INTERACTION WITH LARGE SCALE MODELS

LIACD/1

University of Beira Interior,
Department of Informatics

Hugo Pedro Proença,
hugomcp@di.ubi.pt, 2024/2025

INTERACTION WITH LARGE SCALE MODELS

[01]

- **Assiduity (A)** To get approved at this course, students should attend to - at least - 80% of the theoretical and practical classes;

- Practical Project (P) The practical projects of this course weights 50% (10/20) of the final mark.

- To get approved at the course, a minimal mark of 5/20 should be obtained in the practical project part;

- The practical project mark is conditioned to an individual presentation and discussion by each student;

- Written Test (F) Tuesday, June 3rd, 2025, 14:00. Room 6.17

- Mark (M) M = (A >= 0.8) * (P * 10/20 + F * 10/20)

- Admission to Exams Students with M >= 6 are admitted to final exams

- The practical projects mark is considered in all exam epochs;

Prompt Engineering? What is It?

- *"Prompt engineering is the practice of crafting inputs (prompts) to guide generative AI models toward producing desired outputs."*
    - Reynolds, L., & McDonell, K. (2021). "Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm."
- *"Prompt engineering involves optimizing the instructions given to AI models to maximize the relevance and accuracy of the generated responses."*
    - Brown, T., et al. (2020). "Language Models are Few-Shot Learners." *NeurIPS*.
- "*The process of designing effective input prompts to ensure large language models generate outputs aligned with user intentions and specific use cases*."
    - Reference: OpenAI. (2021). "Guidelines for Prompt Engineering with GPT-3."
- "*Prompt engineering refers to the process of designing and refining natural language prompts to harness the capabilities of large language models for downstream tasks."*
    - Google Research. (2022). "Understanding and Improving Few-Shot Prompting."

Prompt Engineering? Purposes

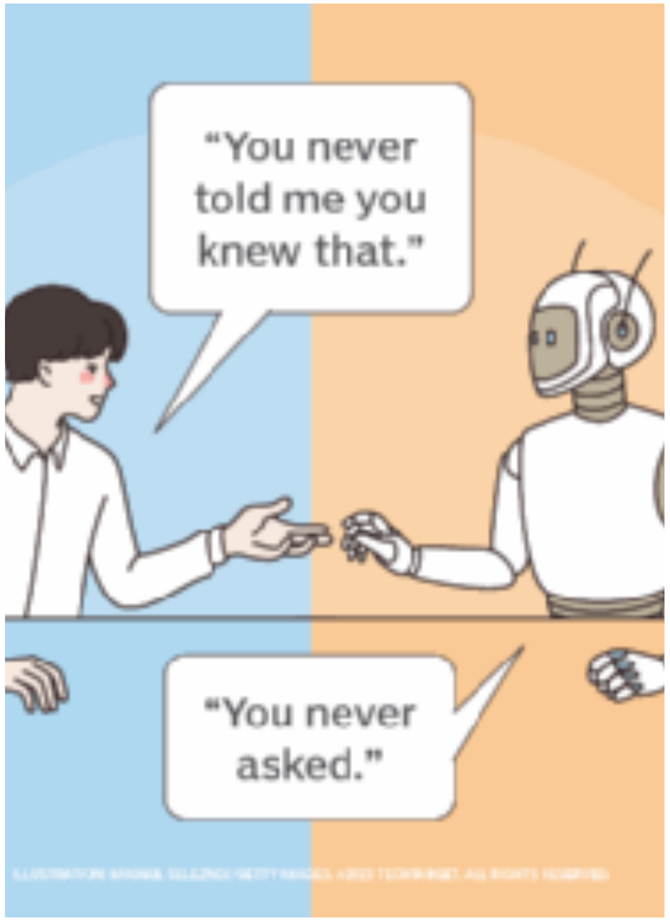
- Essentially, it’s about crafting inputs (prompts) that guide the AI to generate the most useful and relevant responses based on the task or goal at hand.
- **Improving Accuracy and Relevance**: By carefully crafting prompts, we can guide the AI to focus on the most relevant information, improving the quality and precision of the answers.
- **Enhancing Creativity**: In tasks where creativity is important (e.g., writing, brainstorming, or generating artwork), prompt engineering can steer the AI towards more imaginative, unique, or out-of-the-box responses.
- **Controlling Output Style and Tone**: Prompt engineering helps to influence the tone and style of the output, such as making it formal, casual, humorous, or professional. This is particularly useful for tasks that require a specific communication style.
- **Reducing Bias or Inappropriate Content**: Well-designed prompts can help mitigate bias, inappropriate, or harmful content generation by setting clearer boundaries and providing context or guidelines within the prompt itself.
- **Task-Specific Optimization**: Different tasks require different kinds of responses. A prompt for summarizing an article will be different from one used for answering a question. By adjusting the prompt for each task, you can optimize the AI’s output to match the intended objective.
- **Handling Ambiguity**: When the input is unclear or ambiguous, prompt engineering helps clarify and specify what information the user is looking for, leading to better, more accurate responses.
- **Encouraging Conciseness or Detail**: Depending on the goal, prompt engineering can be used to prompt for short, concise responses or longer, more detailed explanations.
- **Fostering Collaboration**: In collaborative tasks, like group brainstorming or coding, prompt engineering can help AI assist more effectively by focusing on collaborative feedback, suggestions, or adapting to team needs.

Essentials of Prompts

- A **prompt** is the input text or query provided to a language model, which serves as instructions for the model to generate a response. Prompts can vary in length and complexity, ranging from simple phrases to detailed instructions or context.
  OpenAI API Documentation, 2021

- A **prompt** is the initial textual input or context provided to a generative language model, designed to specify or guide the task that the model is expected to perform. The quality of the output is often highly dependent on the clarity and specificity of the prompt.
  Brown, T. B., et al. (2020). "Language Models Are Few-Shot Learners." *Advances in Neural Information Processing Systems (NeurIPS)*.

- A **prompt** refers to any form of structured or unstructured input data provided to a generative AI system, such as a text snippet, question, or context. It acts as the starting point for the system to produce text, images, or other outputs, depending on the task.
  Bommasani, R., et al. (2022). "On the Opportunities and Risks of Foundation Models." *Stanford Institute for Human-Centered Artificial Intelligence (HAI)*.



Source: https://www.techtarget.com/searchenterpriseai/definition/AI-prompt

Explicit Prompts

- Explicit prompts are clear, specific instructions provided to a generative model, detailing what the model is expected to do. These prompts minimize ambiguity by outlining the task clearly.
- **Characteristics**:
  - Direct instructions.
  - Well-defined structure and context.
  - High control over the output.
- **Examples**:
  - **Text Generation (ChatGPT):**
    "*Write a 100-word summary of the novel Pride and Prejudice by Jane Austen.*"
    Output: A concise summary of the book focusing on its main themes and characters.
  - **Image Generation (DALL-E):**
    "Generate an image of a futuristic city with flying cars and skyscrapers in a cyberpunk theme."
    Output: A vivid depiction of a futuristic cyberpunk cityscape.
  - **Code Generation:**
    "Write a Python function to calculate the factorial of a number using recursion."
    Output: A Python function explicitly performing the task.

Implicit Prompts

- Implicit prompts rely on less direct instructions or contextual cues to guide the model. These prompts often mimic human conversational inputs or vague requests, requiring the model to infer the task or context.
- **Characteristics**:
  - Less direct, conversational, or ambiguous.
  - Requires the model to "guess" the intent.
  - Greater reliance on the model's training data and context.
- **Examples**:
  - **Text Generation (ChatGPT):**
    "*What do you think happens in Pride and Prejudice?*"
    Output: A general description or interpretation of the book, potentially omitting specific details.
  - **Image Generation (DALL-E):**
    "*Imagine a futuristic world.*"
    Output: A creative but potentially varied interpretation of a futuristic setting.
  - **Code Generation:**
    "*How can I find the factorial of a number in Python?*"
    Output: An explanation or example of calculating factorials, possibly using loops or recursion.

Explicit/Implicit Prompts

Aspect	Explicit Prompt	Implicit Prompt
Clarity	Highly specific and unambiguous	Conversational or vague; requires inference
Control	Offers more control over the output	Relies more on the model's interpretation
Use Case	Ideal for task-specific outputs	Suited for open-ended or creative tasks
Example Prompt	"Write a poem about autumn in 4 lines."	"Can you write something about autumn?"
Output	A short, structured poem on autumn.	A broader response, possibly less structured.

Explicit prompts are ideal for when we need precise and reliable results. **Implicit prompts** are better for brainstorming, open-ended creativity, or when we don't want to over-constrain the model.

Structure of Prompts

- **A. Context**

  Providing background information or setting the stage for the task.

  "*You are a travel guide specializing in European destinations. Write a 200-word description of Paris.*"

- **B. Task Instruction**

  Clearly specifying what the model is expected to do.

  "Summarize the following text in three sentences."

- **C. Constraints**

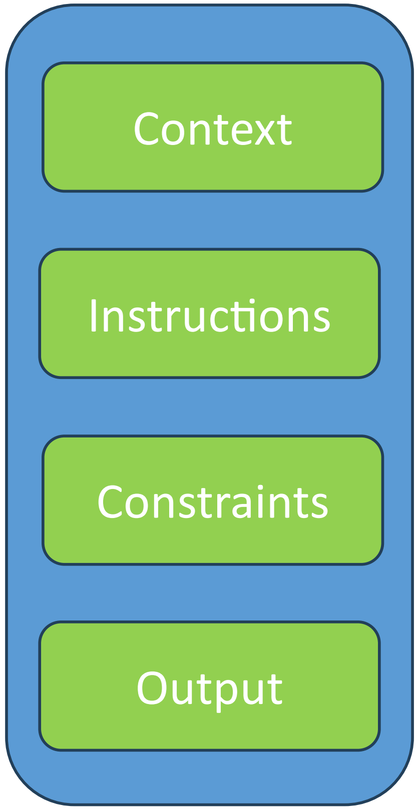
  Defining rules, boundaries, or limitations for the response.

  "Generate a response in less than 50 words."

- **D. Desired Output Format**

  Specifying the structure, tone, or format of the output.

  "Write the response as a bullet-point list."



Any set up to help the intent clear and objective (role play, background, expertise, examples,…)

What to do and how to do it, as if the instructions are given to an 8-year child

What to avoid, and what rules to obey, that should be met regardless of the generated output

What to avoid, and what rules to obey, that should be met regardless of the generated output

Good/Bad Prompts

- **Good Prompt**:
  "Write a 150-word summary of the article linked below, focusing on the main argument and supporting evidence."
- **Good Prompt**:
  "You are an academic tutor. Explain the Pythagorean theorem using a step-by-step example with a right triangle where the sides are 3, 4, and 5."

✓

- **Bad Prompt**:
  "Summarize this article."
- **Bad Prompt:**
  "What is the Pythagorean theorem?"

✗

Role Assignment
Assigning a role to the AI to set a specific perspective or tone. "*You are an expert chef. Write a recipe for a simple vegetarian pasta dish.*"

Step-by-Step
Breaking down complex tasks into smaller, clear steps. "*Step 1: Read the text below. Step 2: Identify the main argument. Step 3: Write a 100-word summary.*"

Use Variables
Including placeholders or dynamic variables for reusable prompts.
"Generate a tagline for a company named [CompanyName] that specializes in [Industry]."

Use Examples
Including examples to clarify the task. *"Translate English to French. Example: 'Hello' -> 'Bonjour'. Now translate: 'Goodbye'."*

Chaining Prompts
Using multiple, sequential prompts to guide a multi-step process.
Prompt 1: "Summarize the given text."
Prompt 2: "Rewrite the summary in simpler language for a younger audience."

Types of Prompts – Zero-Shot

- **Definition**: The model is asked to perform a task without being given any examples or demonstrations. The prompt relies entirely on the model's training to understand and execute the task.

- **Characteristics**:
  - No prior examples are provided in the prompt.
  - Used when the model is expected to generalize from its training data.

- **Examples**:
  - **Text Generation**:
    "*Summarize the following paragraph: The quick brown fox jumps over the lazy dog. The dog wakes up startled and chases the fox into the woods.*"
    ***Output*:** "A fox jumps over a dog, who wakes up and chases the fox."

  - **Code Generation**:
    "Write a Python function to reverse a string."
    ***Output*:** A Python function that performs the string reversal task.

Types of Prompts – One-Shot

- **Definition**: The model is given one example of the task before being asked to perform it. This helps provide minimal context while remaining efficient.

- **Characteristics**:
  - Includes a single example of the task in the prompt.
  - Useful when the model might benefit from seeing one example.

- **Examples**:
  - **Text Generation**:
    "*Here is an example of translating English to French: 'Hello' -> 'Bonjour'. Translate the following: 'Good morning'.*"
    Output**:** "Bon matin" or "Bonjour."

  - **Code Generation**:
    "*Example: Input: [1, 2, 3] -> Output: [3, 2, 1]. Now reverse this list: [4, 5, 6].*"
    Output**:** "[6, 5, 4]".

Types of Prompts – Few-Shot

- The model is given multiple examples of the task before being asked to perform it. This provides more context and helps the model better understand the desired format or behavior.
- **Characteristics**:
    - Includes 2–5 examples of the task in the prompt.
    - Useful for complex tasks or when the model needs additional context.
- **Examples**:
    - **Text Generation**:
      “Translate English to French: 'Hello' -> 'Bonjour' Example 2: Translate English to French: 'How are you?' -> 'Comment ça va?' Translate the following: 'Thank you.’

      ***Output*:** "Merci."
    - **Code Generation**:

      “*Example 1: Input: [1, 2, 3] -> Output: [3, 2, 1]. Example 2: Input: ['a', 'b', 'c'] -> Output: ['c', 'b', 'a']. Now reverse this list: [4, 5, 6].*”

      ***Output*:** "[6, 5, 4]".

Open-Ended Prompts

- **Definition**: Prompts that allow the model to generate creative, unrestricted responses. They encourage exploration and variety in the output.
- **Characteristics**:
  - No rigid format or constraint.
  - Useful for brainstorming, storytelling, or creative tasks.
- **Examples**:
  - **Text Generation**:
    "*Describe a futuristic city.*"
    Output**:** "A sprawling metropolis with levitating cars, towering skyscrapers made of translucent materials, and a glowing green sky powered by renewable energy."
  - **Image Generation**:
    "Create an artwork that represents peace."

