Introduction

Sunday, May 4, 2025 8:04 PM

We will learn about large language models and natural language processing using libraries from Hugging Face ecosystem

- Transformers
- Datasets
- Tokenizers
- Accelerate
- Hugging Face Hub

Natural Language Processing:

Focused on enabling computes to

- Understand
- Interpret
- Generate human language

Examples:

- Sentiment analysis
- Named entity recognition
- Machine translation

Large Language Models:

Powerful subset of NLP models known for their

- Massive size
- Huge training data
- Ability to wide range of tasks with minimal task specific training

Examples:

- Llama
- GPT
- DeepSeek

GPT-4:

- Training Size: 570 GB
- No. of Parameters: 1.7 trillion
- Total size is 6800 GB

Source: LinkedIn

Natural Language Processing and Large Language Models

Sunday, May 4, 2025 8:25 PM

What is NLP?

A field of linguistics & machine learning ---> understand everything about human language.
-> Not only to understand words, but their context also.

Example:

- Sentiment Analysis
- Named entity recognition
- Fillings blanks in the text
- Translating text into another language

The Rise of Large Language Models

LLMs are known for

- **Scale:** trillion of parameters
- **General capabilities:** perform several tasks without getting trained to perform that task.
- **In-context learning:** learn from examples given in the prompt. (Zero-shot and few shot learning)
- **Emergent capabilities:** grow in size --> perform tasks that weren't explicitly programmed.

Zero-shot learning: provide no relevant examples.
Few shot learning: provide relevant examples.

Build tasks specific models : NO
User a single model that can fine-tuned for another tasks: YES

Change in NLP models after LLMs

Limitations

- **Hallucinations:** generate incorrect information confidently.
- **Lack of true understanding:** lack of true understanding of world operate purely on statistical patterns.
- **Bias:** reproduce bias present in their training data. (garbage In = garbage out)
- **Context windows:** limited
- **Computational resources:** demand significant computational resources.

Why is language processing challenging?

- Computers don't process information as human do.
- Providing two sentences "I love you" and "I hate you", we can understand how similar or different they are.
- Need complex method to convert the text in a form that machine understands.

Note:
LLMs still face difficulty while tackling following

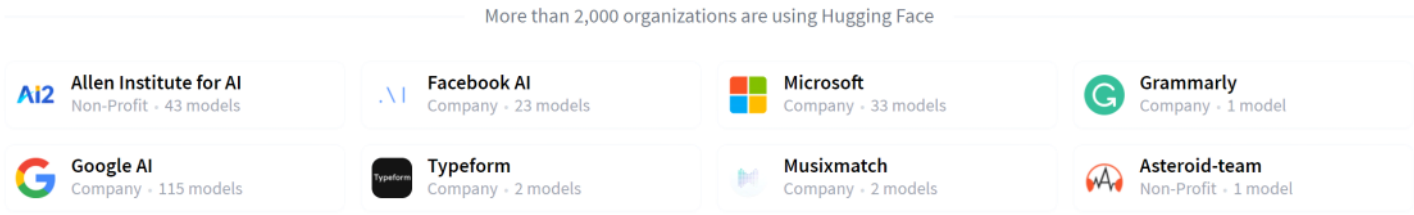
- Understanding ambiguity
- Cultural context
- Sarcasm
- Humor

Although it has been improved due to massive training data, it still lack human-level understanding in many complex scenarios.

Transformers, what can they do?

Sunday, May 4, 2025 8:46 PM

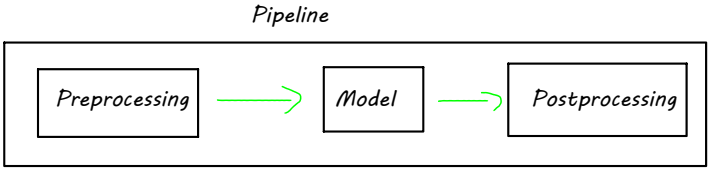
They are everywhere. They are used in language processing, image processing, audio processing and video processing.



What happens when we pass some text to **pipeline**?

- Text is preprocess into a format that model can understand.
- Preprocessed inputs are passed into models.
- Predictions of the models are post-processed, so humans can understand.

Available Pipelines: https://huggingface.co/models



How do Transformers work?

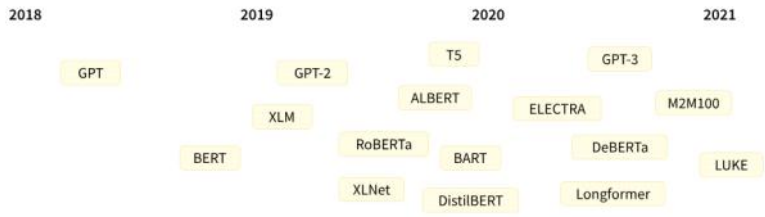
Sunday, May 4, 2025 9:30 PM

We are going to study

- Architecture of transformer models
- Attention mechanism
- Encoder-decoder architecture

Transformer History

Introduced in June 2017
Original research was focused on translation task



- **June 2018**: GPT, first pre-trained transformer model.
- **October 2018**: BERT, another pre-trained model designed to produce better summaries of sentences.
- **February 2019**: GPT-2, an improved version of GPT.
- **October 2019**: T5, a multi-task implementation of the sequence-to-sequence transformer architecture.
- **May 2020**: GPT-3, able to perform several tasks without the need for fine-tuning.
- **January 2022**: InstructGPT, a version of GPT-3, trained to following instructions better.
- **January 2023**: Llama, able to generate text in several languages.
- **March 2023**: Mistral, a 7-billion parameter model.
- **May 2024**: Gemma 2, a family of lightweight, state of the art open models (2B- 27B).
- **November 2024**: SmolLM2, impressive performance despite its compact size + unlock new possibilities for mobile & edge devices.

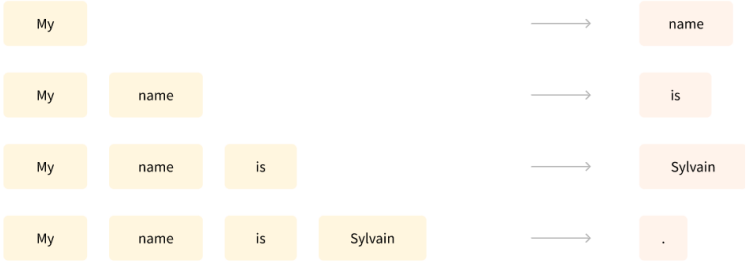
GPT-like: auto-regressive transformer models
BERT-like: auto-encoding transformer models
T5-like: sequence-to-sequence transformer models

Transformers are language models

- Pre-training is done through self-supervised learning ---> no need to label the data.
- Fine-tuning is done through supervised learning ---> Need to label the data before feeding to the model.

