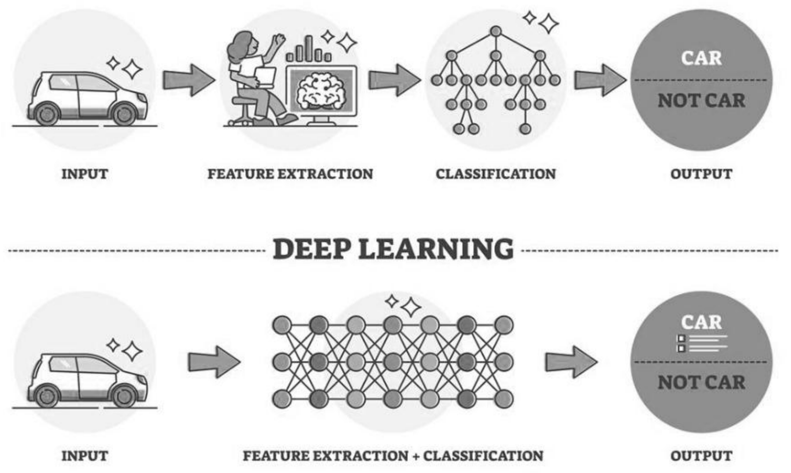
Content

- Introduction to Deep Neural Networks
    - CNNs
    - RNNs
    - The Transformer

Introduction to Deep Learning

- Neural Networks have revolutionized artificial intelligence by enabling machines to learn from data in ways that mimic human neural processes.
- Deep neural networks (DNNs) are Neural Networks that are composed of multiple processing layers that can learn representations of data with multiple levels of abstraction.
- The power of deep learning comes from its ability to automatically discover intricate patterns in raw data through the learning process, without requiring human engineers to manually specify all the knowledge needed by the computer system.





Introduction to Deep Learning

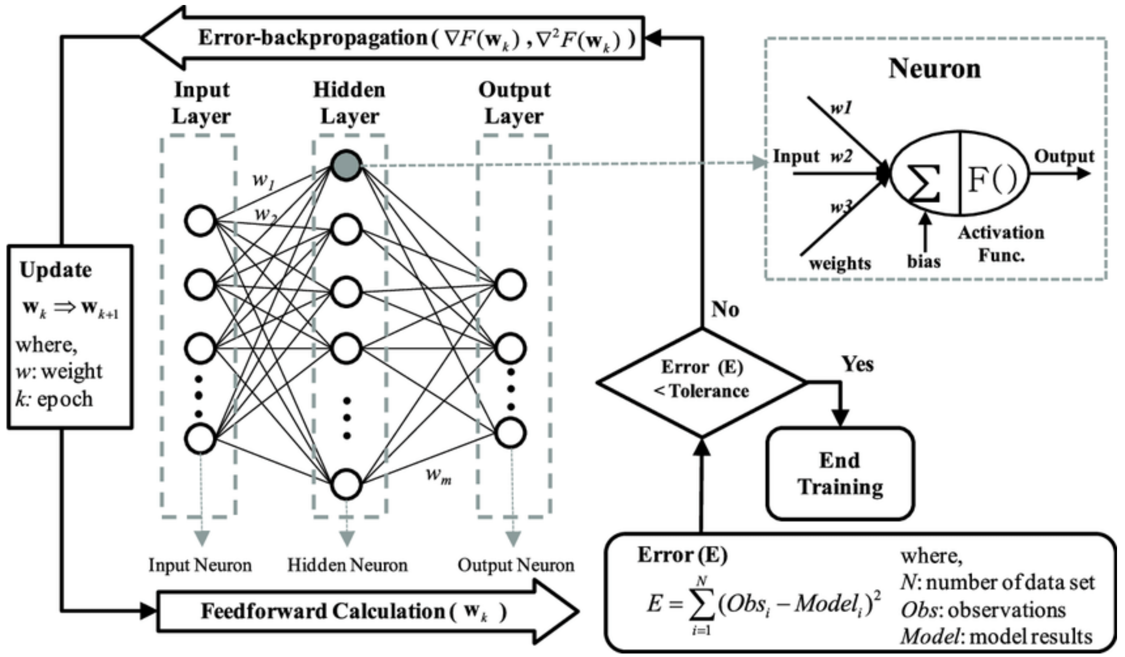
Fundamentals of Neural Networks

At their core, neural networks consist of:

1. **Neurons**: Mathematical functions that take inputs, apply weights, add a bias, and produce an output
2. **Layers**: Collections of neurons that process information in stages
3. **Activation Functions**: Non-linear functions that introduce complexity into the network
4. **Weights and Biases**: Parameters that are adjusted during training

The basic workflow involves:

- Forward propagation: Data flows through the network
- Loss calculation: The network's prediction is compared to the actual value
- Backpropagation: Errors are propagated backward to update weights
- Optimization: Weights are adjusted to minimize errors



Convolutional Neural Networks

CNNs revolutionized image processing by introducing specialized layers that mimic how the visual cortex processes information.

Key components include:

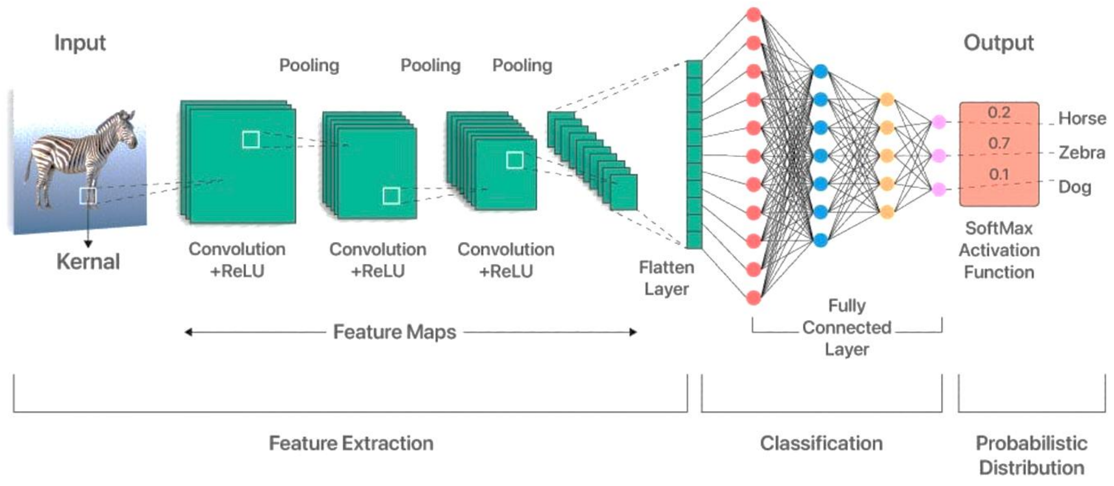
1. **Convolutional Layers**: Apply filters that scan across the input data to detect patterns
2. **Pooling Layers**: Reduce dimensions while preserving important features
3. **Fully Connected Layers**: Connect every neuron to every neuron in adjacent layers

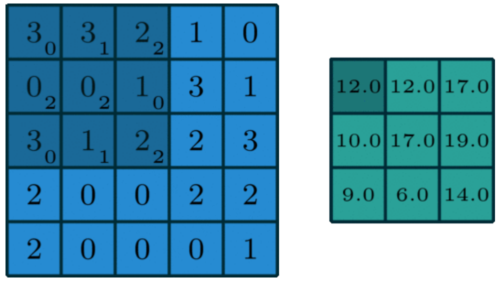
Instead of each neuron connecting to every pixel in an image (which would be computationally expensive), CNNs use:

- **Local connectivity**: Neurons connect only to nearby pixels
- **Parameter sharing**: The same filter is applied across the entire image

Business applications include:

- Product image recognition
- Visual quality control in manufacturing
- Document processing
- Customer behavior analysis in retail





$$\begin{array}{|c|c|c|c|c|}\hline 3_0 & 3_1 & 2_2 & 1 & 0 \\ \hline 0_2 & 0_2 & 1_0 & 3 & 1 \\ \hline 3_0 & 1_1 & 2_2 & 2 & 3 \\ \hline 2 & 0 & 0 & 2 & 2 \\ \hline 2 & 0 & 0 & 0 & 1 \\ \hline \end{array} \qquad \begin{array}{|c|c|c|}\hline 12.0 & 12.0 & 17.0 \\ \hline 10.0 & 17.0 & 19.0 \\ \hline 9.0 & 6.0 & 14.0 \\ \hline \end{array}$$

Recurrent Neural Networks (RNNs)

Unlike traditional neural networks, RNNs process sequences by maintaining a form of memory of previous inputs.