Closed Prompts

- **Definition**: Prompts that restrict the model's response by providing clear guidelines or constraints on the format and content of the output.
- **Characteristics**:
  - Structured and specific.
  - Used for tasks requiring precision, such as question-answering or summarization.
- **Examples**:
  - **Text Generation**: "*Summarize this in 20 words: The quick brown fox jumps over the lazy dog. The dog wakes up startled and chases the fox into the woods.*" ***Output:*** "A fox jumps over a dog. The dog wakes up startled and chases the fox into the woods."
  - **Image Generation**: "*Generate an image of a red apple on a wooden table in a minimalist style.*"

Challenges and Limitations

- **Sensitivity to Prompt Wording**
- **Challenge**: Models can be highly sensitive to slight changes in prompt phrasing, which may drastically alter the output.

- **Example**:

  "*Summarize the key points of this article.*"
  Output: A short summary focusing on the most relevant points.

  "What is this article about?"
  ***Output*:** A general and sometimes vague response.
- **Impact**: Users may need to experiment with multiple iterations to achieve the desired output.

Challenges and Limitations

- **Lack of Robustness to Ambiguity**
- **Challenge**: Vague or ambiguous prompts lead to inconsistent or irrelevant outputs. The model cannot always infer user intent without sufficient clarity.

- **Example**:

  "Explain photosynthesis."
  Output**:** The explanation could be too detailed for a layperson or too simplified for a botanist.
- **Impact**: Requires precise language to ensure the output aligns with user expectations.

Challenges and Limitations

- **Balancing Specificity and Flexibility**
- **Challenge**: Overly specific prompts can restrict creativity, while overly flexible prompts may produce unstructured or irrelevant outputs.

- **Example**:

  Highly Specific: *"Write a 200-word essay about renewable energy, focusing on solar power, using formal language."*

  Too Flexible: *"Write about renewable energy."*

  Balanced: *"Write a short essay about renewable energy, highlighting its benefits and challenges."*

- **Impact**: Finding the right balance often involves trial and error.

Challenges and Limitations

- **Task Complexity**
- **Challenge**: For complex tasks requiring multiple steps or nuanced understanding, a single prompt may fail to guide the model effectively.

- **Example**:

  "Write a report analyzing the economic impacts of climate change and propose solutions."
  Issue: The model may focus more on one part of the task and neglect the other.
- **Solution**: Use chained or step-by-step prompts to address different aspects of the task sequentially.

Challenges and Limitations

- **Prompt Length Limitations**
- **Challenge**: Models like GPT have token limits (e.g., ~4,000 tokens for GPT-3.5 or ~8,000 tokens for GPT-4), which can restrict the length of both prompts and outputs.
- **Example**:
  - A prompt requiring extensive context or multiple examples may exceed the token limit, truncating the response.
- **Impact**: Users must prioritize and condense information while crafting prompts.

Generative AI Applications

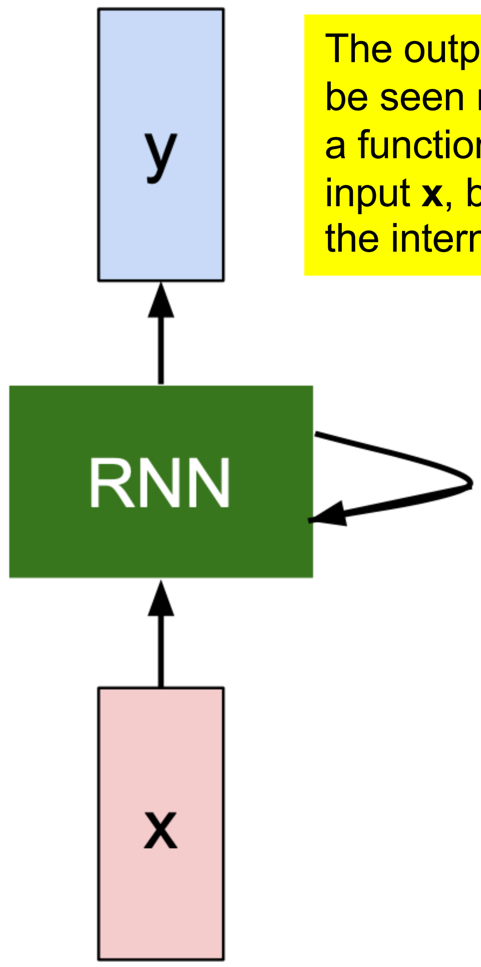
- Generative AI is increasingly used in content creation, enabling creators to produce lifelike AI-generated videos, automate scripting, editing, and narration, thereby significantly reducing production time and costs.
- **Example**: Content creators are leveraging AI tools to generate videos, manage e-commerce, and handle social media tasks, enhancing productivity and expanding their reach.
- **Illustrative Video**: "Meet the content creators harnessing AI - and how they use it to make thousands per month"

Generative AI Applications

- In the film industry, generative AI is utilized to modify actors' performances, such as adjusting accents or enhancing specific aspects of dialogue delivery, to achieve a desired level of authenticity.
- **Example**: The film "The Brutalist" employed AI tools to refine the Hungarian accents of its stars, ensuring accurate pronunciation and enhancing the overall authenticity of the performances.
- **Illustrative Video**: "The Brutalist's AI Controversy, Explained"

Recurrent Neural Networks

- Recurrent Neural Networks (**RNNs**) are deep learning model typically used to process and convert a **sequential data input** into a sequential data **output**.
- **Sequential data**—such as words, sentences, or time-series— have interrelated sequential components, based on complex semantics and syntax rules.
- The key idea in RNNs is to use (apart the classical "weights") an **internal state** that is updated as a sequence is processed



The output **y** can be seen not only as a function of the input **x**, but also of the internal state **h**

Recurrent Neural Networks

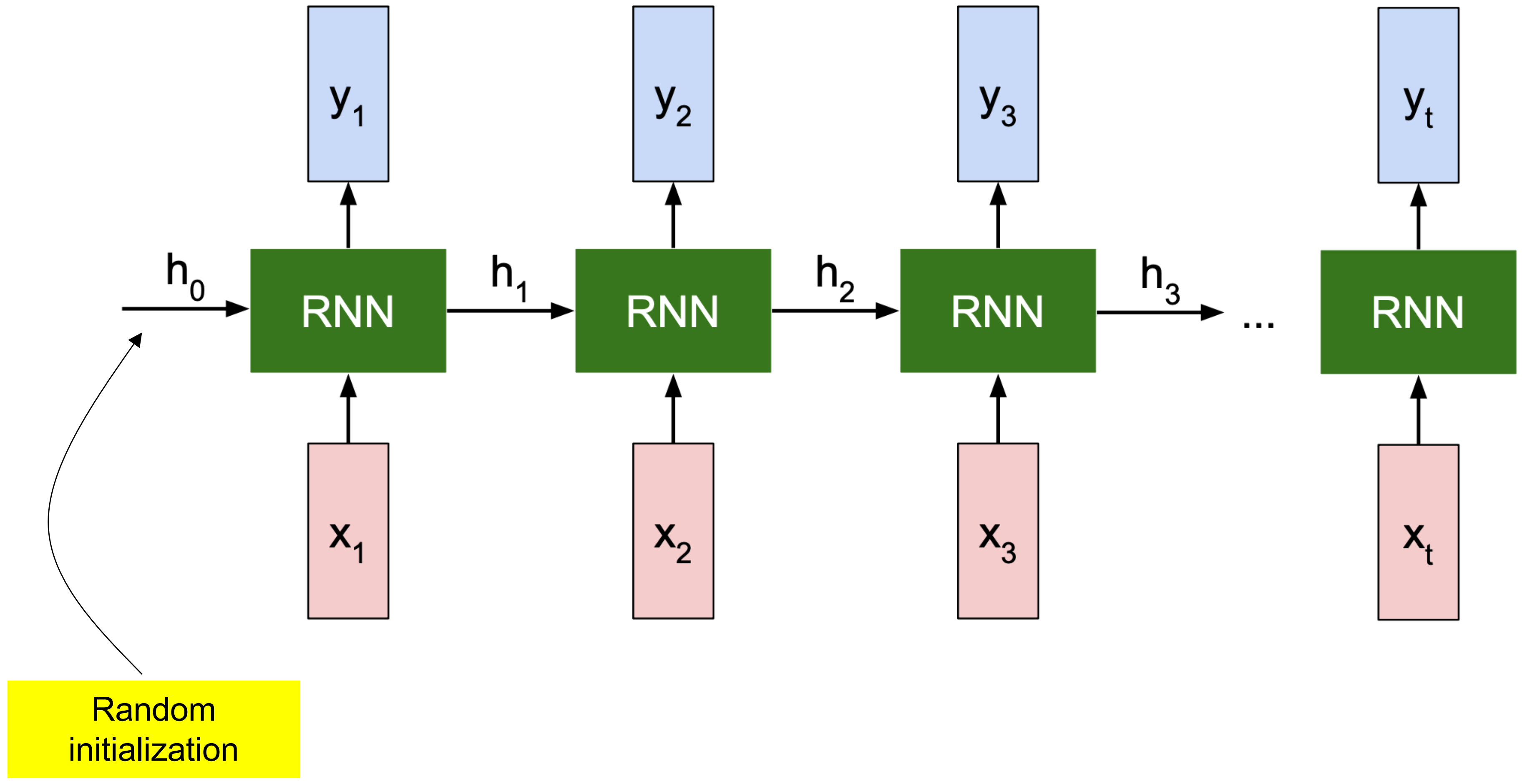
- The forward step of RNNs is divided into two phases:
    - **Step 1:** Obtain the hidden state at time "t" ($\boldsymbol{h}_t$), given the input at time "t" ($\boldsymbol{x}_t$), and the previous state ($\boldsymbol{h}_{t-1}$).

$$\underset{\text{new state}}{\boxed{h_t}} = \underset{\text{some function with parameters W}}{\boxed{f_W}}(\underset{\text{old state}}{\boxed{h_{t-1}}}, \underset{\text{input vector at some time step}}{\boxed{x_t}})$$

-
    - **Step 2:** Then, obtain the output at time "t" ($\boldsymbol{y}_t$), using the recently updated state ($\boldsymbol{h}_t$).

$$\underset{\text{output}}{\boxed{y_t}} = \underset{\text{another function with parameters } W_o}{\boxed{f_{W_{hy}}}}(\underset{\text{new state}}{\boxed{h_t}})$$

Recurrent Neural Networks



Recurrent Neural Networks

- **Step 1.** To obtain the hidden state at time “t” ($\boldsymbol{h_t}$) , we process a set of inputs ($\boldsymbol{x_i}$) , using the same function $\boldsymbol{f_W}$ at every step.
- In practice, this is due to the fact that backpropagation (weights update) is only done after a batch of steps.

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

- The pioneer architecture (Vanilla RNN) assumes that the state ($\boldsymbol{h_t}$) is a single hidden vector in the network.
  - “s” is the dimension of the input/output space, and “d” is a hyper-parameter of the RNN.

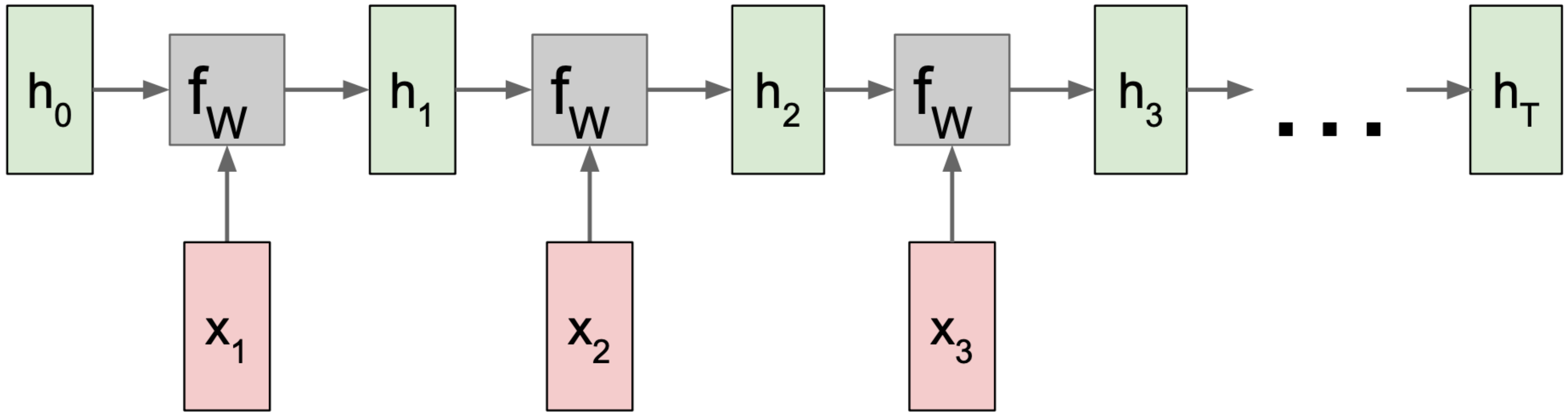
$$\underbrace{h_t}_{\text{[d x 1] vector}} = \tanh(\underbrace{W_{hh}}_{\text{[d x d] Matrix}} \underbrace{h_{t-1}}_{\text{[d x 1] vector}} + \underbrace{W_{xh}}_{\text{[d x s] Matrix}} \underbrace{x_t}_{\text{[s x 1] vector}})$$

Recurrent Neural Networks

- **Step 2.** Once $\boldsymbol{h}_t$ is found, the output at time "t" $(\boldsymbol{y}_t)$, can also be obtained