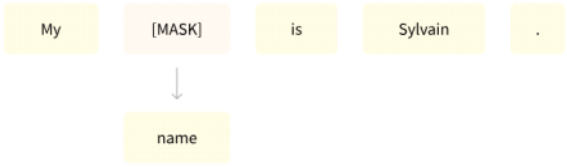
Causal language modeling

Predicting in the next word in a sentence having read the n previous words.



Masked Language Modeling

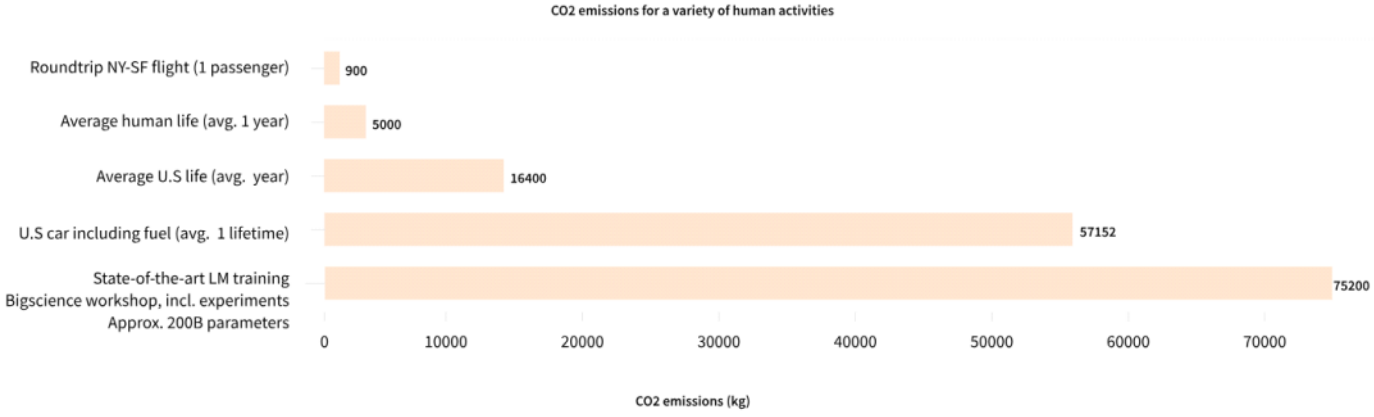
Model predicts a masked in the sentence.



Transformers are big models

A general strategy to achieve performance is by increasing the size of model and the amount of training data. (Though some exceptions do exist like DistilBERT)

The negative side of this rule is that training a model requires a large amount of data. This directly affects the cost and the computer resources. It even translates to environmental impact.



To further study: [The carbon footprint of transformers]

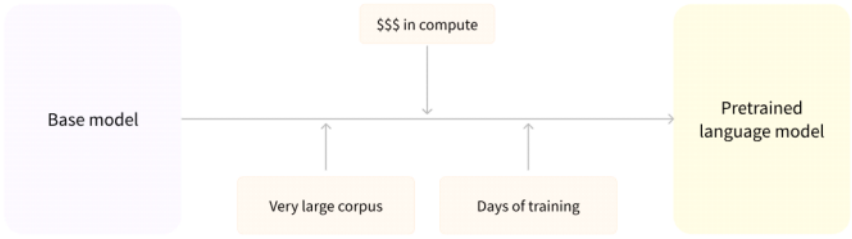
That's why sharing large language models are paramount (of utmost importance). Sharing the trained weights and building on top of already trained weights reduces the overall compute cost and carbon footprint of the community.

Transfer Learning:

Before we discuss what transfer learning is, let's see what pre-training is.

Pre-training: Act of training a model from scratch.

- The weights are randomly initialized.
- There is no prior knowledge.

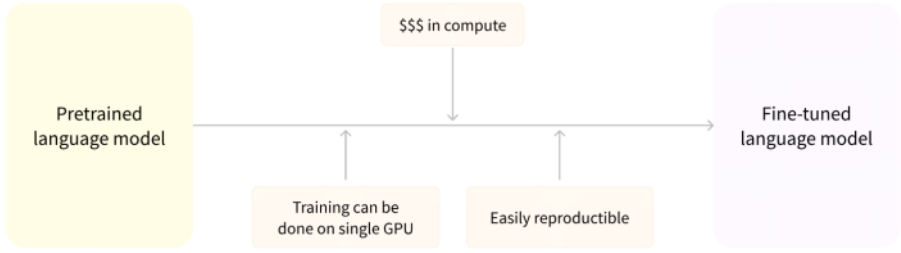


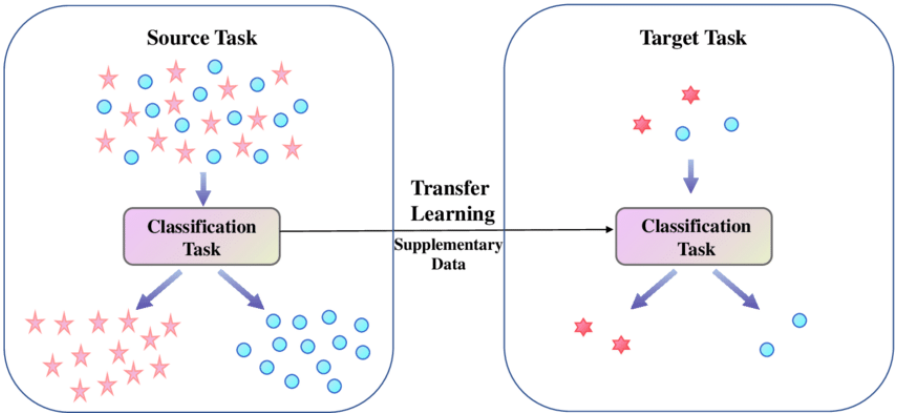
Transfer learning / Fine-tuning:

Training done after a model has been pre-trained.

Why NOT Pre-train?

- The data that is going to be used for fine-tuning has some similarity with the dataset that was used during pre-training. It makes the fine-tuning able to take advantage of the knowledge acquired during initial training.
- Requires very less amount of data.
- Efficient in the use of resources like computer power and time.