Key characteristics:

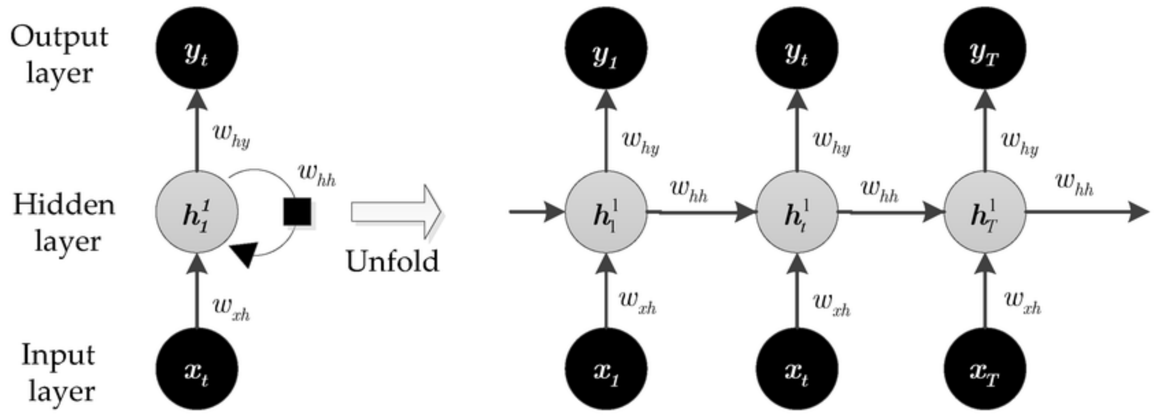
- **Time-dependent processing**: Output depends on both current and previous inputs
- **Shared parameters**: The same weights are applied at each time step
- **Memory**: Internal state acts as a form of short-term memory

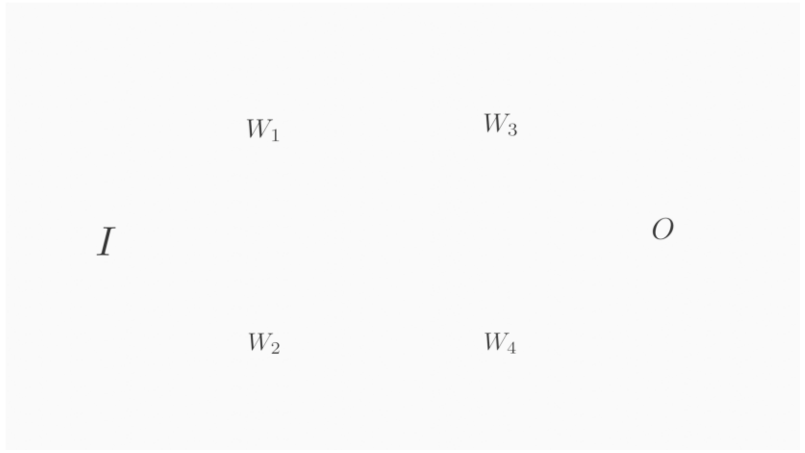
However, basic RNNs struggle with long-term dependencies due to:

- **Vanishing gradient problem**: The influence of early inputs fades over time
- **Exploding gradient problem**: Gradients grow uncontrollably during training

Business applications include:

- Language Modeling & Text Generation – Predicting the next word in a sequence (e.g., autocomplete, chatbots).
- Machine Translation – Translating text from one language to another.
- Speech Recognition – Converting spoken language into written text.
- Stock Price Prediction – Predicting future stock or financial data.
- Weather Forecasting – Modeling temporal patterns in weather data.
- Patient Monitoring – Analyzing sequences of medical data (e.g., ECG signals).
- Music Generation – Creating sequences of musical notes.
- Fraud Detection – Detecting unusual sequences in financial transactions.
- Network Intrusion Detection – Monitoring patterns of activity over time.





Transformers

Transformers – Architecture and Principles

What is a Transformer?

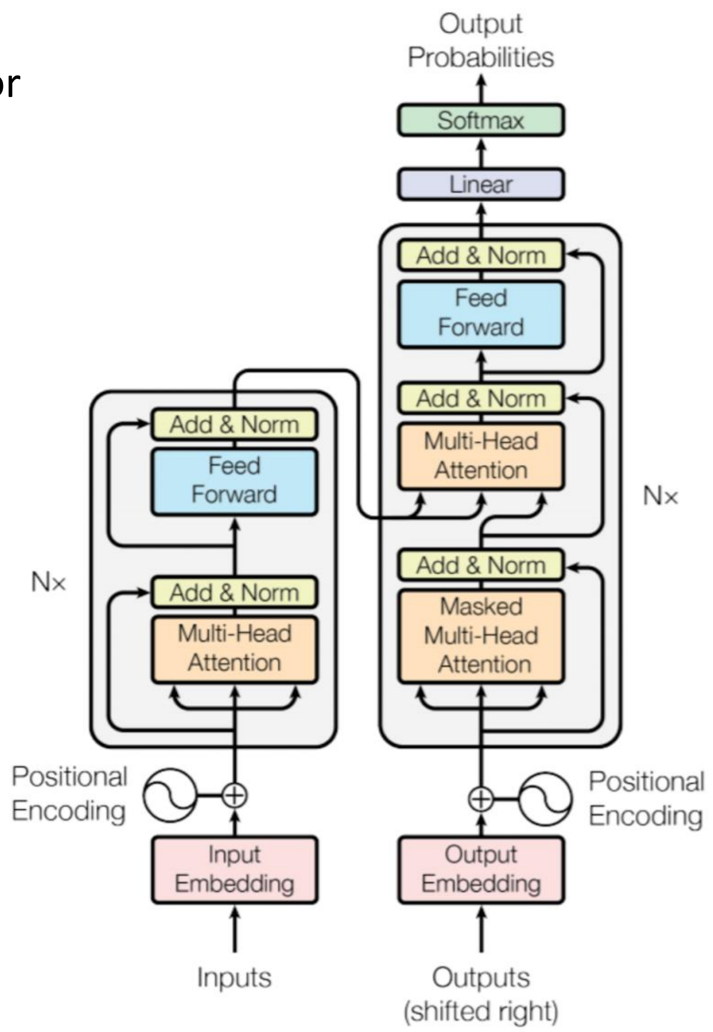
- A deep learning model based entirely on **self-attention**, with no recurrence or convolutions
- Introduced in the paper *"Attention Is All You Need"* (Vaswani et al., 2017)

Transformer Components:

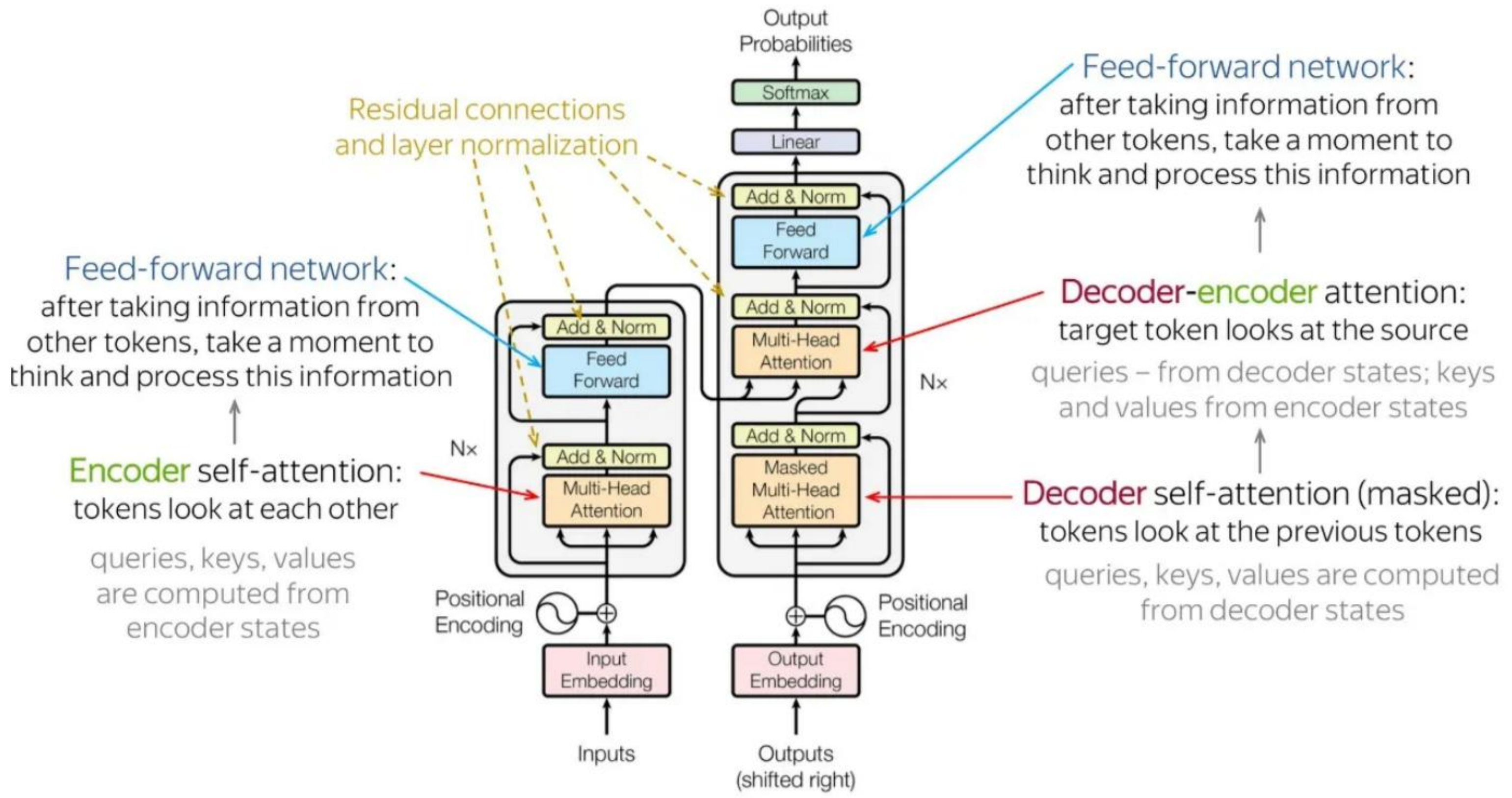
1. **Embeddings**: Convert tokens to vector representations
2. **Positional Encoding**: Adds position information
3. **Multi-Head Attention**: Processes relationships from multiple perspectives
4. **Feed-Forward Networks**: Process each position independently
5. **Layer Normalization**: Stabilizes training
6. **Residual Connections**: Helps with gradient flow

Architecture Variations:

- **Encoder-only** (BERT): Good for understanding (classification, NER)
- **Decoder-only** (GPT): Good for generation
- **Encoder-decoder** (T5): Good for transformation tasks (translation, summarization)



Transformers – Architecture and Principles


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Input Embedding
Inputs
Residual connections and layer normalization
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Encoder self-attention: tokens look at each other
queries, keys, values are computed from encoder states
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Decoder-encoder attention: target token looks at the source
queries – from decoder states; keys and values from encoder states
Decoder self-attention (masked): tokens look at the previous tokens
queries, keys, values are computed from decoder states

Input Sequence

A sequence of words or tokens that the model will process.

The Transformer in Detail

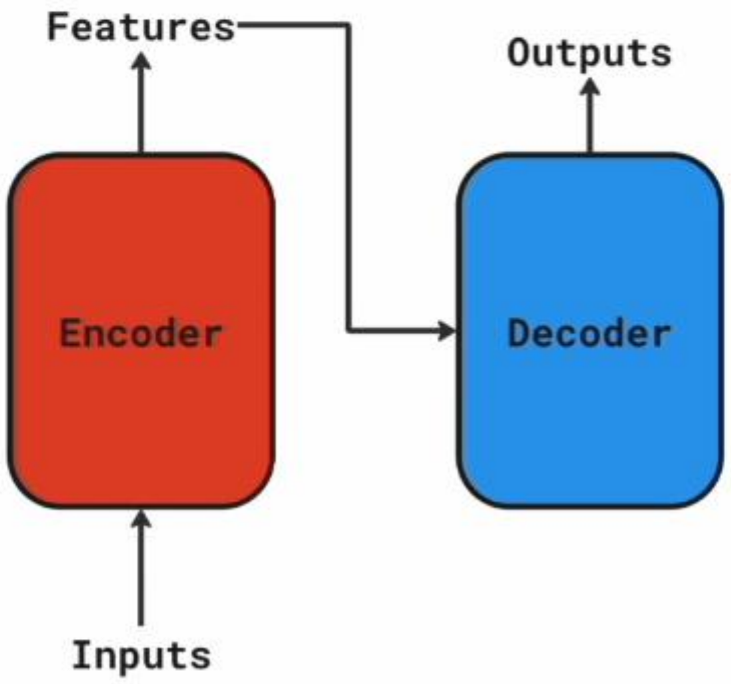
- The transformer architecture is designed to process sequential data, such as natural language, in a highly efficient and effective manner.
- Unlike traditional models that rely on sequential processing (like RNNs), transformers utilize a mechanism called self-attention, allowing them to analyze the entire input sequence at once.
- This capability enables them to capture complex relationships and dependencies between tokens in the sequence.

The transformer model consists of two main parts:

1. **Encoder:** The encoder processes the input sequence and generates a continuous representation of it. This representation captures the contextual information of the input tokens.
2. **Decoder:** The decoder takes the encoder's output and generates the final output sequence. It does this by predicting one token at a time, using the encoded representations and previously generated tokens.

- Both the encoder and decoder are composed of multiple identical layers—typically six layers in the original transformer architecture—allowing for deep learning and complex feature extraction.

Transformer Architecture



"Hello world" -> encoder -> knowledge ->
-> decoder -> *"Hola mundo"*

Key Components of Transformers

1. **Multi-Head Attention:**

Function: Multi-head attention allows the model to focus on different parts of the input sequence simultaneously.

It computes attention scores for each token in relation to all other tokens, enabling the model to weigh the importance of each token when making predictions.

Mechanism: The attention mechanism uses three vectors: Query (Q), Key (K), and Value (V).

The attention scores are calculated as the dot product of the query and key vectors, scaled by the square root of the dimension of the key vectors.

These scores are then used to weight the value vectors, producing a context-aware representation of the input.

2. Feed-Forward Networks:

Function: Each layer of the encoder and decoder contains a feed-forward neural network that processes the output from the attention mechanism.

This network enhances the model's ability to learn complex representations.

Structure: The feed-forward network consists of two linear transformations with a non-linear activation function (usually ReLU) applied between them.

This allows the model to capture intricate patterns in the data.

Key Components of Transformers

3. Positional Encoding:

Purpose: Since transformers do not inherently understand the order of tokens, positional encodings are added to the input embeddings.

This encoding provides information about the position of each token within the sequence, allowing the model to consider the order of words.

Implementation: Positional encodings are typically generated using sine and cosine functions, which create unique encodings for each position that can be added to the input embeddings.

4. Layer Normalization:

Function: Layer normalization stabilizes the training process by normalizing the inputs to each layer.

This helps mitigate issues related to internal covariate shift and improves convergence during training.

Application: It is applied after the attention and feed-forward layers, ensuring that the outputs are centered and scaled appropriately.

5. Residual Connections:

Purpose: Residual connections help facilitate the flow of gradients during training, addressing the vanishing gradient problem.

They allow the model to learn more effectively by providing a direct path for gradients to flow through the network.