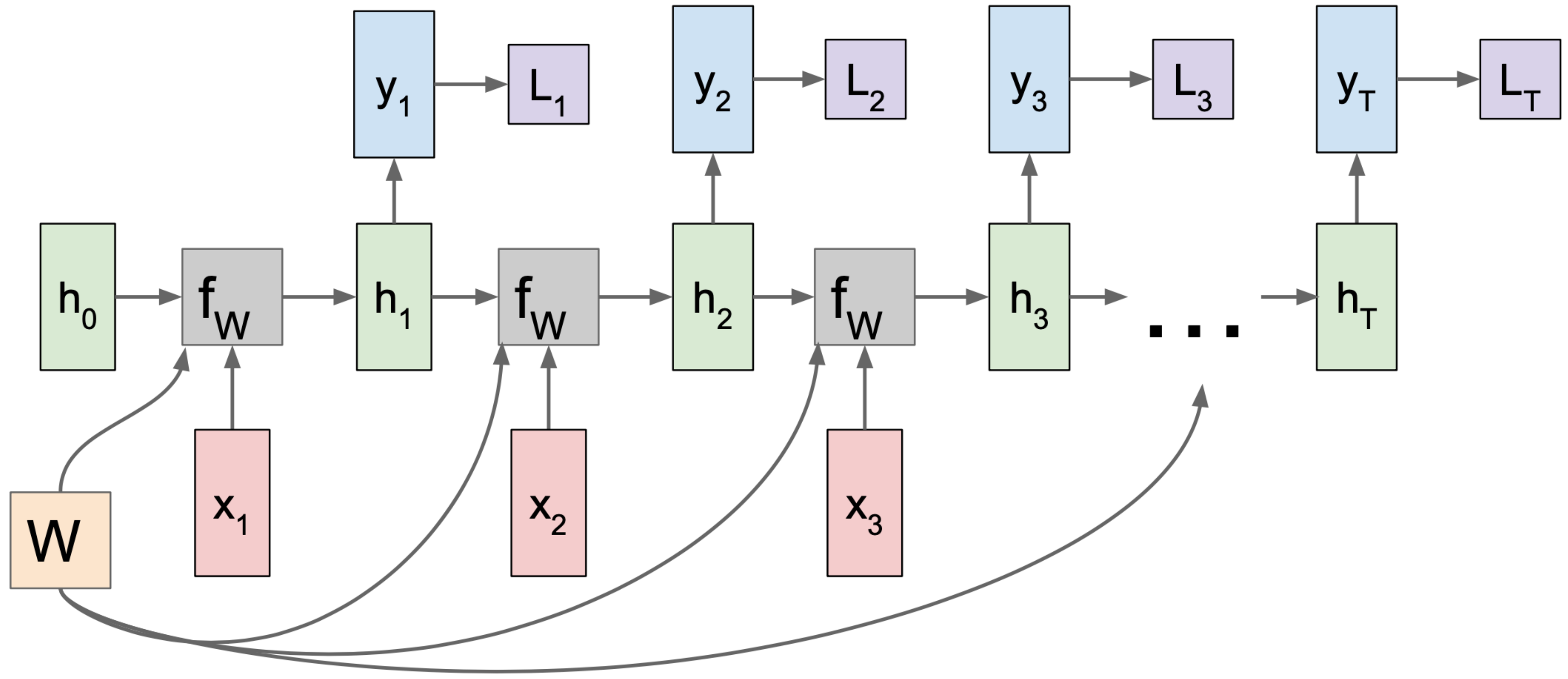
$$\underbrace{y_t}_{\text{[s x 1] vector}} = \underbrace{W_{hy}}_{\text{[s x d] Matrix}} \underbrace{h_t}_{\text{[d x 1] vector}}$$

- Hence, the first step of the corresponding computational graph is given by:



Recurrent Neural Networks

- Only at the second step, the outputs $(\boldsymbol{y}_t)$ are obtained and the partial losses found.
- Such partial loss values are then used to obtain the final loss $\mathcal{L}$ that will be used in backpropagation.



Recurrent Neural Networks: Example

- Text Generation. Consider a single training sequence (“hello”).
- The vocabulary is a set of four symbols: {“h”, “e”, “l”, “o”}
- We start by obtaining a latent representation of each element in the training set. The simplest one is the hot-one encoding.
- $h \rightarrow [1, 0, 0, 0]^T$; $e \rightarrow [0, 1, 0, 0]^T$; $l \rightarrow [0, 0, 1, 0]^T$; $\mathrm{o} \rightarrow [0, 0, 0, 1]^T$
- More sophisticated content generation techniques (e.g., Chat GPT) obtain richer representations, which elements lie in topological spaces (i.e., neighbor representations are related or are alike).
- It is reported that these representations play a very important role in the final effectiveness of the model.
- In this example, we are working at the character level. However, “word” or even “small sentence” levels can also be considered.
- “cat” $\rightarrow [1, 0, \ldots, 0, 0]^T$; $dog \rightarrow [0, 1, \ldots, 0, 0]^T$;

Recurrent Neural Networks: Example

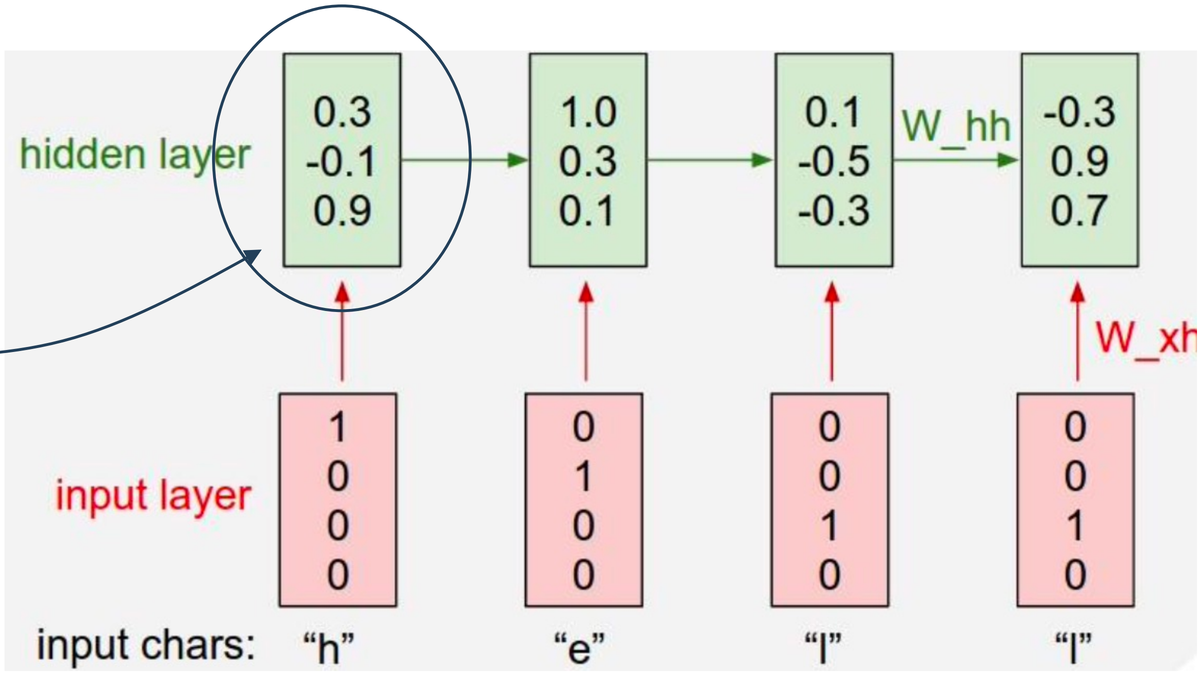
- **Step 1**. Obtain the hidden state representations ($\boldsymbol{h}_t$) for the training sequence (“hell”).

Why isn’t the complete set considered?

- Suppose that ($\boldsymbol{W}_{hh}$) and ($\boldsymbol{W}_{xh}$) were initialized randomly.

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

($\boldsymbol{h}_t$)



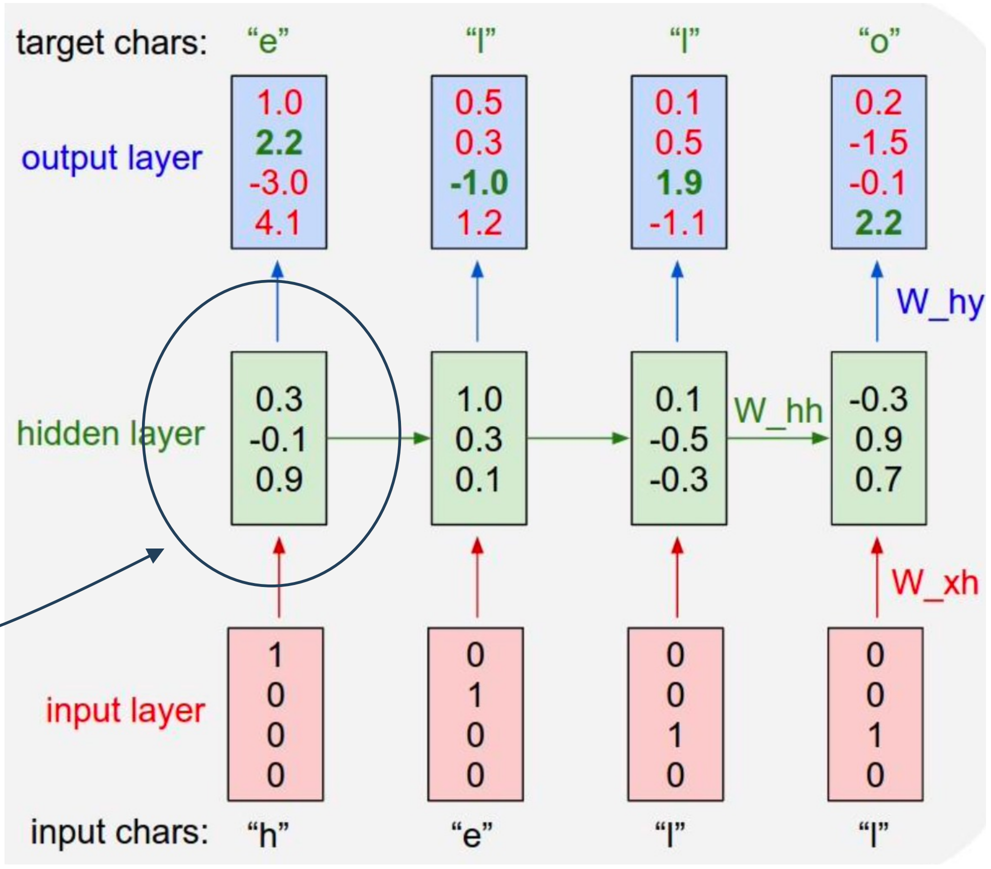
Recurrent Neural Networks: Example

- **Step 2**. Next, we can obtain the predicted elements at each time.

$$y_t = W_{hy}h_t$$

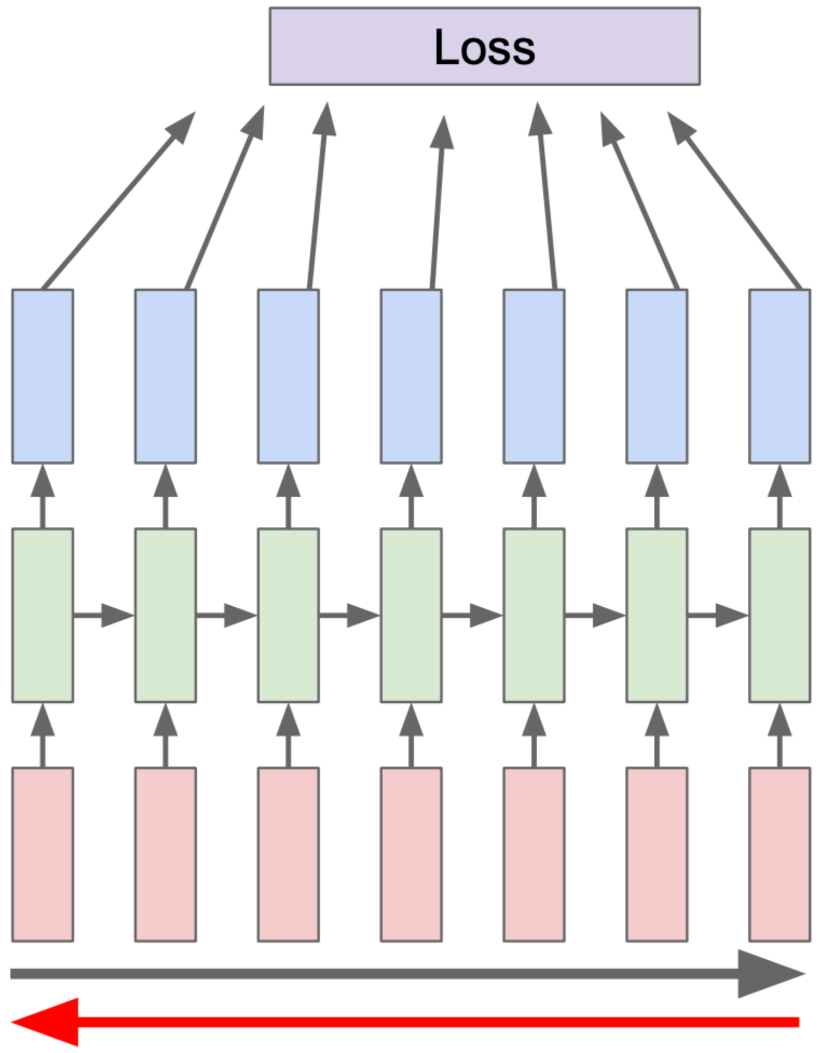
- Again, suppose that ($\boldsymbol{W}_{hy}$) was initialized randomly.