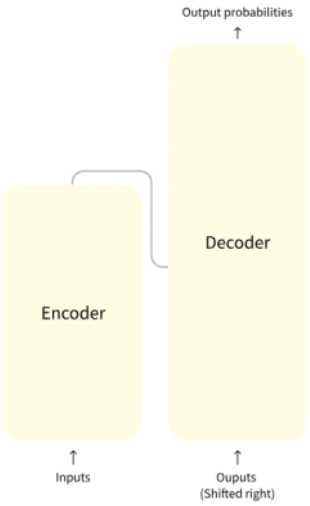


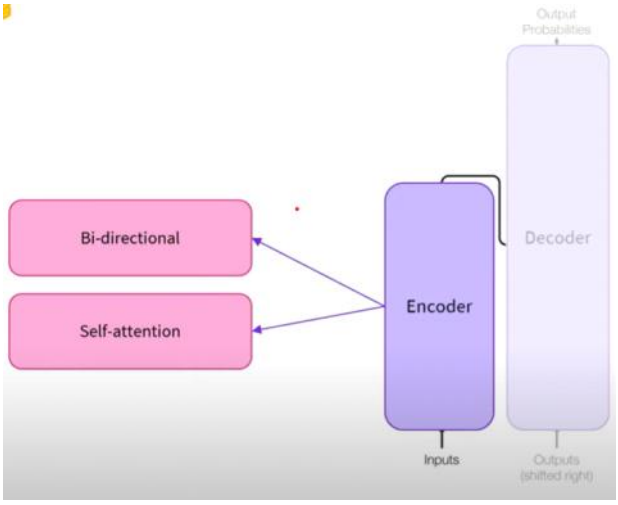
Source of Image: [Deep Transfer Learning for Identifications of Slope Surface Cracks]

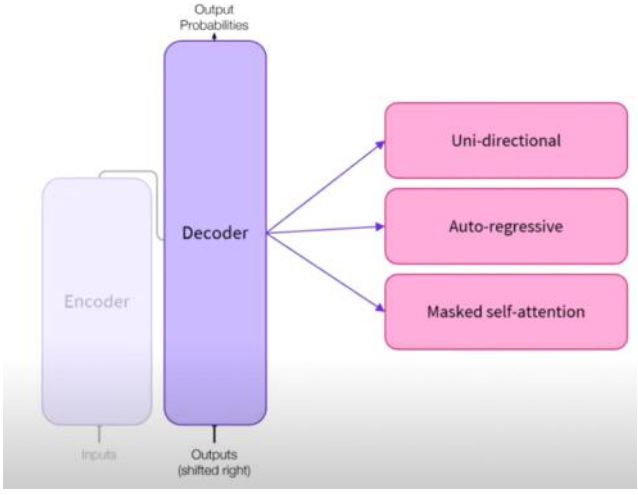
General Transformer Architecture

Primarily composed of two blocks

- Encoder: Receives an input and builds a representation of it (features). Model is built to acquire understanding from its inputs.
- Decoder: Uses encoder's representation (features) and other inputs to generate a target sequence.







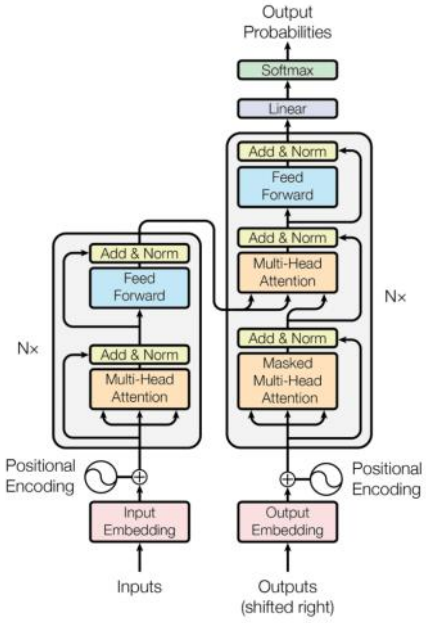
Each of these parts can be used independently, depending on the task:

- ***Encoder-only models**: Good for tasks that require understanding of the input, such as sentence classification and named entity recognition.*
- ***Decoder-only models**: Good for generative tasks such as text generation.*
- ***Encoder-decoder models or sequence-to-sequence models**: Good for generative tasks that require an input, such as translation or summarization.*

Attention Layers:

- Provide attention to certain tokens in the input.

The Original Architecture:



A bit scary. Isn't it?

Architecture vs Checkpoints:

Architecture: the structure/skeleton of model - the definition of each layer and each operation
Checkpoint: the weights that will be loaded into a specific architecture.

Solving Tasks with Transformers

Tuesday, May 6, 2025 12:28 PM

We have seen what transformers can do. Now is the right time to observe how transformers actually solve a task?
We will cover following models

- Wav2Vec2
- Mask2Former
- BERT
- GPT2
- BART

How language models work?
They work by being trained to predict the probability of a word given the context of surrounding words.

Two main approaches for training a transformer model

- Masked Language modeling (MLM)
- Causal Language Modeling (CLM)

Masked Language Modeling

- Used by encoder models like BERT.
- Randomly masks some tokens in the input and trains the model to predict the original tokens based on the surrounding context.
- Bidirectional context

Example:

he	goes/went	to	hospital	
he	went	to	hospital	yesterday
he	Will/shall	go to	hospital	Tomorrow

Causal Language Modeling

- Used by decoder models (GPT).
- Predicts the next tokens based on all previous tokens in the sequence
- Can only access the context from the left (previous tokens)

Types of language models

- Encoder
- Decoder
- Encoder-decoder

Encoder:

- Bidirectional approach
- Best for tasks that require rich understanding of text.
- Example: text classification, named entity recognition, question answering

Decoder:

- Process text from left to right.
- Good at text generation

Encoder-Decoder:

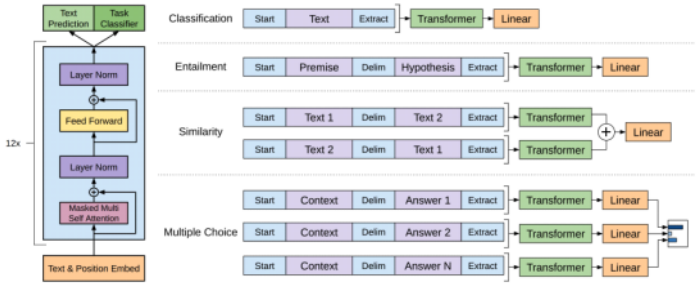
- Combine both approaches, encoder and decoder
- Good for tasks like translation, summarization and question answering

Text generation

- Involves creating coherent and contextually relevant text based on a prompt or input

GPT-2

- Decoder only model
- Can produce text



- Uses Byte Pair Encoding

Rest of the topic will be discussed after some practical coding examples of transformers.