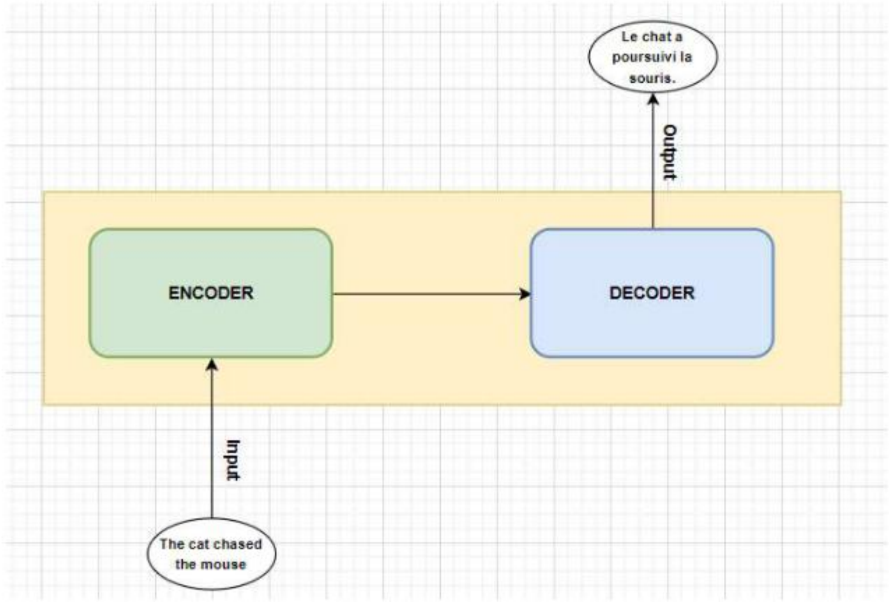
Implementation: The output of each sub-layer (attention and feed-forward) is added back to the original input, creating a shortcut that enhances learning.

The transformer architecture is a powerful and flexible model that has transformed the landscape of natural language processing and other fields. Its ability to process entire sequences simultaneously, leverage self-attention mechanisms, and utilize deep learning through stacked layers makes it a robust choice for a wide range of tasks.

How Transformer Works

Let's understand how the transformer takes input, processes and gives output.
We will consider a simple example of translating an English sentence to French using a transformer model.
Suppose we have,

- **Input sentence:** "Your cat is a lovely cat”
  We want to translate this to French:
- **Output sentence:** "Ton chat est un chat adorable.“
  The transformer takes the input, translates, and gives the output.

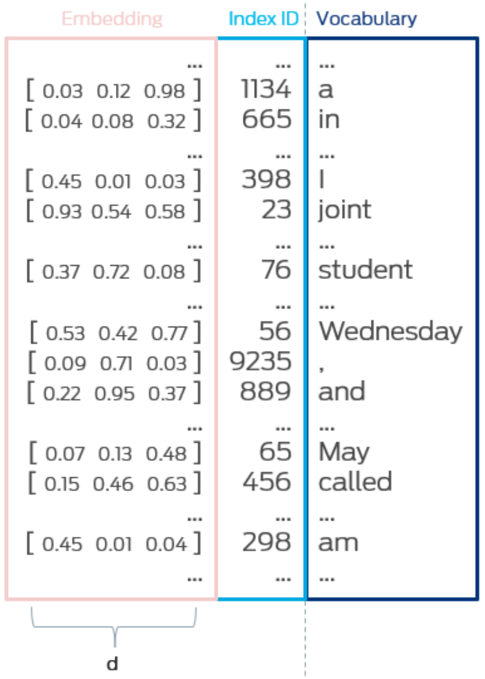


Input Preparation

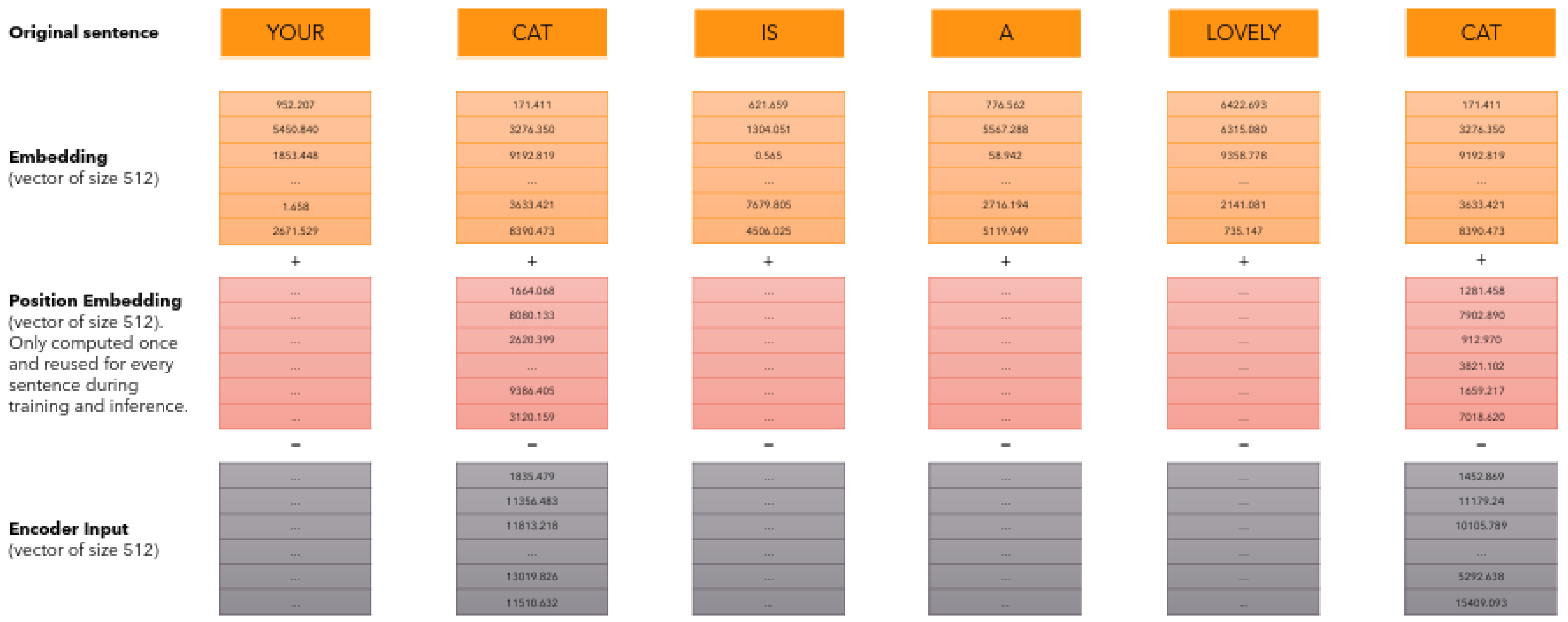
English sentence (source):

“Your cat is a lovely cat“

- Tokenization: Break the sentence into tokens: ["<s>", "Your", "cat", "is", "a", "lovely", "cat", "</s>"]
- Embedding: Convert each token into a vector using a learned embedding matrix. These embeddings capture the semantic meaning of each word.
- Positional Encoding: Add position-based information to each token embedding to provide information about the position of each token in the sequence. This is necessary because transformers process the entire sequence simultaneously, unlike RNNs that process one word at a time.
- Without positional encoding, the self-attention mechanism would treat the sentence as a bag of words.



Embedding	Index ID	Vocabulary
...	...	...
[0.03 0.12 0.98]	1134	a
[0.04 0.08 0.32]	665	in
...	...	...
[0.45 0.01 0.03]	398	I
[0.93 0.54 0.58]	23	joint
...	...	...
[0.37 0.72 0.08]	76	student
...	...	...
[0.53 0.42 0.77]	56	Wednesday
[0.09 0.71 0.03]	9235	,
[0.22 0.95 0.37]	889	and
...	...	...
[0.07 0.13 0.48]	65	May
[0.15 0.46 0.63]	456	called
...	...	...
[0.45 0.01 0.04]	298	am
...	...	...