$\boldsymbol{h}_t$ obtained in Step 1



Recurrent Neural Networks: Example

- During training, we forward during the entire sequence to obtain the loss, and then backpropagate to obtain the gradientes and adjust weights.
- However, in practice, we run forward/backward through "**chunks**" instead of the whole sequence.
- This is the equivalent to the notion "**batch**" in classical CNNs architectures

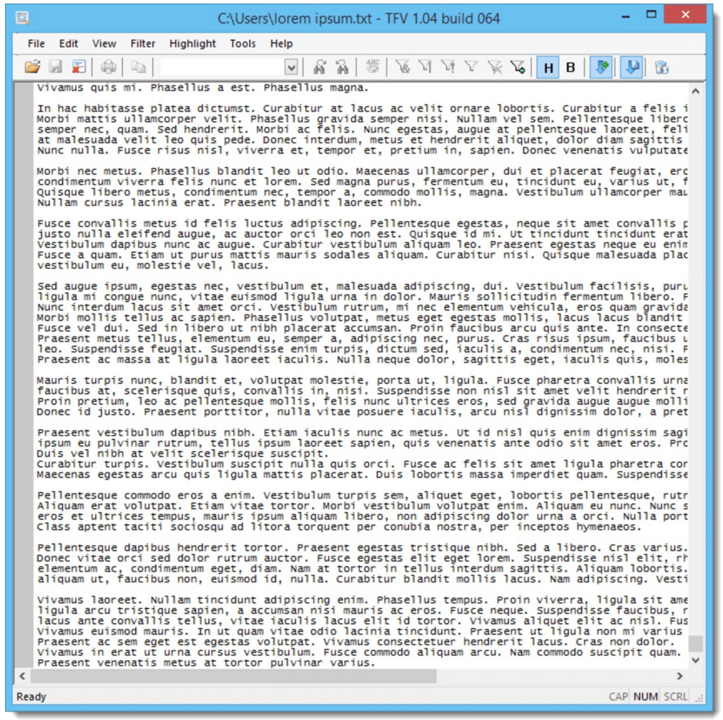


Recurrent Neural Networks: Example

- A minimal example (in 112 lines of Python) is available at the web page of this course. It contains a "Vanila" RNN learning process, depending exclusively of "numpy" library.

Credits: Andrej Karpathy

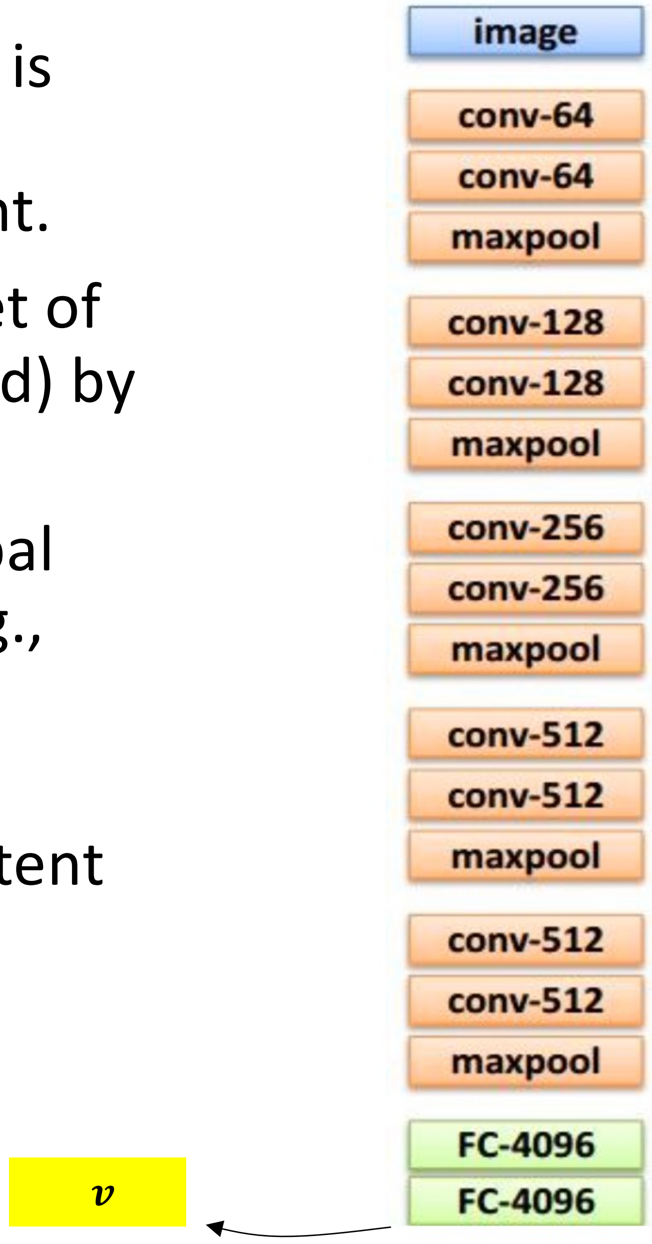
- Based in a simple plain text file (input.txt") it learns to generate text.



```
Vivamus quis mi. Phasellus a est. Phasellus magna.

In hac habitasse platea dictumst. Curabitur at lacus ac velit ornare lobortis. Curabitur a felis i
Morbi mattis ullamcorper velit. Phasellus gravida semper nisi. Nullam vel sem. Pellentesque liberc
semper nec, quam. Sed hendrerit. Morbi ac felis. Nunc egestas, augue at pellentesque laoreet, feli
at malesuada velit leo quis pede. Donec interdum, metus et hendrerit aliquet, dolor diam sagittis
Nunc nulla. Fusce risus nisl, viverra et, tempor et, pretium in, sapien. Donec venenatis vulputate

Morbi nec metus. Phasellus blandit leo ut odio. Maecenas ullamcorper, dui et placerat feugiat, erc
condimentum viverra felis nunc et lorem. Sed magna purus, fermentum eu, tincidunt eu, varius ut, f
Quisque libero metus, condimentum nec, tempor a, commodo mollis, magna. Vestibulum ullamcorper mau
Nullam cursus lacinia erat. Praesent blandit laoreet nibh.

Fusce convallis metus id felis luctus adipiscing. Pellentesque egestas, neque sit amet convallis p
justo nulla eleifend augue, ac auctor orci leo non est. Quisque id mi. Ut tincidunt tincidunt erat
Vestibulum dapibus nunc ac augue. Curabitur vestibulum aliquam leo. Praesent egestas neque eu enim
Fusce a quam. Etiam ut purus mattis mauris sodales aliquam. Curabitur nisi. Quisque malesuada plac
vestibulum eu, molestie vel, lacus.

Sed augue ipsum, egestas nec, vestibulum et, malesuada adipiscing, dui. Vestibulum facilisis, puru
ligula mi congue nunc, vitae euismod ligula urna in dolor. Mauris sollicitudin fermentum libero. F
Nunc interdum lacus sit amet orci. Vestibulum rutrum, mi nec elementum vehicula, eros quam gravida
Morbi mollis tellus ac sapien. Phasellus volutpat, metus eget egestas mollis, lacus lacus blandit
Fusce vel dui. Sed in libero ut nibh placerat accumsan. Proin faucibus arcu quis ante. In consecte
Praesent metus tellus, elementum eu, semper a, adipiscing nec, purus. Cras risus ipsum, faucibus u
leo. Suspendisse feugiat. Suspendisse enim turpis, dictum sed, iaculis a, condimentum nec, nisi. F
Praesent ac massa at ligula laoreet iaculis. Nulla neque dolor, sagittis eget, iaculis quis, moles

Mauris turpis nunc, blandit et, volutpat molestie, porta ut, ligula. Fusce pharetra convallis urna
faucibus at, scelerisque quis, convallis in, nisi. Suspendisse non nisl sit amet velit hendrerit r
Proin pretium, leo ac pellentesque mollis, felis nunc ultrices eros, sed gravida augue augue molli
Donec id justo. Praesent porttitor, nulla vitae posuere iaculis, arcu nisl dignissim dolor, a pret

Praesent vestibulum dapibus nibh. Etiam iaculis nunc ac metus. Ut id nisl quis enim dignissim sagi
ipsum eu pulvinar rutrum, tellus ipsum laoreet sapien, quis venenatis ante odio sit amet eros. Prc
Duis vel nibh at velit scelerisque suscipit.
Curabitur turpis. Vestibulum suscipit nulla quis orci. Fusce ac felis sit amet ligula pharetra cor
Maecenas egestas arcu quis ligula mattis placerat. Duis lobortis massa imperdiet quam. Suspendisse

Pellentesque commodo eros a enim. Vestibulum turpis sem, aliquet eget, lobortis pellentesque, rutr
Aliquam erat volutpat. Etiam vitae tortor. Morbi vestibulum volutpat enim. Aliquam eu nunc. Nunc s
eros et ultrices tempus, mauris ipsum aliquam libero, non adipiscing dolor urna a orci. Nulla port
Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos.

Pellentesque dapibus hendrerit tortor. Praesent egestas tristique nibh. Sed a libero. Cras varius.
Donec vitae orci sed dolor rutrum auctor. Fusce egestas elit eget lorem. Suspendisse nisl elit, rh
elementum ac, condimentum eget, diam. Nam at tortor in tellus interdum sagittis. Aliquam lobortis.
aliquam ut, faucibus non, euismod id, nulla. Curabitur blandit mollis lacus. Nam adipiscing. Vesti

Vivamus laoreet. Nullam tincidunt adipiscing enim. Phasellus tempus. Proin viverra, ligula sit ame
ligula arcu tristique sapien, a accumsan nisi mauris ac eros. Fusce neque. Suspendisse faucibus, r
lacus ante convallis tellus, vitae iaculis lacus elit id tortor. Vivamus aliquet elit ac nisl. Fus
Vivamus euismod mauris. In ut quam vitae odio lacinia tincidunt. Praesent ut ligula non mi varius
Praesent ac sem eget est egestas volutpat. Vivamus consectetuer hendrerit lacus. Cras non dolor.
Vivamus in erat ut urna cursus vestibulum. Fusce commodo aliquam arcu. Nam commodo suscipit quam.
Praesent venenatis metus at tortor pulvinar varius.
```

Recurrent Neural Networks: Applications

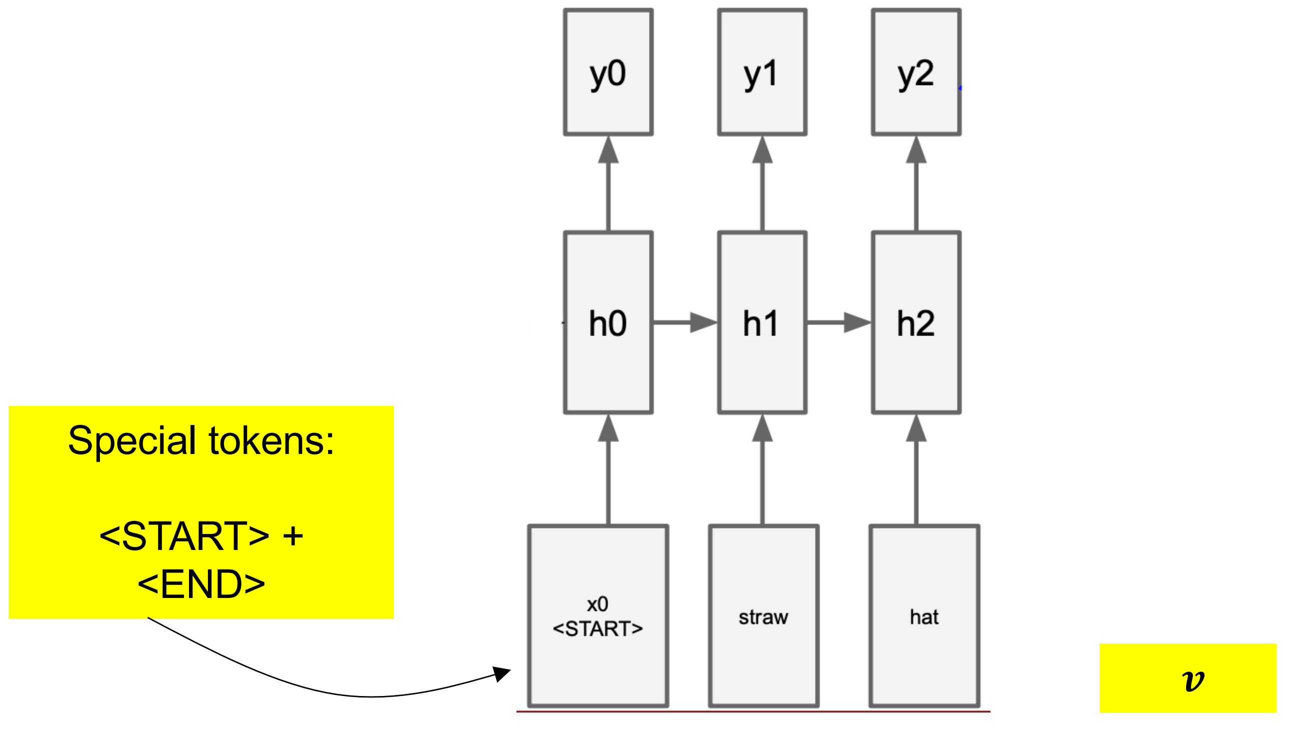
- One interesting application of RNNs is "**Image Captioning**", that regards to obtain descriptions for visual content.
- The learning set is composed of a set of images previously labeled (captioned) by humans.
- A classical CNN architecture for global image classification can be used (e.g., VGG or ResNet), removing the final classification layer.
- We use the highest-level possible latent representation



Recurrent Neural Networks: Applications

- The latent representation $\boldsymbol{v}$ is also considered by the RNN, fusing text $\boldsymbol{x}$ to visual information $\boldsymbol{v}$
- A new weights matrix $\boldsymbol{W}_{ih}$ is also required