Transformer Architecture

Tuesday, May 6, 2025 12:25 PM

Three main architectures
- Encoder only
- Decoder only
- Encoder-decoder

Encoder models

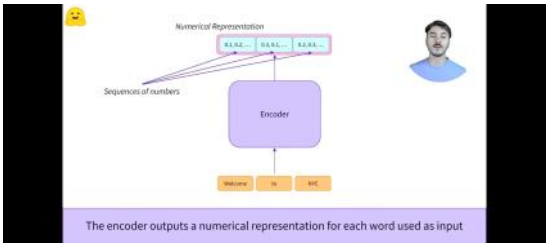
- Use only encoder of a transformer model.
- At each stage, the attention layer can access all the words in the initial sentence.
- Have "bi-directional" attention: context from left and right.
- Also known as auto-encoding models.

Best for tasks that require full understanding of a sentence such as
- Sentence classification
- Named entity recognition
- Extractive question answering
- Masked language modeling

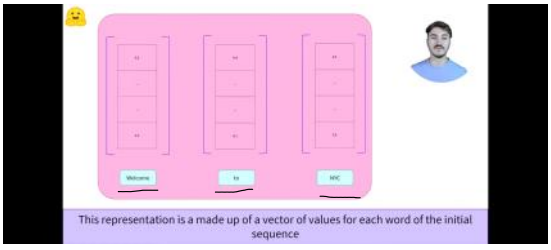
Pre-training of Encoder:
- Corrupt a given sentence (masking words) and then let the model correct it.

Examples:
- BERT - Bidirectional Encoder Representation from Transformers
- DistilBERT
- ModernBERT

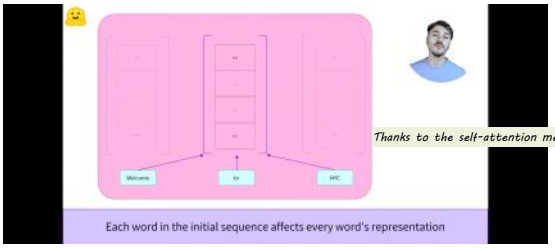
1



2

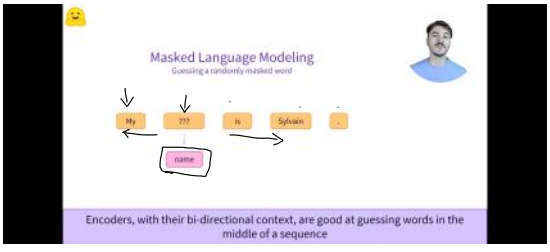


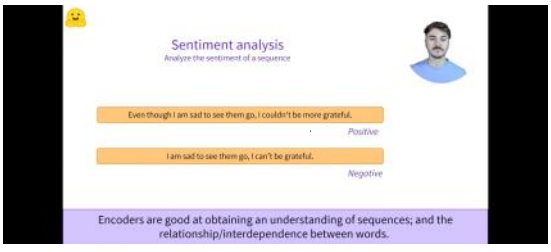
3



Thanks to the self-attention mechanism

Examples





Decoder models

- Uses only decoder part of transformer.
- At each stage, the attention layer, for a given word, can only access the words positioned before it.
- Also known as auto-regressive models.

Decoder models

- Uses only decoder part of transformer.
- At each stage, the attention layer, for a given word, can only access the words positioned before it.
- Also known as auto-regressive models.

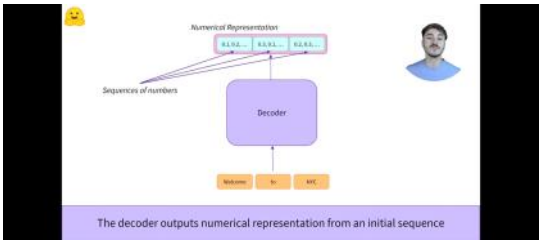
Pre-training:

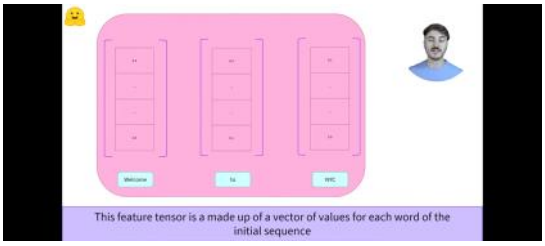
- Predict the next word.

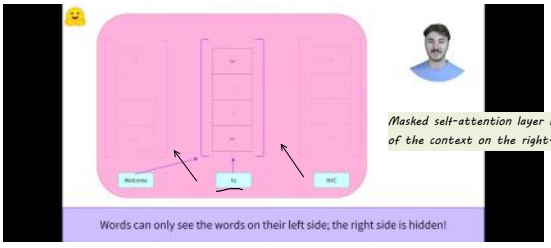
Best for tasks involving text-generation.

Examples:

- Hugging Face SmolLM Series
- Meta's Llama Series
- Google's Gemma Series
- DeepSeek's V3







Masked self-attention layer hides the value of the context on the right.

Modern Large Language Models

- Only decoder (exceptions do exist)
- Trained in two phases
  - Pretraining
  - Fine-tuning

Key capabilities of modern LLMs

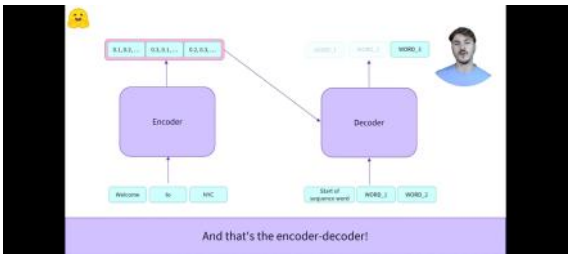
Capability	Description	Example
Text generation	Creating coherent and contextually relevant text	Writing essays, stories, or emails
Summarization	Condensing long documents into shorter versions	Creating executive summaries of reports
Translation	Converting text between languages	Translating English to Spanish
Question answering	Providing answers to factual questions	"What is the capital of France?"
Code generation	Writing or completing code snippets	Creating a function based on a description
Reasoning	Working through problems step by step	Solving math problems or logical puzzles
Few-shot learning	Learning from a few examples in the prompt	Classifying text after seeing just 2-3 examples

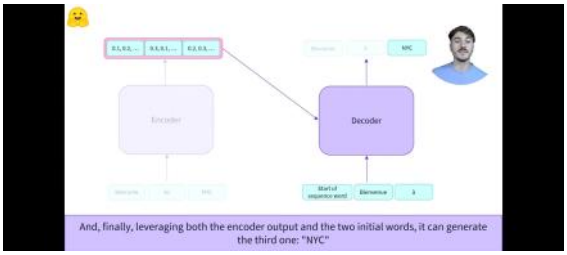
Sequence-to-Sequence Models

- Use both parts of transformer architecture. Encoder + decoder
- Good for tasks to generate new sentences based on input like
  - Summarization
  - Translation
  - Generative question answering