Original sentence	YOUR	CAT	IS	A	LOVELY	CAT
Embedding (vector of size 512)	952.207	171.411	621.659	776.562	6422.693	171.411
	5450.840	3276.350	1304.051	5567.288	6315.080	3276.350
	1853.448	9192.819	0.565	58.942	9358.778	9192.819
	...	...	...	...	...	...
	1.658	3633.421	7679.805	2716.194	2141.081	3633.421
	2671.529	8390.473	4506.025	5119.949	735.147	8390.473
	+	+	+	+	+	+
Position Embedding (vector of size 512). Only computed once and reused for every sentence during training and inference.	...	1664.068	...	...	...	1281.458
	...	8080.133	...	...	...	7902.890
	...	2620.399	...	...	...	912.970
	...	...	...	...	...	3821.102
	...	9386.405	...	...	...	1659.217
	...	3120.159	...	...	...	7018.620
	=	=	=	=	=	=
Encoder Input (vector of size 512)	...	1835.479	...	...	...	1452.869
	...	11356.483	...	...	...	11179.24
	...	11813.218	...	...	...	10105.789
	...	...	...	...	...	...
	...	13019.826	...	...	...	5292.638
	...	11510.632	...	...	...	15409.093

Encoder

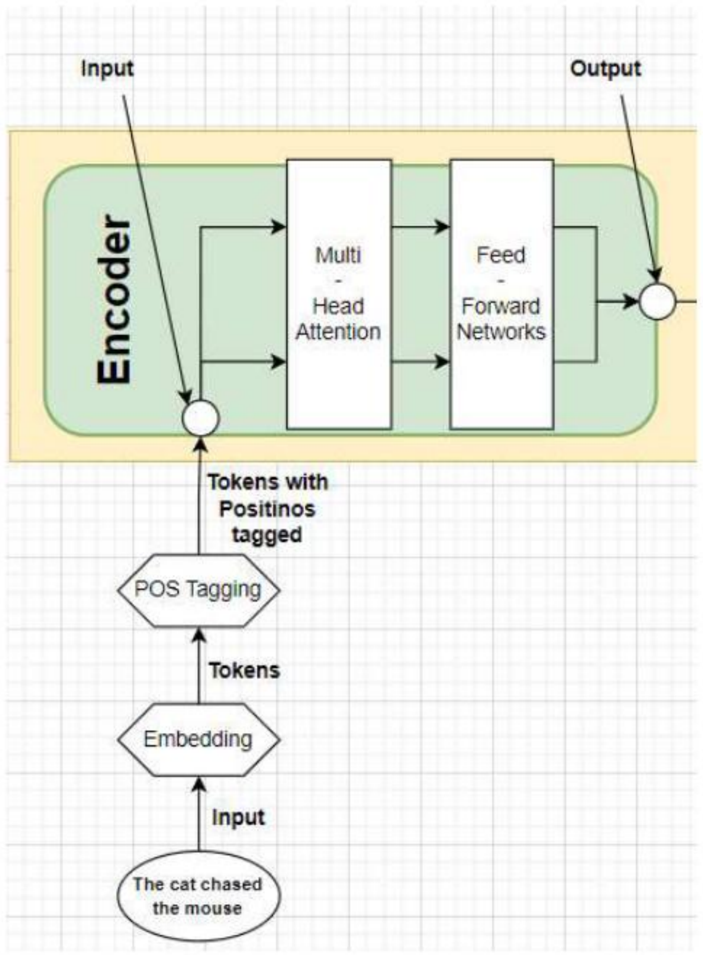
The encoder processes the full input sequence in parallel through a stack of layers.

The encoder takes the input embeddings with positional encodings and passes them through multiple layers of multi-head attention and feed-forward networks. Each encoder layer processes the input, allowing the model to learn complex representations of the sentence.

For example, in a sentence like "The cat chased the mouse", the attention mechanism in the encoder might learn that "cat" is related to "chased" and "mouse", capturing the semantic relationships between the tokens.

Each encoder layer includes:

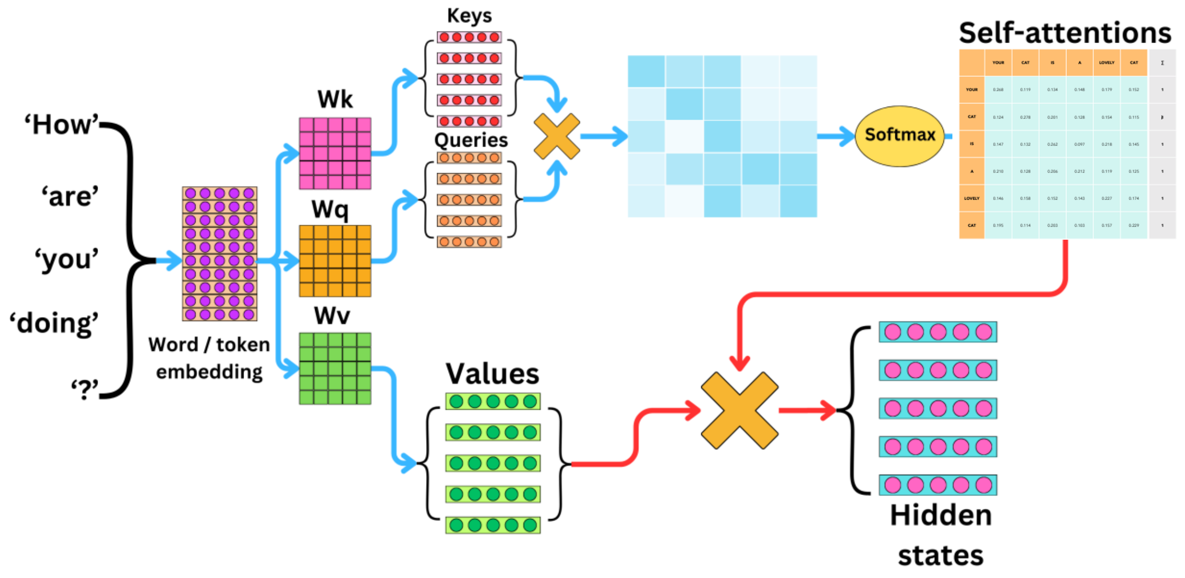
- **Multi-head Self-Attention:**
  Each word learns which other words to focus on.
  Example: The second occurrence of "cat" may attend to the first "cat" to recognize repetition or coreference.
- **Add & Norm:**
  A residual connection followed by layer normalization.
- **Feedforward Network:**
  A two-layer neural network processes each position independently.
- **Add & Norm:**
  Another residual connection and normalization.
- This stack is repeated several times (e.g., 6 layers), producing a set of **contextualized vectors**, one for each token.



The Attention Mechanism

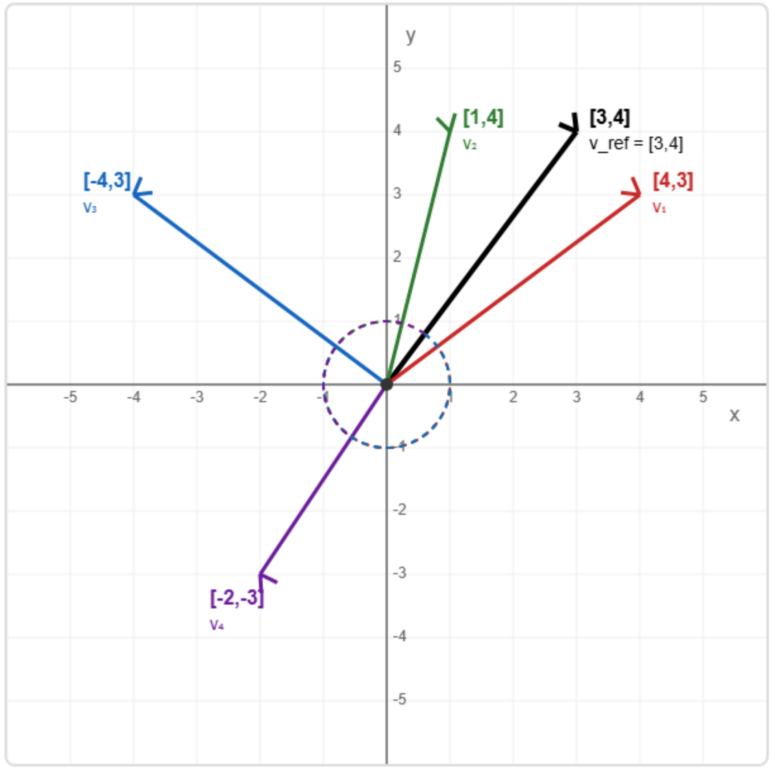
1. **Create Q, K, V matrices**: Each word embedding is multiplied by three learned weight matrices (WQ, WK, WV) to create Query, Key, and Value representations:
   - Q = X · WQ
   - K = X · WK
   - V = X · WV
2. **Compute attention scores**: Each query vector is dot-producted with all key vectors to get raw attention scores:
   - Scores = Q · K^T
3. **Scale the scores**: Divide by √dk (where dk is the dimension of the key vectors) to prevent the values from getting too large:
   - Scaled scores = (Q · K^T) / √dk
4. **Apply softmax**: Convert the scaled scores to probabilities:
   - Attention weights = softmax(Scaled scores)
5. **Compute weighted values**: Multiply the attention weights by the value vectors:
   - Output = Attention weights · V