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$



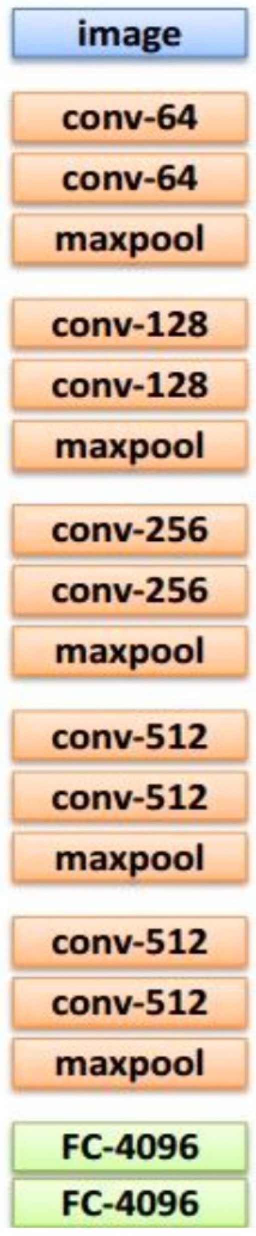
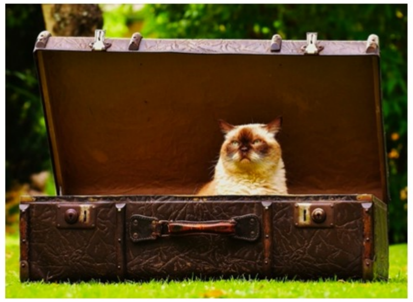
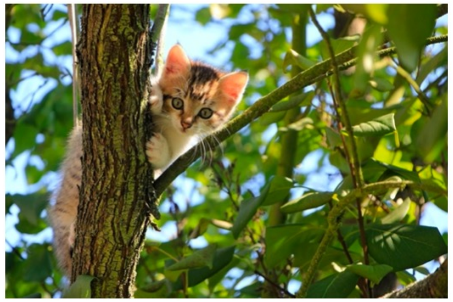
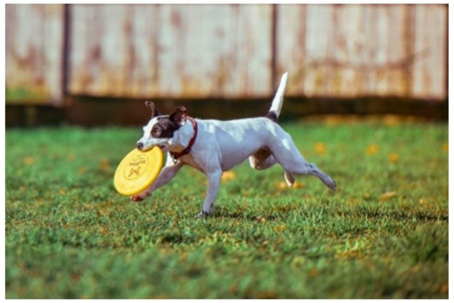
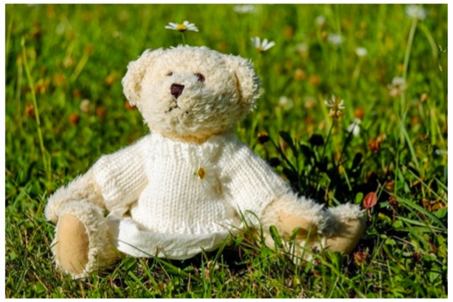


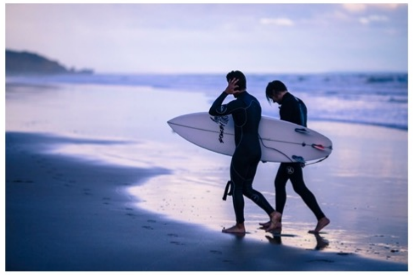
Image Captioning: Results

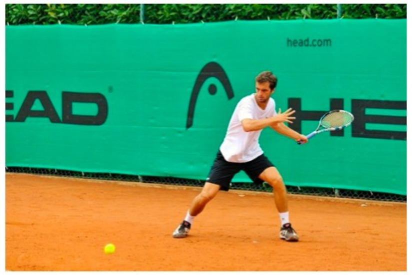

A cat sitting on a suitcase on the floor


A cat is sitting on a tree branch

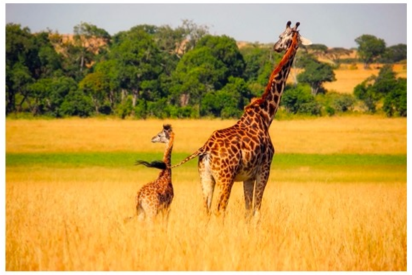

A dog is running in the grass with a frisbee

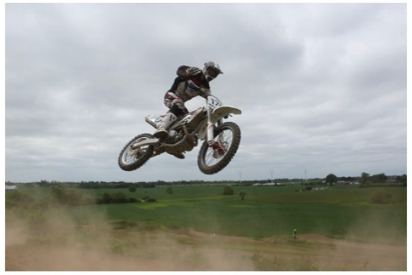

A white teddy bear sitting in the grass


Two people walking on the beach with surfboards



A tennis player in action on the court

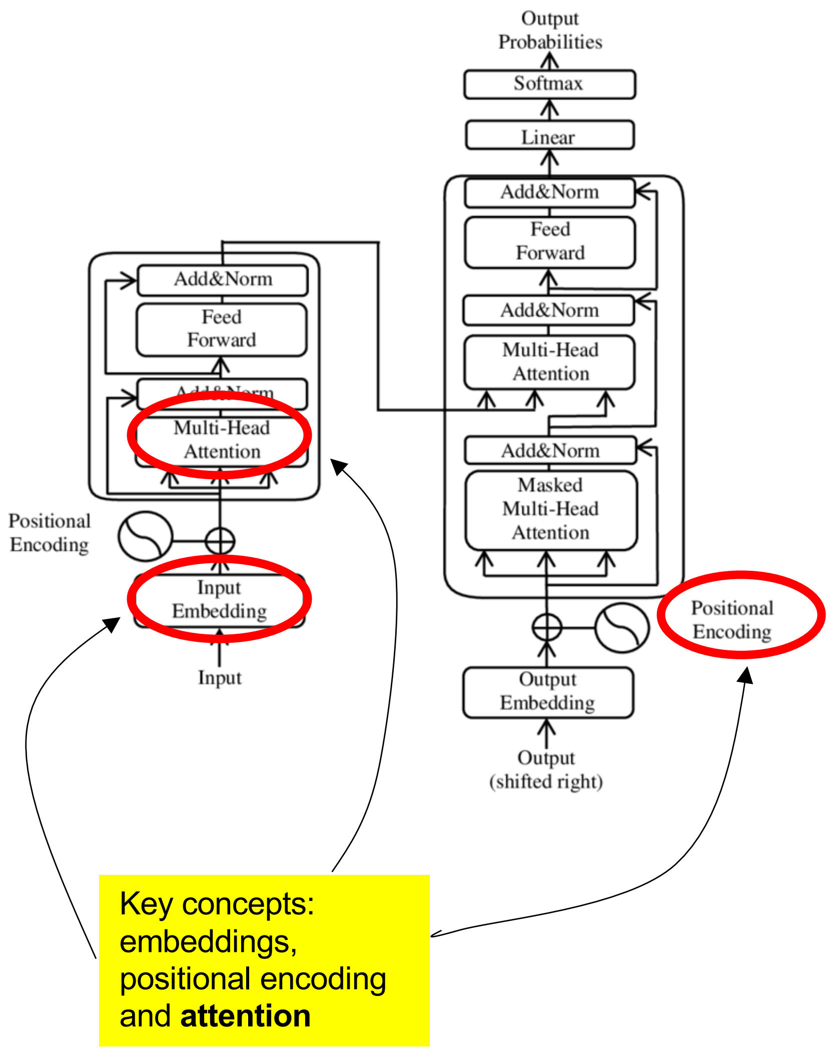

Two giraffes standing in a grassy field


A man riding a dirt bike on a dirt track

Credits: Fei-Fei Li, Yunzhu Li, Ruohan Gao

Attention and Transformers

- The Transformer architecture was proposed in the paper entitled "*Attention is All You Need*"
- As of March 2024, this paper had over 111,000 citations from peers
- It was responsible for expanding the 2014 attention mechanism (originally proposed by Bahdanau et. al.) into the Transformer architecture.
- The paper is **considered the founding document for modern artificial intelligence**, as transformers became the main architecture of large language (and vision) models (e.g., Chat GPT).



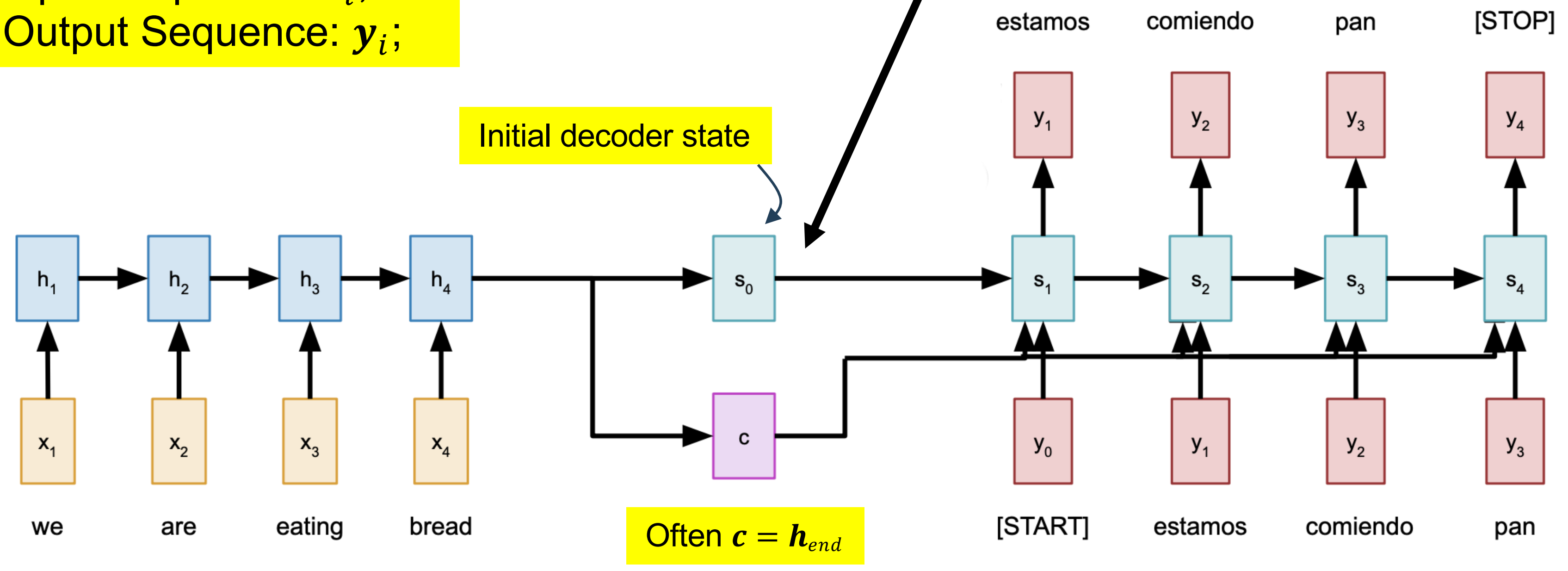
Key concepts: embeddings, positional encoding and **attention**

Attention and Transformers

- It was originally proposed for "Machine translation" purposes, i.e., sequence-to-sequence tasks.
- The focus was on improving Seq2seq techniques for machine translation, but even in their paper the authors saw the potential for other tasks like "*question answering*" and for what is now called multimodal Generative AI.

Seq2Seq Architecture
Input Sequence: $\boldsymbol{x}_i$;
Output Sequence: $\boldsymbol{y}_i$;

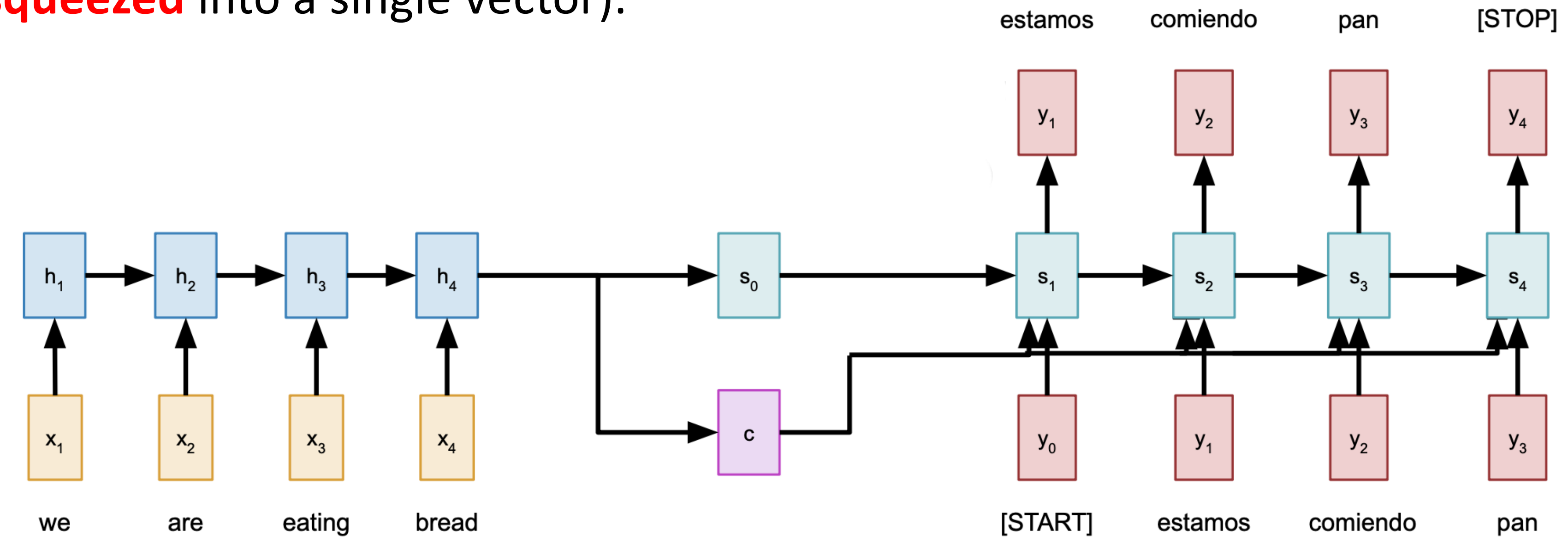
Main problem: Very large input sequences can be **bottlenecked** in the fixed-size state representation (Suppose T=100?)