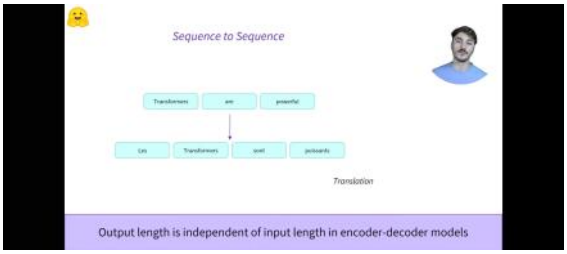
Examples:

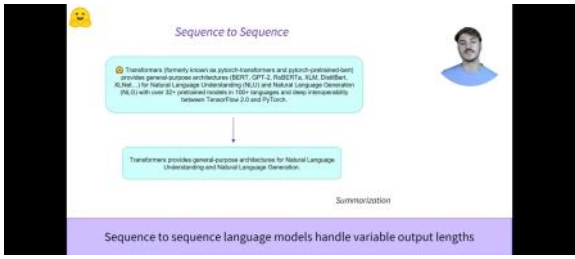
- BART
- mBART
- Marian
- T5





Examples:





Choosing the right architecture

Task	Suggested Architecture	Examples
Text classification (sentiment, topic)	Encoder	BERT, RoBERTa
Text generation (creative writing)	Decoder	GPT, LLaMA
Translation	Encoder-Decoder	T5, BART
Summarization	Encoder-Decoder	BART, T5
Named entity recognition	Encoder	BERT, RoBERTa
Question answering (extractive)	Encoder	BERT, RoBERTa
Question answering (generative)	Encoder-Decoder or Decoder	T5, GPT
Conversational AI	Decoder	GPT, LLaMA

Details about LSH attention, Local attention and axial positional encoding is available at https://huggingface.co/learn/llm-course/chapter1/6?fw=pt

Inference with LLMs

Wednesday, May 7, 2025 9:27 AM

Understanding the Basics

- Inference: the process of using a trained LLM to generate human like text from a given input prompt.
- LLMs generate this response from their knowledge they have gained during training.
- LLM generate one word at a time.

The Role of Attention

- Gives LLM ability to understand context and generate coherent response.
- While predicting the next word, not every word carries equal importance / weight.

Example:

The Capital of France is Paris

In short, the attention mechanism is the key to LLMs being able to generate text that is both coherent and context-aware. It sets modern LLMs apart from previous generations of language models.

Context Length Or Attention Span

- Context: maximum number of tokens that the LLM can process at once.

It is affected by:

- Model's architecture and size
- Available computational resources
- Complexity of input and desired output

The Art of Prompting

- The way we design our prompt to generate better results.

The Two Phase Inference Process

Inference is done in two phases

- The Prefill Phase
- The Decode Phase

The Prefill Phase

- Tokenization: converting text into tokens
- Embedded conversion: transforming these tokens into numerical representation that capture meaning
- Initial Processing: Passing embeddings to the neural network to create a rich understanding of the context.

Play with different tokenizer @ The Tokenizer Playground

The Decode Phase

- The model generates one token at a time known as autoregressive process.
- **Attention Computation**: Looking back at all previous tokens to understand context.
- **Probability Calculation**: Determining the likelihood of each possible next token.
- **Token Selection**: Choosing the next token based on these probabilities.
- **Continuation Check**: Deciding whether to continue or stop generation

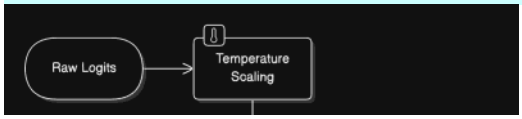
This phase is memory intensive.

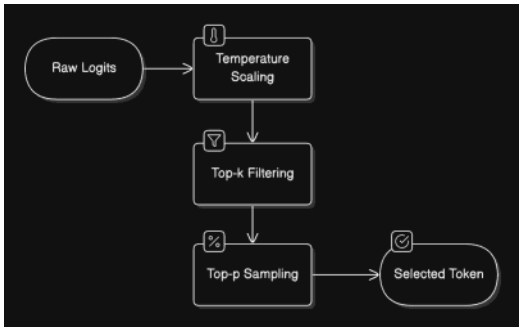
Sampling Strategies

How can we control text generation

Play with SmolLM2

Understanding Token Selection: From Probabilities to Token Choices





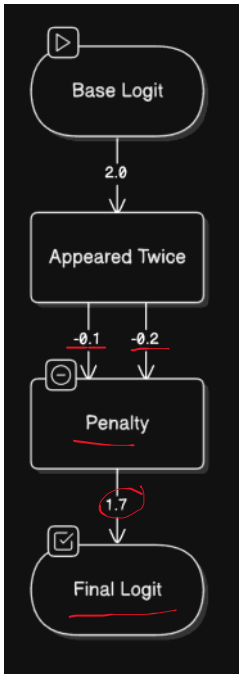
Raw Logits: *model's initial response before applying activation function·*
Temperature Control:
- *higher setting > 1·0 --> more random and creative·*
- *lower setting < 1·0 --> more focused and deterministic·*

Top-p (Nucleus) Sampling: *Instead of considering all possible words, we only look at the most likely ones that add up to our chosen probability threshold (e·g·, top 90%)*
Top-k Filtering: *An alternative approach where we only consider the k most likely next words·*

Managing Repetition: *Keeping Output Fresh*

LLMs repeat themselves - just like a speaker who keeps returning to the same points·

Presence Penalty: *If a token has appeared before, regardless of how often·*
Frequency Penalty: *Increases based on how often a token has been used·*



These penalties are applied early in the token selection process, adjusting the raw probabilities before other sampling strategies are applied· Think of them as gentle nudges encouraging the model to explore new vocabulary·