So the formula is: Attention(Q,K,V) = softmax((Q·K^T)/√dk) · V



Vector Dot Product as Similarity Measure

$$v_1 \cdot v_2 = x_1x_2 + y_1y_2$$



1. Very Similar (High Positive)

`Red Vector v₁ = [4, 3]`

$v_1 \cdot$ v_ref = (4×3) + (3×4) = 24

Almost same direction as reference

2. Similar (Positive)

`Green Vector v₂ = [1, 4]`

$v_2 \cdot$ v_ref = (1×3) + (4×4) = 19

Generally same direction

3. Neutral (Zero)

`Blue Vector v₃ = [-4, 3]`

$v_3 \cdot$ v_ref = (-4×3) + (3×4) = 0

Perpendicular (90° angle)

4. Dissimilar (Negative)

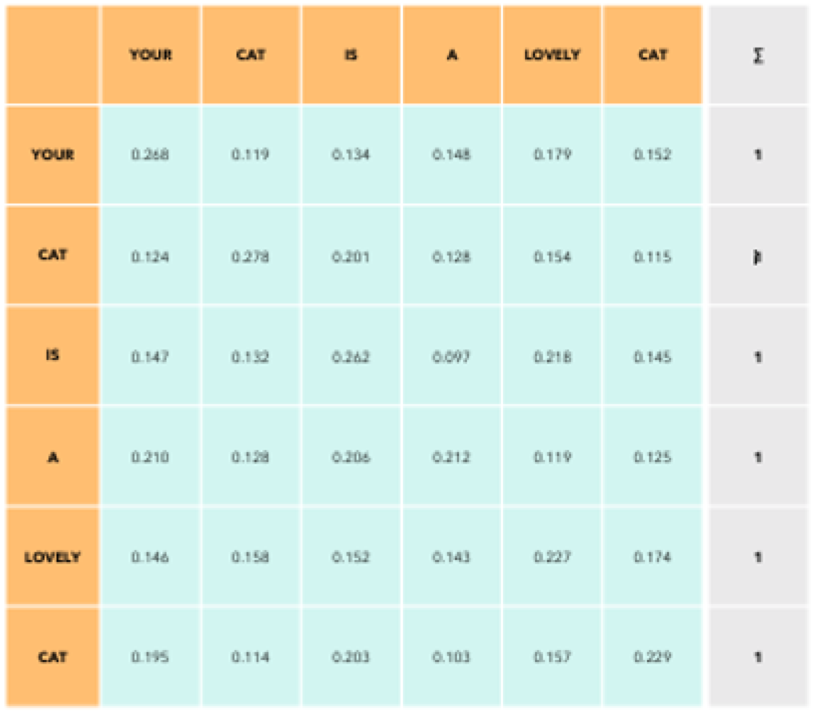
`Purple Vector v₄ = [-2, -3]`

$v_4 \cdot$ v_ref = (-2×3) + (-3×4) = -18

Opposite direction

Key Innovation: Self-Attention Mechanism

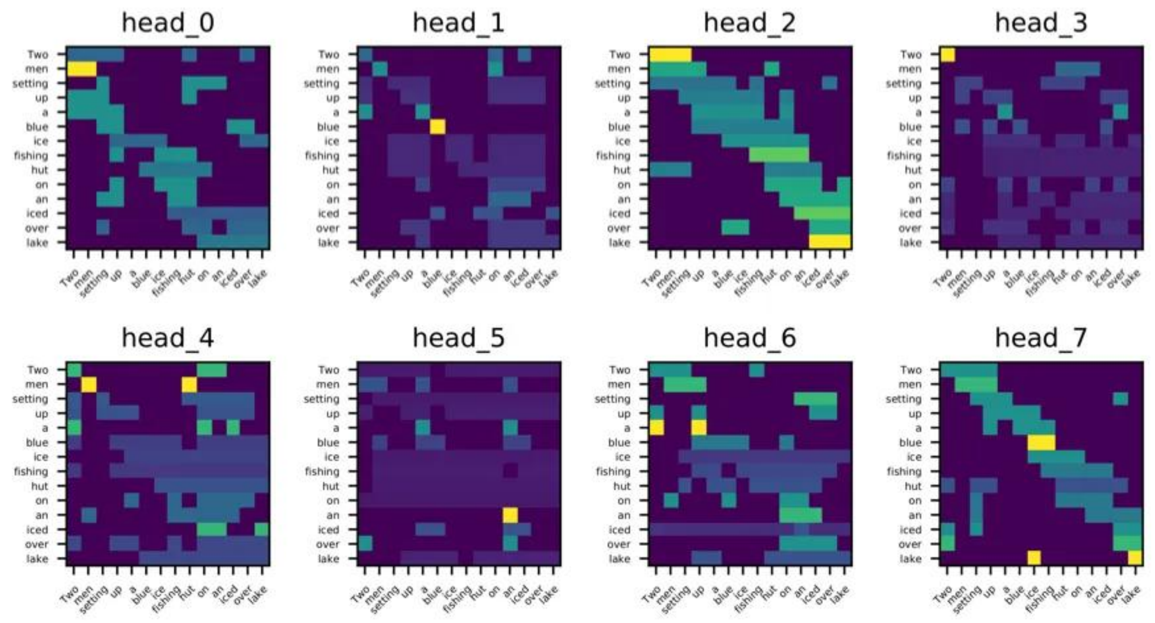
- Self-attention allows each word to compute its embedding by gathering information from all other words in the sequence.
- The "attention weights" determine how much each word should focus on other words.
- Words that are semantically related tend to have higher attention scores between them.
- This mechanism helps capture long-range dependencies and relationships regardless of word distance.
- Multiple attention heads in parallel (Multi-head Attention) allow the model to focus on different aspects of relationships.



	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

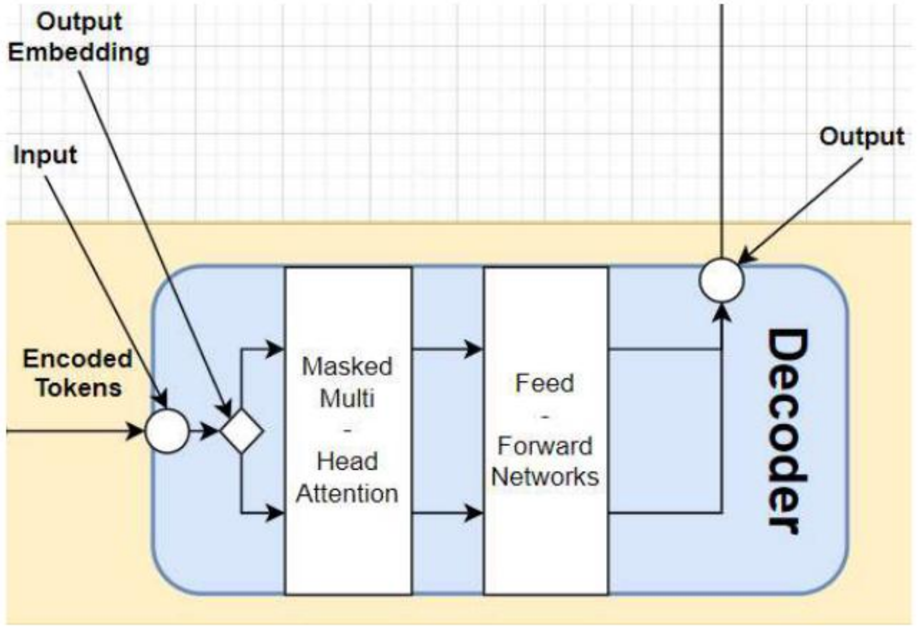
Multiple Attention Heads

- Multiple attention heads in parallel (Multi-head Attention) allow the model to focus on different aspects of relationships.
- For example, one head might learn which words are related by grammar, while another might focus on semantic meaning.
- This allows the model to capture a richer and more comprehensive understanding of the input.