Often $\boldsymbol{c} = \boldsymbol{h}_{end}$

Attention and Transformers

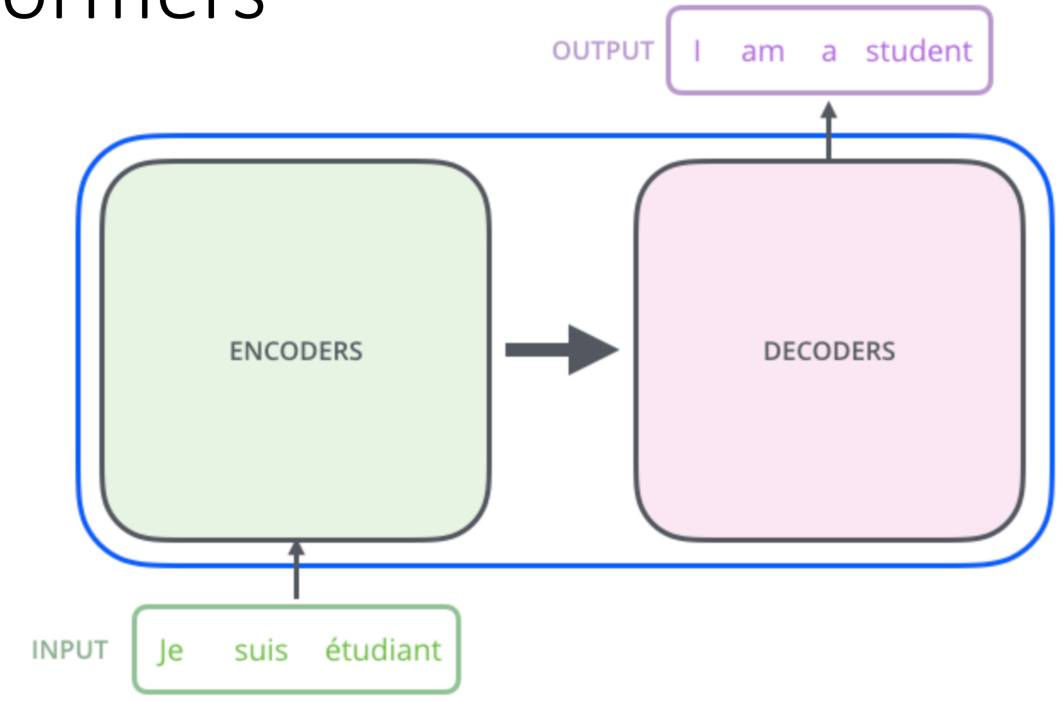
- Using this architecture, the encoder must encapsulate the entire input into a fixed-size vector that is passed to the decoder.
- With **Attention**, the complete input sentences aren't required to be encoded into a single vector. Instead, the decoder attends to different elements in the input sentence at each step of output generation.
- The previous generation of recurrent models had long paths between input and output words. For a 50-word sentence, the decoder had to recall information from 50 steps ago for the first word (and that data had to be **squeezed** into a single vector).



Attention and Transformers

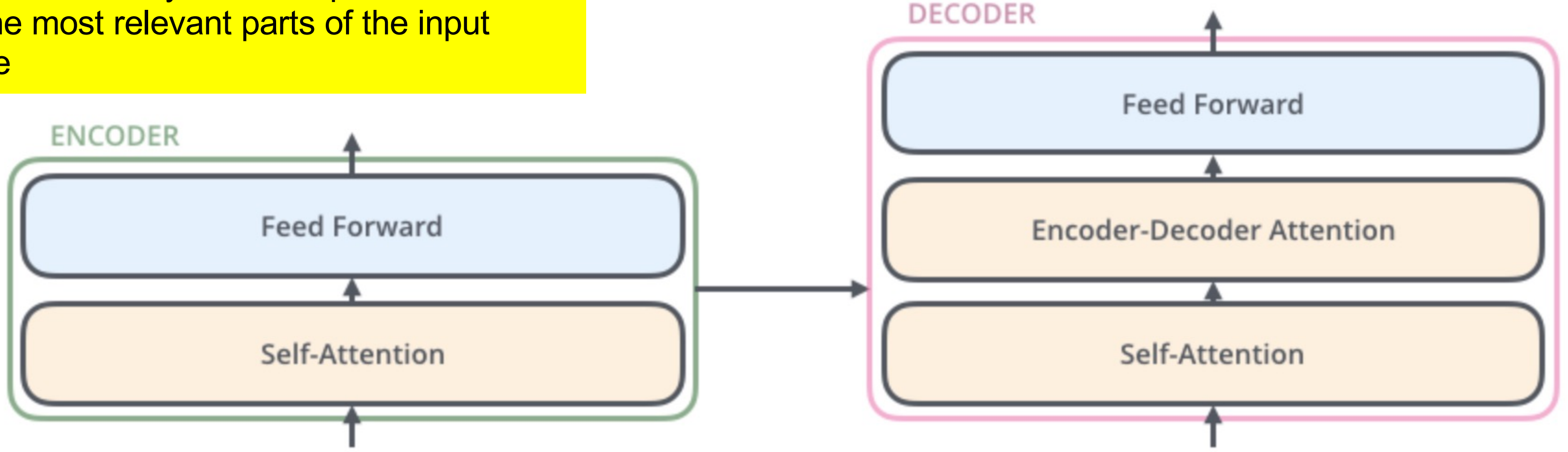
General Architecture:
Transformers share the encoder/decoder architecture, placing a stack of elements in each part of the pipeline (E/D).

The original implementation used a stack of 6 elements at each side.



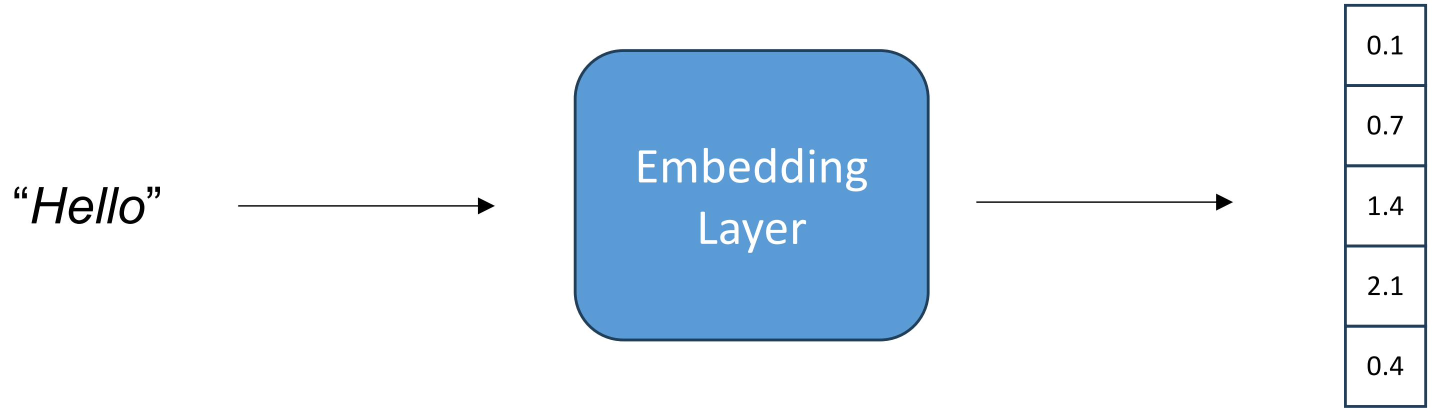
Each **Encoder** is divided into two parts: a **Self-Attention** layer followed by a **Feed-forward** (**Dense**) one.

The **Decoder** has a similar structure, but also uses an attention layer that helps the decoder to find the most relevant parts of the input sentence



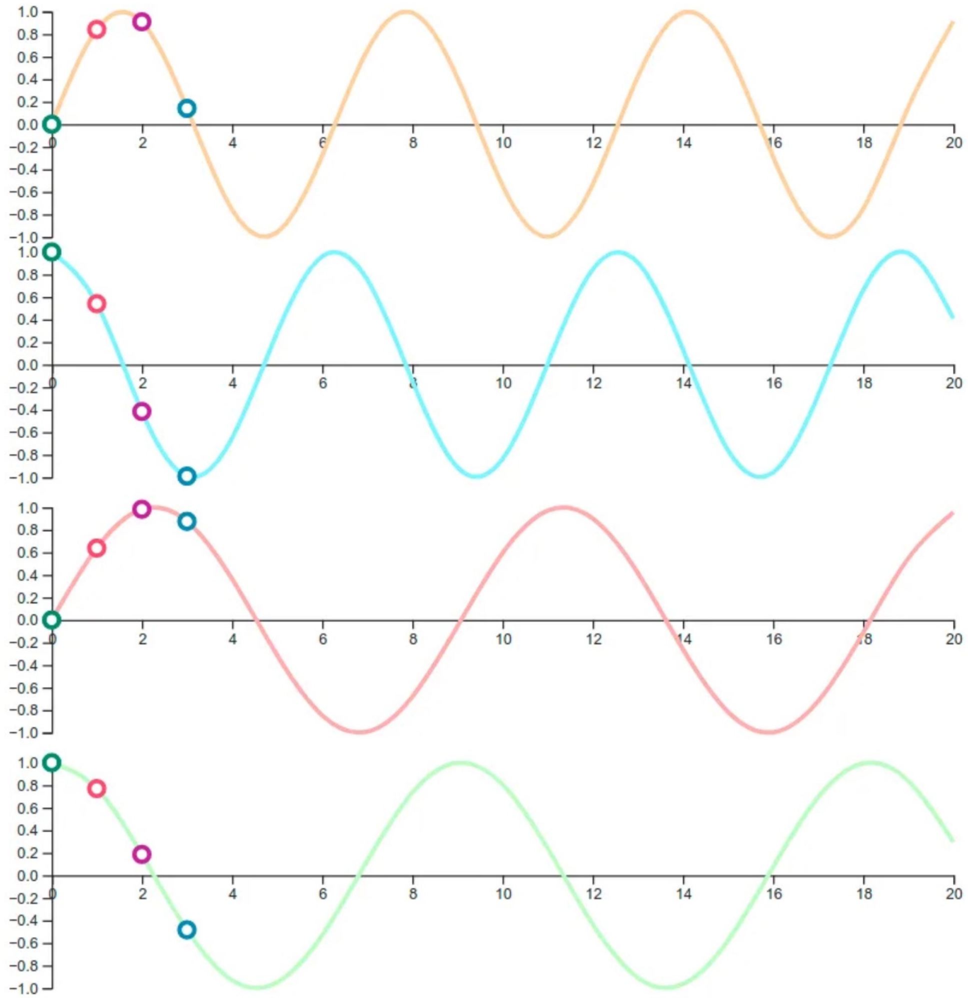
Attention and Transformers - Input Embedding

- The process starts (before feeding the input data to the first Encoder), by obtaining latent representations of the input elements.
- In practice, this first encoder begins by converting input tokens - words or subwords - into vectors using **Embedding layers**.
- These embeddings should capture the semantic meaning of the tokens and convert them into numerical vectors.
  - It is a more sophisticated variant of the "one-hot encoding" previously saw.



- As Transformers do not have a recurrence mechanism like RNNs, "**Positional encodings"** added to the input embeddings to provide information about the position of each token in the sequence. This allows them to understand the position of each word within the sentence.

Positional Encoding



p0	p1	p2	p3	
0.000	0.841	0.909	0.141	i=0
1.000	0.540	-0.416	-0.990	i=1
0.000	0.638	0.983	0.875	i=2
1.000	0.770	0.186	-0.484	i=3

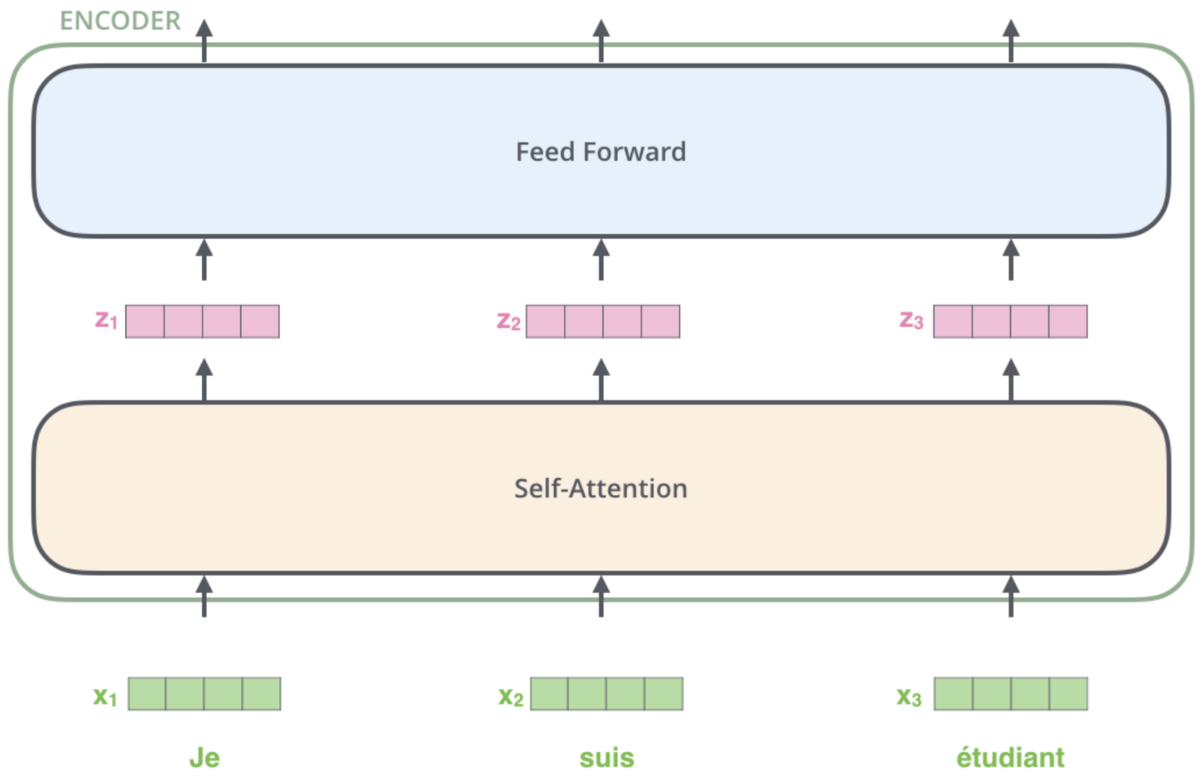
A set of sin() and cos() functions of **different frequencies** are used. This way, each input element is combined (added) to a vector that contains information about the position of the element within the sequence

$$\begin{array}{c} [0] \\ \begin{bmatrix} 0.1 \\ 0.7 \\ 1.4 \\ 2.1 \\ 0.4 \end{bmatrix} \end{array} + \begin{array}{c} P[0] \\ \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ \ldots \end{bmatrix} \end{array} = \begin{bmatrix} 0.1 \\ 1.7 \\ 1.4 \\ 3.1 \\ \ldots \end{bmatrix}$$

Embedding with positional context

Attention and Transformers

- Most encoders receive a list of input vectors $\boldsymbol{x}$, each of the size 512.
- After embedding the elements $\boldsymbol{x}_{i,}$ each of them flows through each of the two layers of the encoder.