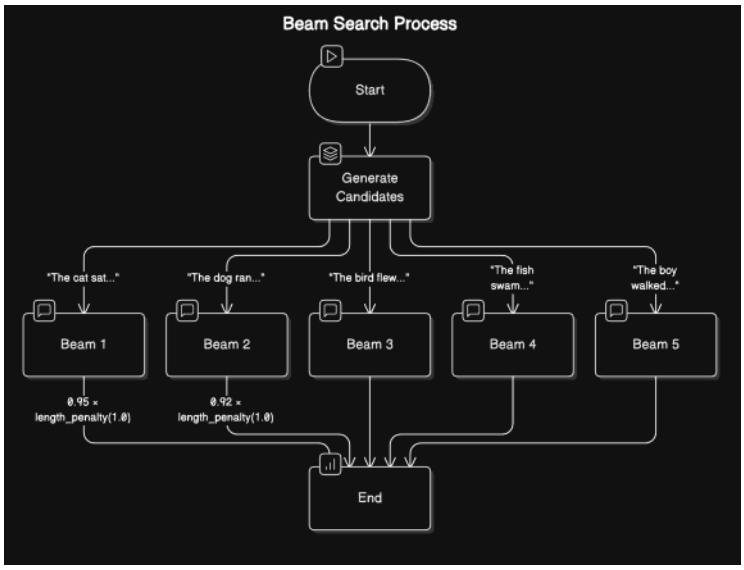
Controlling Generation Length: Setting Boundaries

We need to control how much text our LLM generates·
We can control in following ways:
Token Limits: *setting minimum and maximum token counts·*
Stop sequence: *specific patterns that signal the end of generation*
End-of-Sequence Detection: *let the model naturally conclude its response*

Beam Search: Looking Ahead for Better Coherence

Here's how it works:

- At each step, maintain multiple candidate sequences (typically 5-10)
- For each candidate, compute probabilities for the next token
- Keep only the most promising combinations of sequences and next tokens
- Continue this process until reaching the desired length or stop condition
- Select the sequence with the highest overall probability



Beam Search Visualizer

Practical Challenges and Optimization

Key Performance Metrics:
Time to First Token (TTFT): How quickly can you get the first response? This is crucial for user experience and is primarily affected by the prefill phase.
Time Per Output Token (TPOT): How fast can you generate subsequent tokens? This determines the overall generation speed.
Throughput: How many requests can you handle simultaneously? This affects scaling and cost efficiency.
VRAM Usage: How much GPU memory do you need? This often becomes the primary constraint in real-world applications.

The Context Length Challenge:

Memory Usage: Grows quadratically with context length
Processing Speed: Decreases linearly with longer contexts
Resource Allocation: Requires careful balancing of VRAM usage

Bias and limitations

Wednesday, May 7, 2025 10:16 PM

Attention Mechanism

Saturday, May 10, 2025 7:59 PM

A supplementary lecture.

- Roots back to **Recurrent Neural Networks**.

Input: I love you.

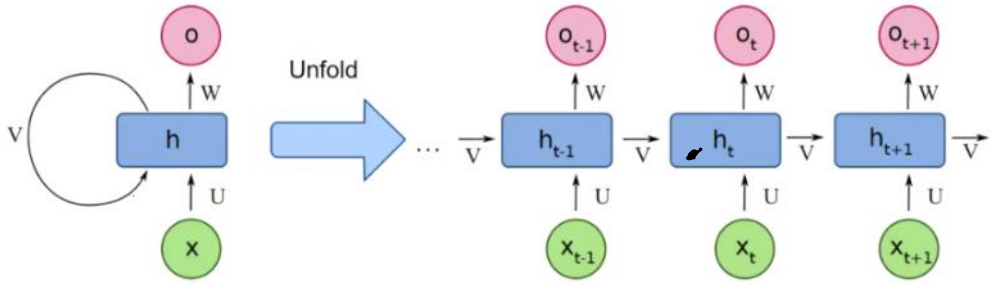
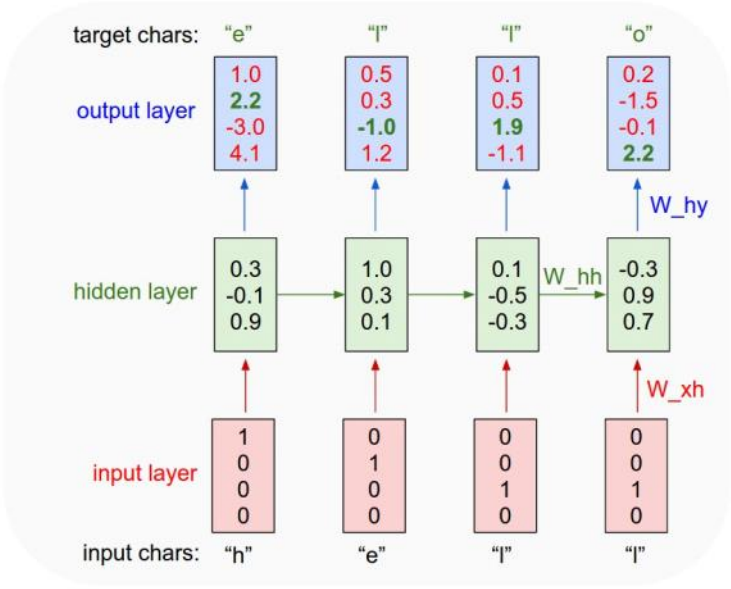
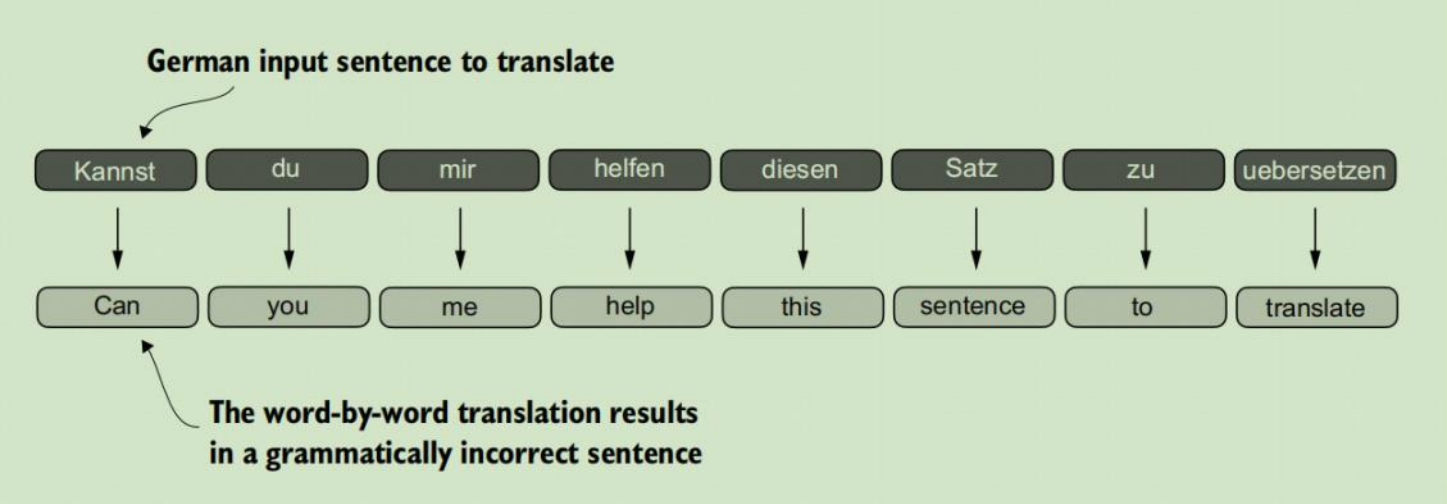