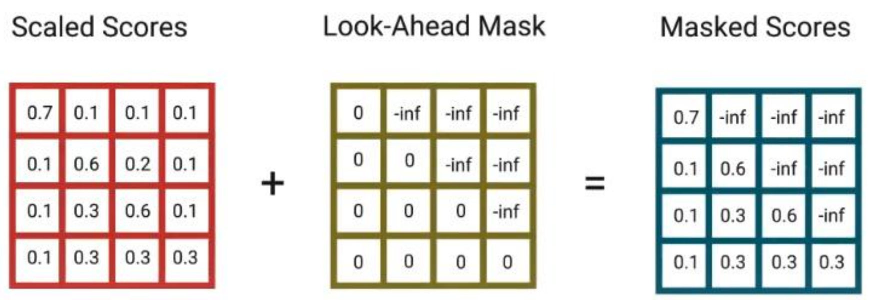
Decoding Process

- The decoder takes the encoder's output and generates the output sequence in French.
- It uses masked multi-head attention to ensure that predictions for a given token do not depend on future tokens.
- This allows the decoder to generate the output one token at a time.
- The decoder also attends to the encoder's output, enabling it to incorporate context from the input sentence while generating the translation.
- For instance, the attention mechanism in the decoder might focus on the representation of "cat" when generating "Le chat", ensuring that the translation is consistent with the input.



Decoder Components

1. **Masked Multi-head Self-Attention:**
   Looks at the previously generated French words. Future words are masked to prevent cheating.

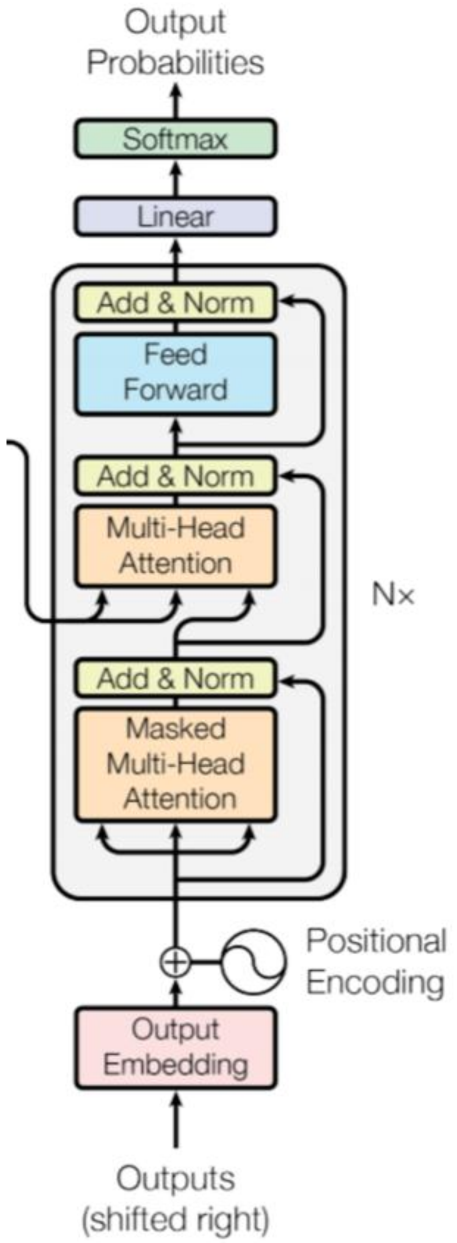


$$\begin{bmatrix} 0.7 & 0.1 & 0.1 & 0.1 \\ 0.1 & 0.6 & 0.2 & 0.1 \\ 0.1 & 0.3 & 0.6 & 0.1 \\ 0.1 & 0.3 & 0.3 & 0.3 \end{bmatrix} + \begin{bmatrix} 0 & -inf & -inf & -inf \\ 0 & 0 & -inf & -inf \\ 0 & 0 & 0 & -inf \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0.7 & -inf & -inf & -inf \\ 0.1 & 0.6 & -inf & -inf \\ 0.1 & 0.3 & 0.6 & -inf \\ 0.1 & 0.3 & 0.3 & 0.3 \end{bmatrix}$$

2. Encoder-Decoder Attention:
Each decoder token can attend to all encoder outputs.
Example: The decoder token "chat" may attend to the English "cat" to align the translation.

3. Feedforward Network:
Processes each token vector separately.

Also, each sub-layer includes residual connections and layer normalization.



Decoder Training and Inference

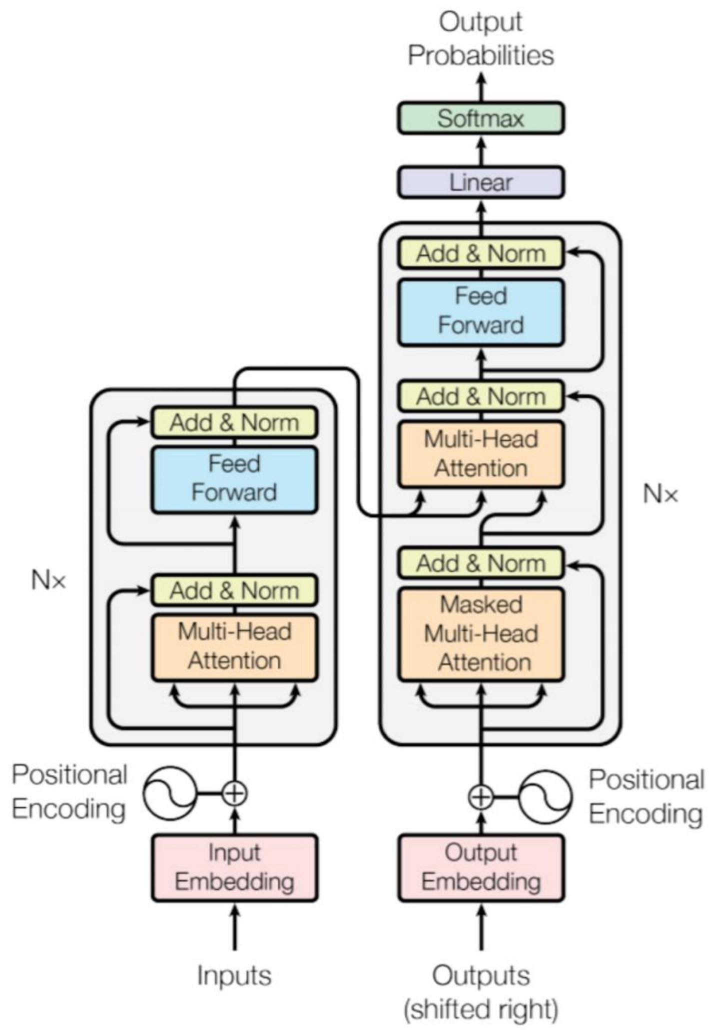
Training Phase:

- The decoder begins with a special start-of-sequence token <s>
- At each time step, it receives the actual previous target words.
- Example: Step 1: <s>, Step 2: <s> Ton, Step 3: <s> Ton chat, etc.