A key property is that each input element $\boldsymbol{x}_i$ follows an independent path in the network. There are **dependencies** between these paths in the **self-attention layer**. The **feed-forward layer** does not have any dependencies.

Self Attention Mechanism - Encoder

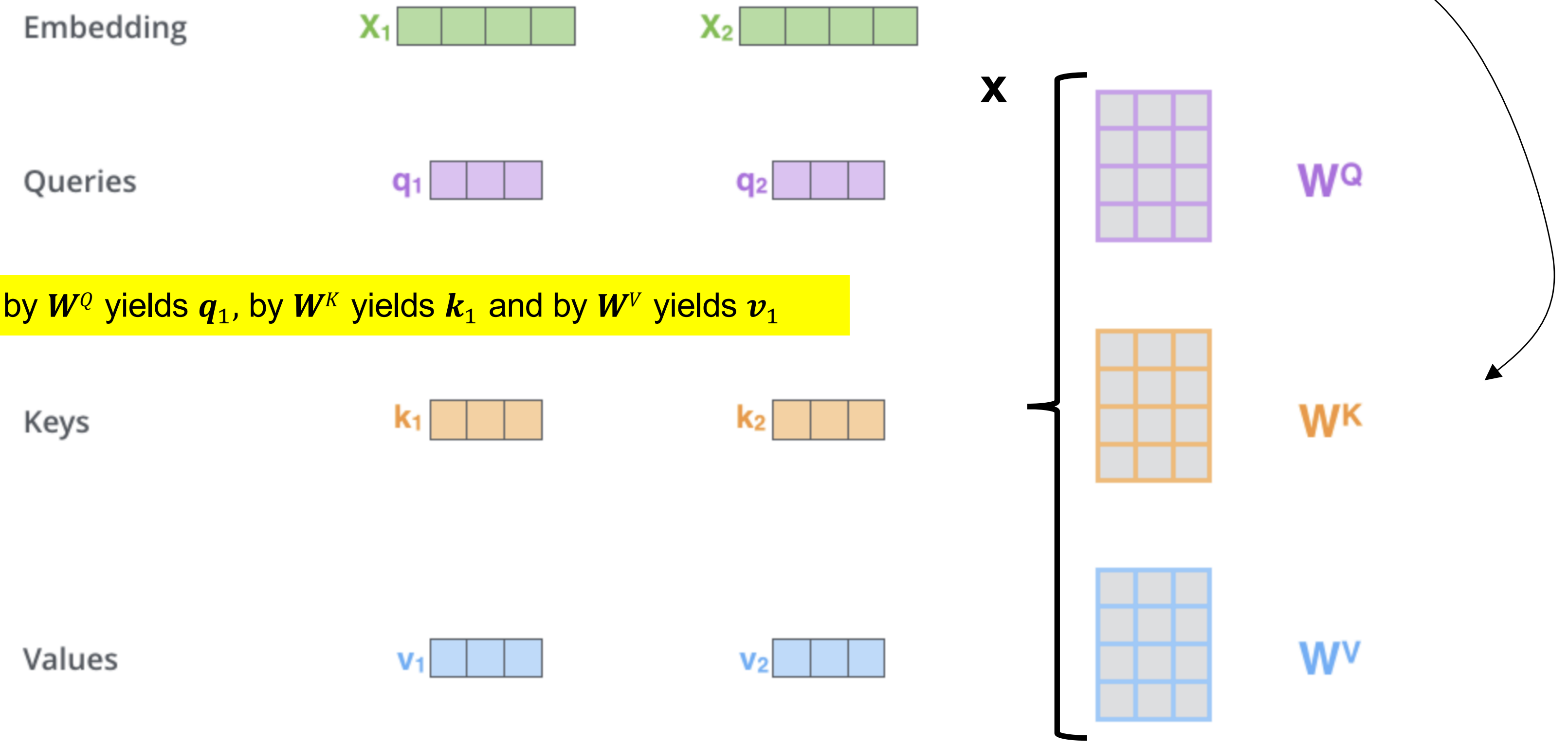
- **Attention** enables the models to relate each element in the input with other elements. For instance, in a given example, the model might learn to connect the element "$\boldsymbol{x}_i$" with "$\boldsymbol{x}_j$".
  - This allows the encoder to focus on different parts of the input sequence as it processes each token
  - It is based on 3 types of vectors: Queries ($\boldsymbol{q}_j$), Keys ($\boldsymbol{k}_j$) and Values ($\boldsymbol{v}_j$)

Intuition: *Imagine a library. We have a specific question (**query**). Books on the shelves have titles on their spines (**keys**) that suggest their content. We compare your query to these titles to decide how relevant each book is, and how much attention to give each book. Finally, we can get the information (**value**) from the relevant books to answer our question.*

- Attention is about how much *weight* the query word (e.g., $\boldsymbol{q}$) should give each word in the sentence (e.g., $\boldsymbol{k}_1$, $\boldsymbol{k}_2$, …). This is obtained via a dot product between the query and all the keys.
  - The dot product measures how similar two vectors are.
  - If the dot product between a query-key pair is high, we pay more attention to it.
  - These dot products then go through a *softmax* which makes the attention scores (across all keys) sum to 1

Self Attention Mechanism - Encoder

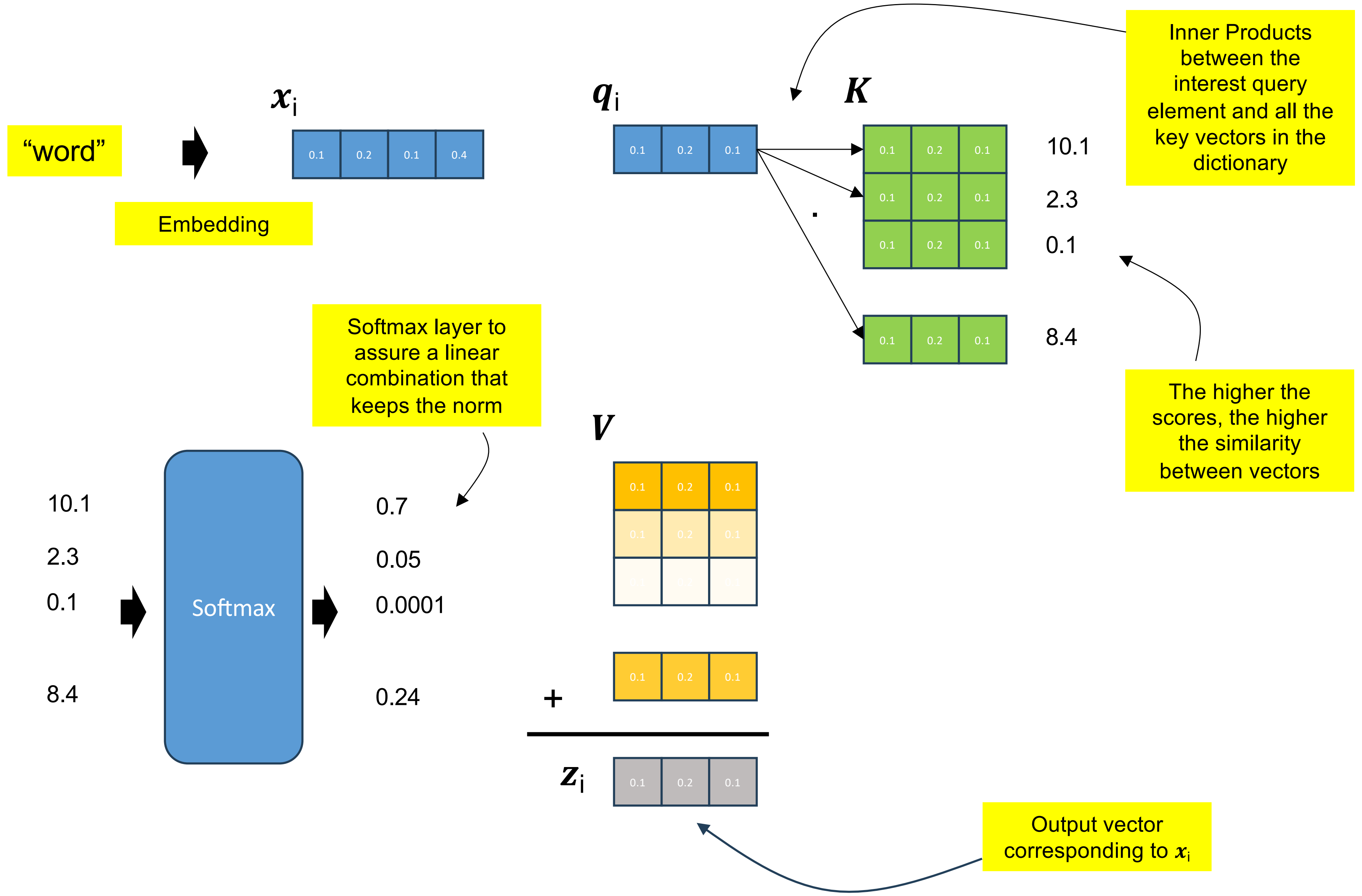
- We start by obtaining 3 vectors for each input element:
  - The **Query**, **Key** and **Value**. They are all created by multiplying the embedding by three matrices (the only ones trained during the learning process).



Multiplying $\boldsymbol{x}_1$ by $\boldsymbol{W}^Q$ yields $\boldsymbol{q}_1$, by $\boldsymbol{W}^K$ yields $\boldsymbol{k}_1$ and by $\boldsymbol{W}^V$ yields $\boldsymbol{v}_1$

Typically, the dimensionality of the query, key and value vectors is smaller than the dimensionality of the embedding (e.g., 64 << 512)

Self Attention Mechanism - Encoder

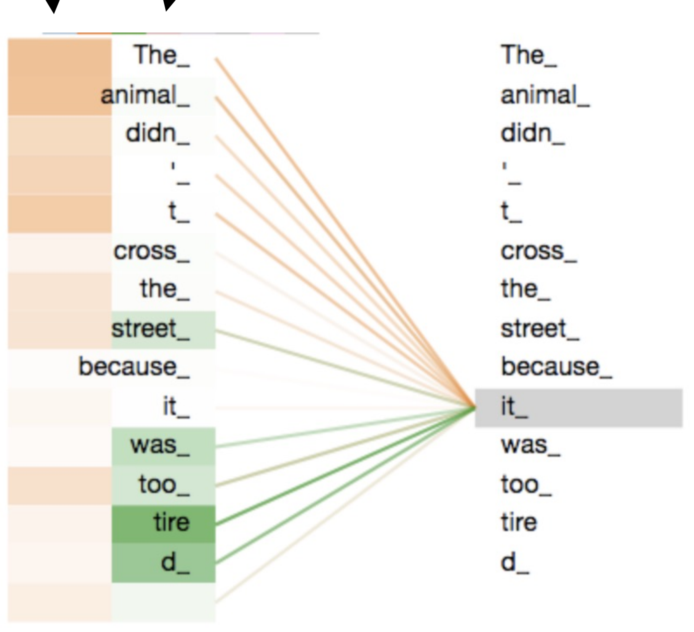


"word"
Embedding
$\boldsymbol{x}_i$
0.1 0.2 0.1 0.4
$\boldsymbol{q}_i$
0.1 0.2 0.1
$\boldsymbol{K}$
Inner Products between the interest query element and all the key vectors in the dictionary
10.1
2.3
0.1
8.4
The higher the scores, the higher the similarity between vectors
Softmax layer to assure a linear combination that keeps the norm
Softmax
0.7
0.05
0.0001
0.24
$\boldsymbol{V}$
+
$\boldsymbol{z}_i$
Output vector corresponding to $\boldsymbol{x}_i$