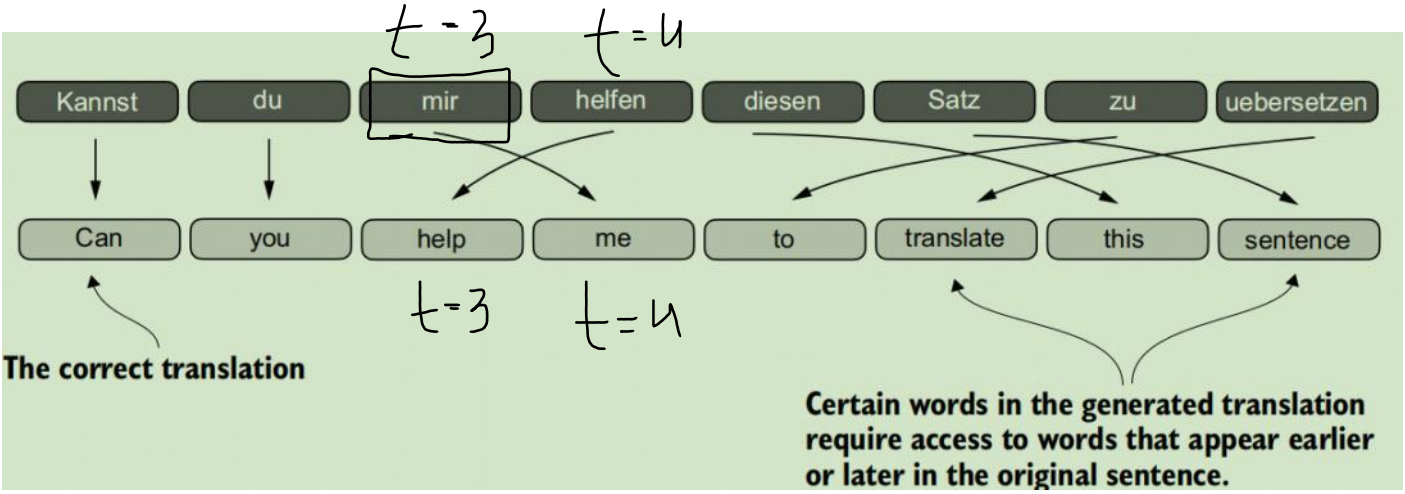
Image Source: [What is Recurrent Neural Networks (RNN)?]



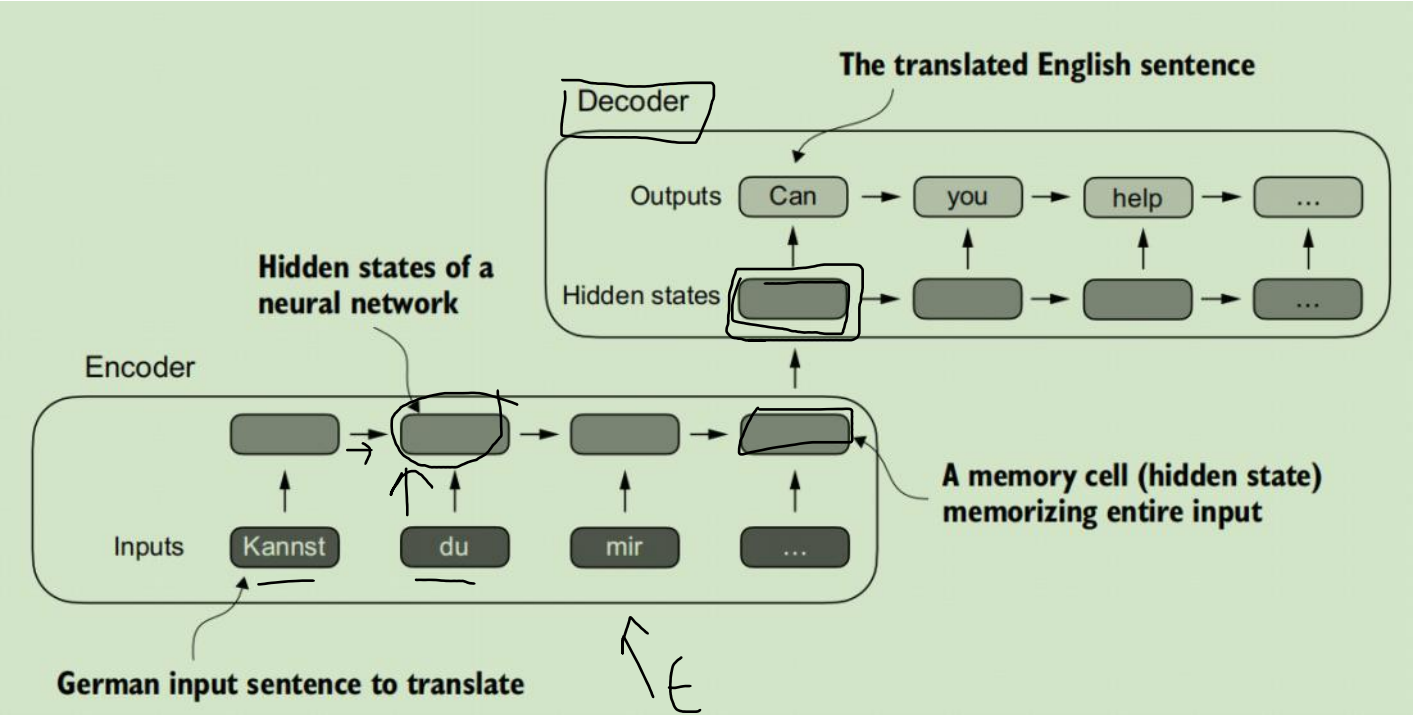
Source: [Visualizing RNNs]

- Doesn't work well with long sequences.