Inference Phase:

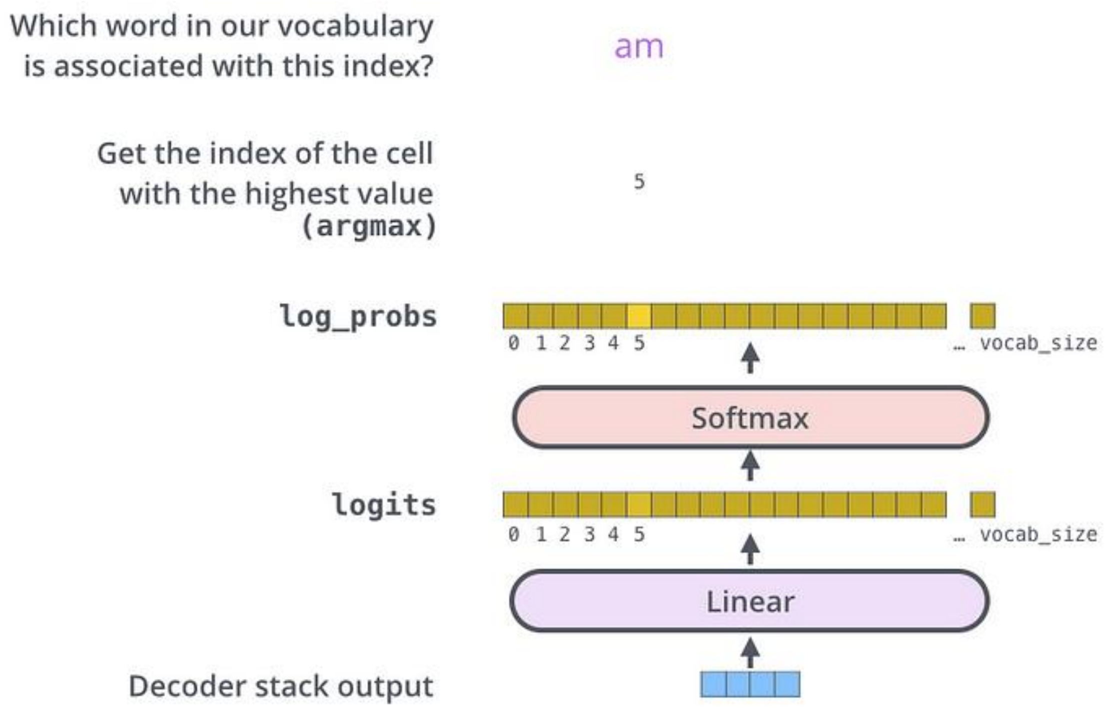
- Starts with <s> and generates one word at a time.
- Each new word is used as input for the next step.
- Example: <s> → Ton → chat → est → ...

Summary:
By combining attention to past outputs and the encoded input, the decoder generates coherent, context-aware text—essential for tasks like translation and summarization.



Output Generation

- The decoder outputs a vector at each time step.
- A linear layer followed by softmax turns this vector into a probability distribution over the French vocabulary.
- The model selects the most probable next word ("Ton", then "chat", then "est", etc.).
- This continues until an end-of-sentence token </s> is generated or a length limit is reached.


Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5
… vocab_size
Softmax
logits
0 1 2 3 4 5
… vocab_size
Linear
Decoder stack output

A Summary of how the Transformer Works

Input Sentence: "The cat chased the mouse."

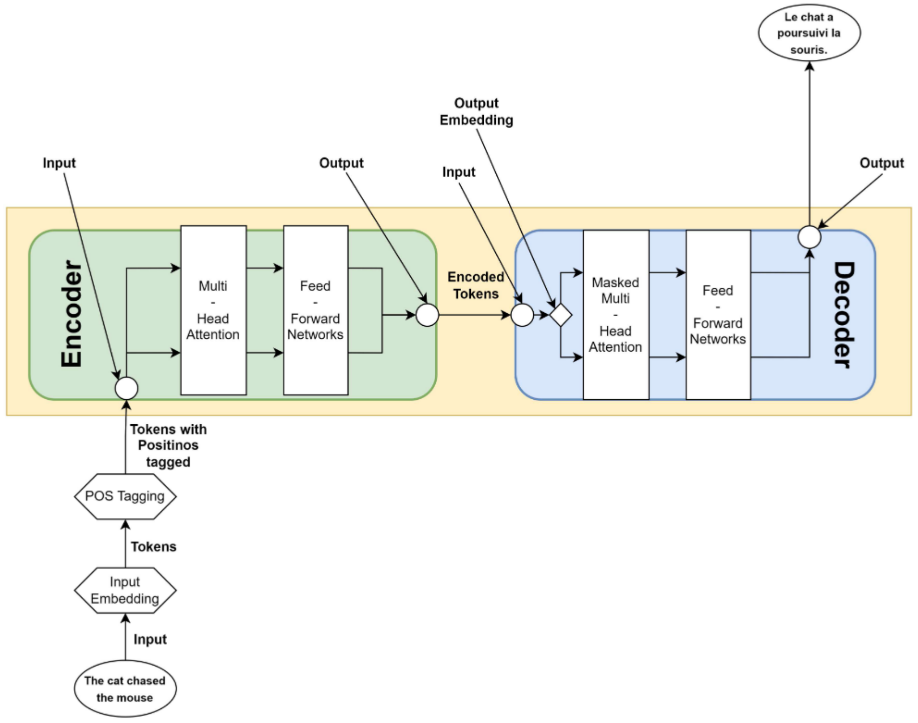
Input Encoding:
- Break down the sentence into tokens (words).
- Convert each token into a numerical representation called an embedding.
- Add positional encodings to the embeddings to provide information about the position of each token.

Encoder Processing:
- The encoder takes the input embeddings with positional encodings.
- Pass the input through multiple layers of multi-head attention and feed-forward networks.
- Each encoder layer processes the input, allowing the model to learn complex representations of the sentence.
- The attention mechanism in the encoder learns relationships between tokens (e.g., "cat" is related to "chased" and "mouse").

Decoder Processing:
- The decoder takes the encoder’s output.
- Use masked multi-head attention to generate the output one token at a time.
- Attend to the encoder’s output to incorporate context from the input sentence while generating the translation.
- The attention mechanism in the decoder focuses on relevant parts of the input (e.g., the representation of "cat" when generating "Le chat").

Output Generation:
- The decoder generates the output sequence token by token.
- For the example, it generates: “Le”, “chat”, “a”, “poursuivi”, “la”, and “souris”.
- The complete French translation is: “Le chat a poursuivi la souris.”

Key Advantages:
- Process the entire input sequence simultaneously.
- Use attention mechanisms to capture relationships between tokens.
- Efficiently translates sentences, even with long-range dependencies.



Number of Parameters - Original Transformer Model

Component	Parameter	Formula / Size	Total Parameters
Input	Token embedding	Vocab_Size × d_model = 37000 × 512	≈ 18.94M
	Positional encoding (fixed)	n × d_model	Not learned (original paper used fixed)
Attention (per layer)	Q/K/V weights per head	3 × d_model × d_k = 3 × 512 × 64	98,304
	Output projection	d_model × d_model = 512 × 512	262,144
	Total per Multi-Head block	–	≈ 360K
Feed-Forward (per layer)	Linear 1: 512 × 2048	–	1,048,576
	Linear 2: 2048 × 512	–	1,048,576
	Total FFN per layer	–	≈ 2.10M
LayerNorm	2 × γ, β per layer	2 × d_model = 2 × 512	1,024
Encoder Block Total	–	Attention + FFN + LayerNorm	≈ 2.46M
Encoder Total (6 layers)	–	6 × 2.46M	≈ 14.76M
Decoder Total (6 layers)	Similar structure + cross attention	≈ 2.6M per layer	≈ 15.6M
Output Layer	d_model × Vocab_Size = 512 × 37000	tied/shared with embedding	≈ 18.94M
Total Model Parameters	–	Encoder + Decoder + Embedding + Output	**≈ 65M**

References

- https://jalammar.github.io/illustrated-transformer/
- https://tamoghnasaha-22.medium.com/transformers-illustrated-5c9205a6c70f