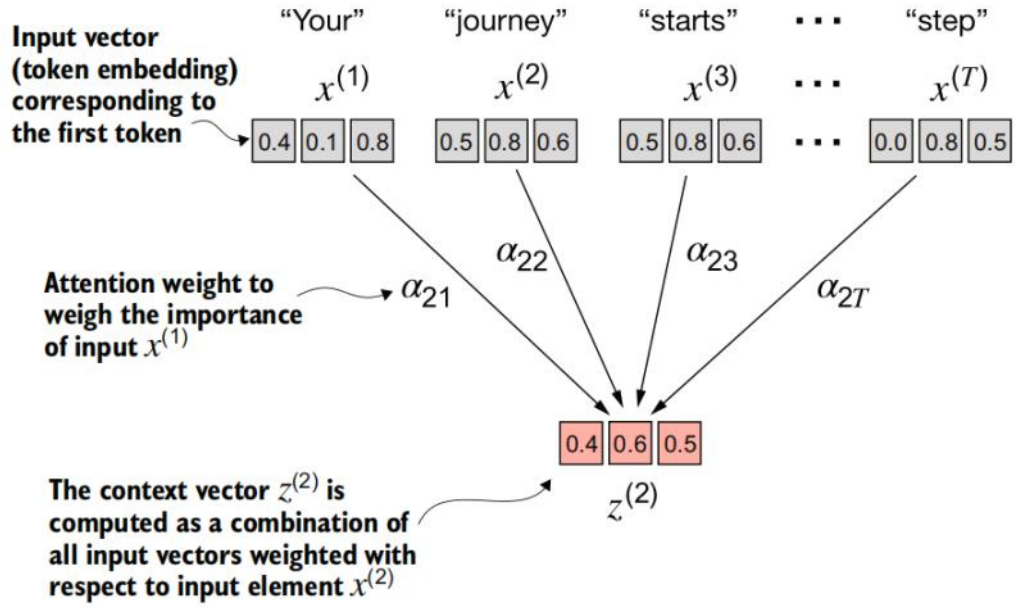
Since the RNN wasn't able to perform the translation task, encoder-decoder architecture came into highlight.



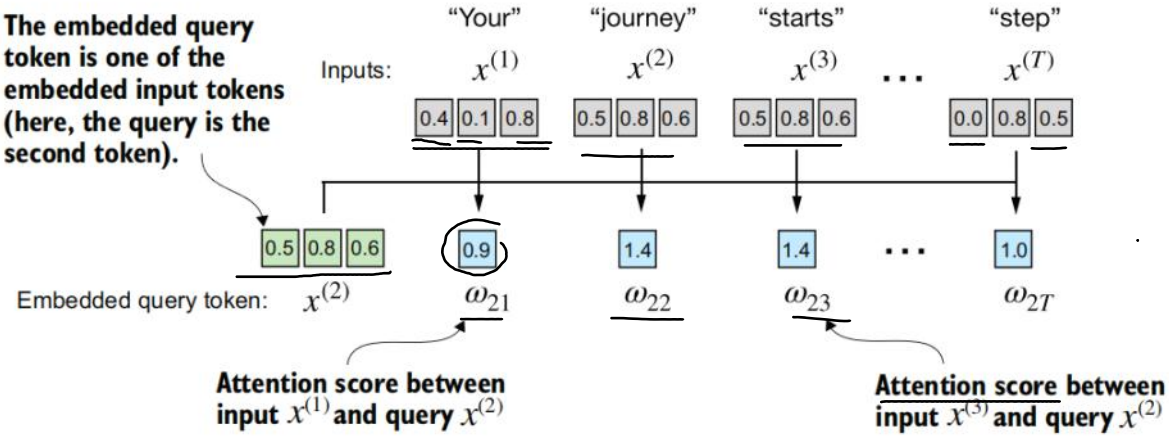
Let's visualize it : [Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)]

Self Attention:

- Attending to different parts of the inputs

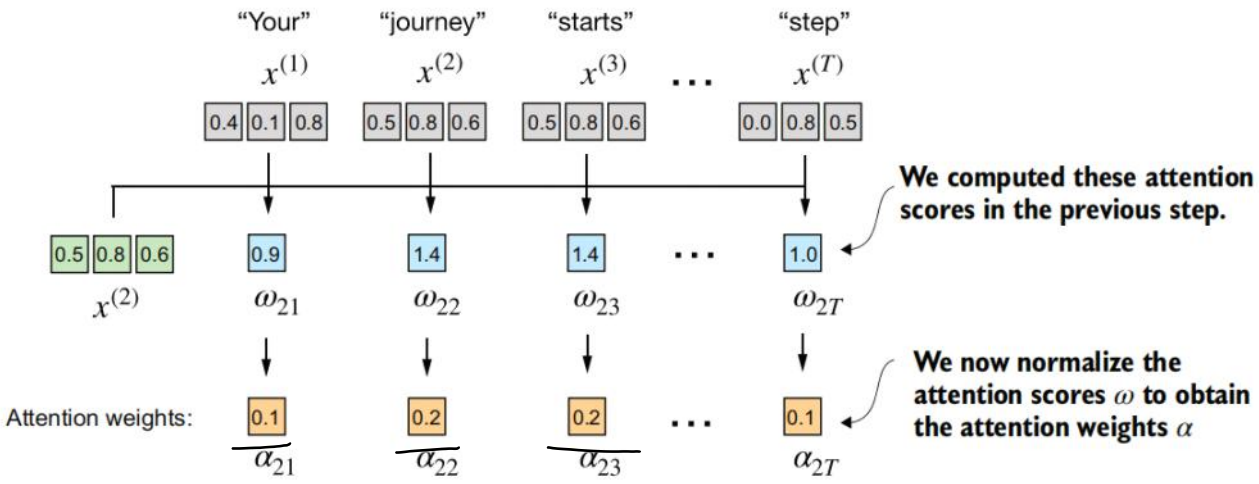



```
import torch
inputs = torch.tensor(
  [[0.43, 0.15, 0.89], # Your     (x^1)
   [0.55, 0.87, 0.66], # journey  (x^2)
   [0.57, 0.85, 0.64], # starts   (x^3)
   [0.22, 0.58, 0.33], # with     (x^4)
   [0.77, 0.25, 0.10], # one      (x^5)
   [0.05, 0.80, 0.55]] # step     (x^6)
)
```




```
tensor([0.9544, 1.4950, 1.4754, 0.8434, 0.7070, 1.0865])
```

```
tensor([0.9544, 1.4950, 1.4754, 0.8434, 0.7070, 1.0865])
```




```
Attention weights: tensor([0.1385, 0.2379, 0.2333, 0.1240, 0.1082, 0.1581])
Sum: tensor(1.)
```