Self Attention Mechanism - Encoder

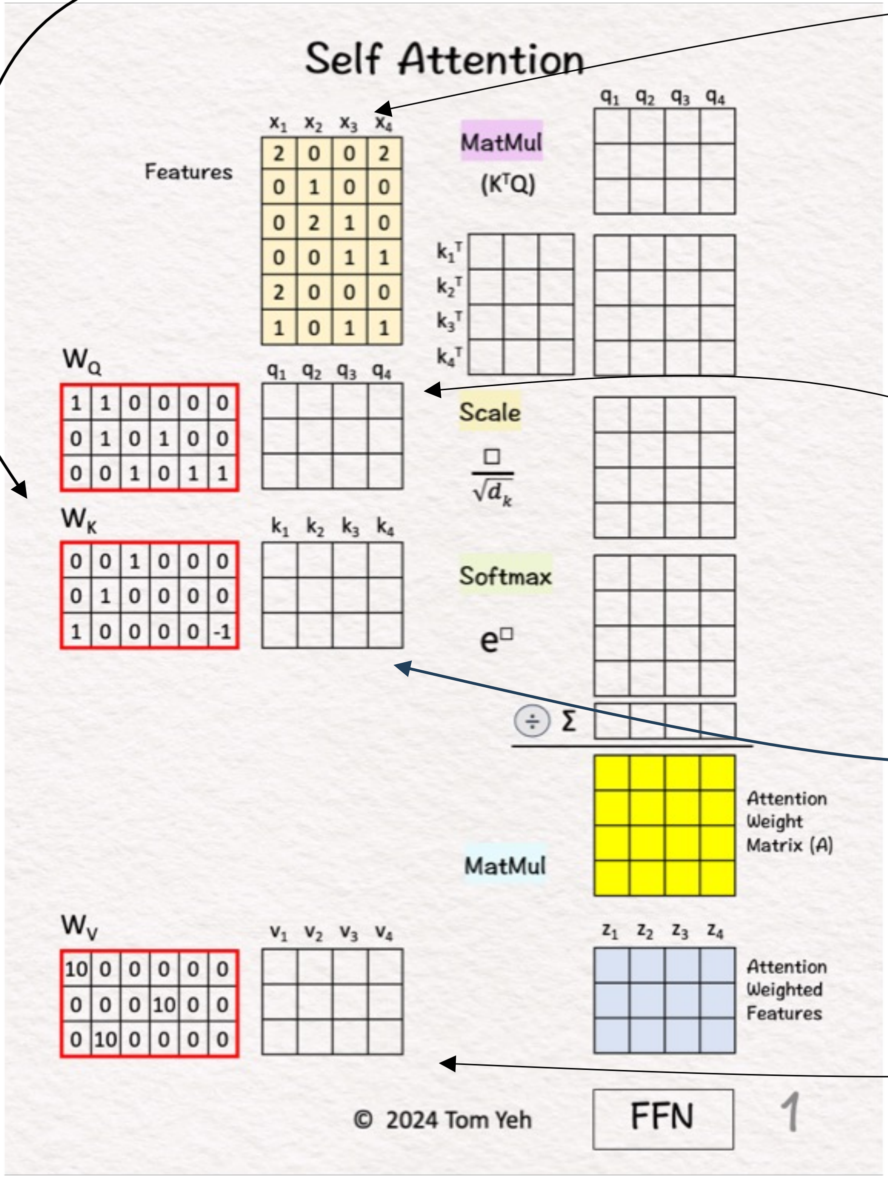
- Next, the inner product between the query $\boldsymbol{q}_i$, and all the key elements ($\boldsymbol{k}_1$,… $\boldsymbol{k}_n$) measures the similarity of the query with respect to every other element ($\boldsymbol{q}_i . \boldsymbol{k}_j$)
  - Normalizing and applying a *softmax* for all products gives us *how much* of the corresponding value vector should be used in the final sum to obtain the output vector $\boldsymbol{z}_i$.
  - Formally, this step yields the parameters of a linear combination between all the vectors, that will be used to represent the input $\boldsymbol{x}_i$.
- The resulting vector is sent to the feed-forward layer.
- The output of the final encoder layer is a set of vectors, each representing the input sequence with a rich contextual understanding. This output is then used as the input for the decoder in a Transformer model.

Attention that the element "it" gives to the remaining elements, for two two different "heads".



❤️ This is the "heart" of the Attention mechanism. To **replace** a representation of the input element by a **linear combination** of the other elements in the space, depending on the similarity/importance of each one with respect to the input. ❤️

Self Attention Mechanism – Matrix Form Example



Features

	x_1	x_2	x_3	x_4
	2	0	0	2
	0	1	0	0
	0	2	1	0
	0	0	1	1
	2	0	0	0
	1	0	1	1

W_Q

1	1	0	0	0	0
0	1	0	1	0	0
0	0	1	0	1	1

W_K

0	0	1	0	0	0
0	1	0	0	0	0
1	0	0	0	0	-1

W_V

10	0	0	0	0	0
0	0	0	10	0	0
0	10	0	0	0	0

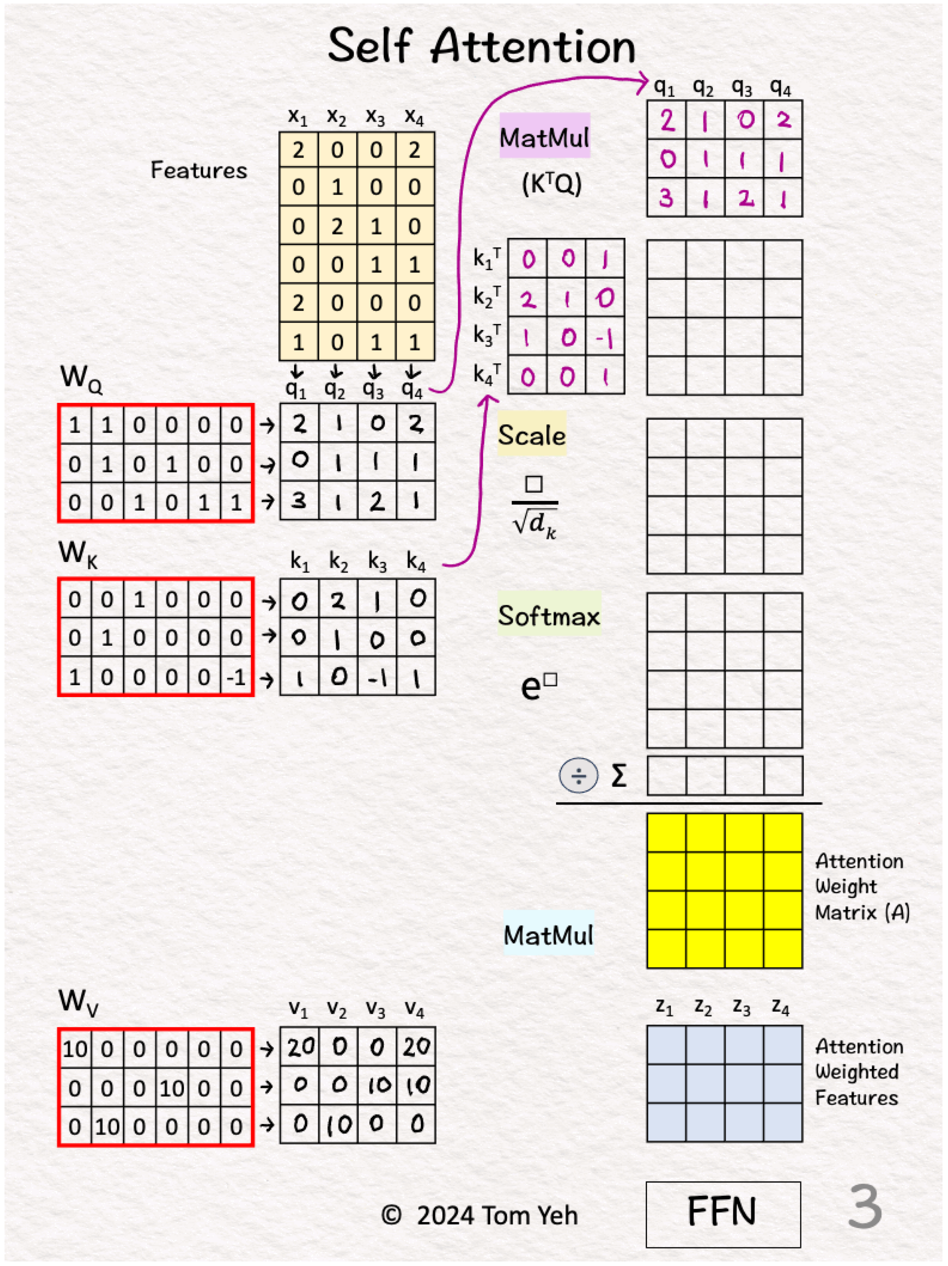
Input: Four 6D vectors x_i

(Suppose that at the current iteration, the Query, Quey and Value matrices have these values)

Step 1. Obtain the query, key and value representations (by multiplying the input vectors by the corresponding matrices)

Credits: https://www.linkedin.com/posts/tom-yeh_deeplearning-transformer-neuralnetwork-activity-7153234215709360129-9J8y/

Self Attention Mechanism – Matrix Form Example



Features:

x_1	x_2	x_3	x_4
2	0	0	2
0	1	0	0
0	2	1	0
0	0	1	1
2	0	0	0
1	0	1	1

W_Q:

1	1	0	0	0	0
0	1	0	1	0	0
0	0	1	0	1	1

q_1	q_2	q_3	q_4
2	1	0	2
0	1	1	1
3	1	2	1

W_K:

0	0	1	0	0	0
0	1	0	0	0	0
1	0	0	0	0	-1

k_1	k_2	k_3	k_4
0	2	1	0
0	1	0	0
1	0	-1	1

k_1^T	0	0	1
k_2^T	2	1	0
k_3^T	1	0	-1
k_4^T	0	0	1

q_1	q_2	q_3	q_4
2	1	0	2
0	1	1	1
3	1	2	1

W_V:

10	0	0	0	0	0
0	0	0	10	0	0
0	10	0	0	0	0

v_1	v_2	v_3	v_4
20	0	0	20
0	0	10	10
0	10	0	0

Step 2. Multiply K^T and Q
This is equivalent to taking the dot product between every pair of query and key vectors.

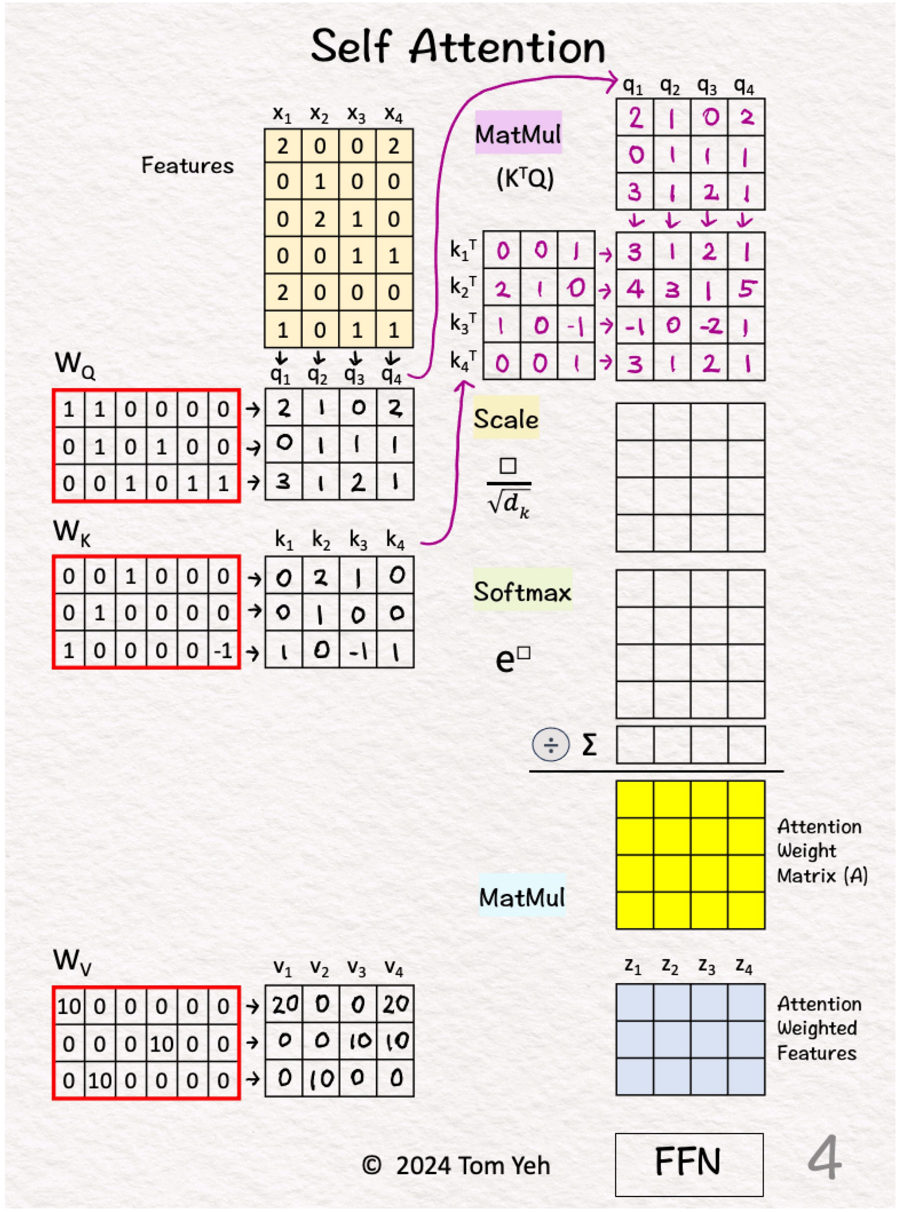
(4x3) x (3x4) = (4x4)

The idea is to use the dot product *as an estimate of the "matching score" between every key-value pair.*

This estimate makes sense because the dot product is the *numerator* of **cosine similarity** between two vectors.

Credits: https://www.linkedin.com/posts/tom-yeh_deeplearning-transformer-neuralnetwork-activity-7153234215709360129-9J8y/

Self Attention Mechanism – Matrix Form Example



$$\text{Features: }\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ 2 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{array}$$

$$W_Q = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \rightarrow \begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \\ 2 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 3 & 1 & 2 & 1 \end{array}$$

$$W_K = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \rightarrow \begin{array}{cccc} k_1 & k_2 & k_3 & k_4 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 \end{array}$$

$$W_V = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \\ 20 & 0 & 0 & 20 \\ 0 & 0 & 10 & 10 \\ 0 & 10 & 0 & 0 \end{array}$$

$$\begin{array}{c|ccc} k_1^T & 0 & 0 & 1 \\ k_2^T & 2 & 1 & 0 \\ k_3^T & 1 & 0 & -1 \\ k_4^T & 0 & 0 & 1 \end{array} \rightarrow \begin{array}{cccc} 3 & 1 & 2 & 1 \\ 4 & 3 & 1 & 5 \\ -1 & 0 & -2 & 1 \\ 3 & 1 & 2 & 1 \end{array}$$

$$\frac{\square}{\sqrt{d_k}} \qquad e^{\square} \qquad \div \Sigma$$

$z_1 \quad z_2 \quad z_3 \quad z_4$

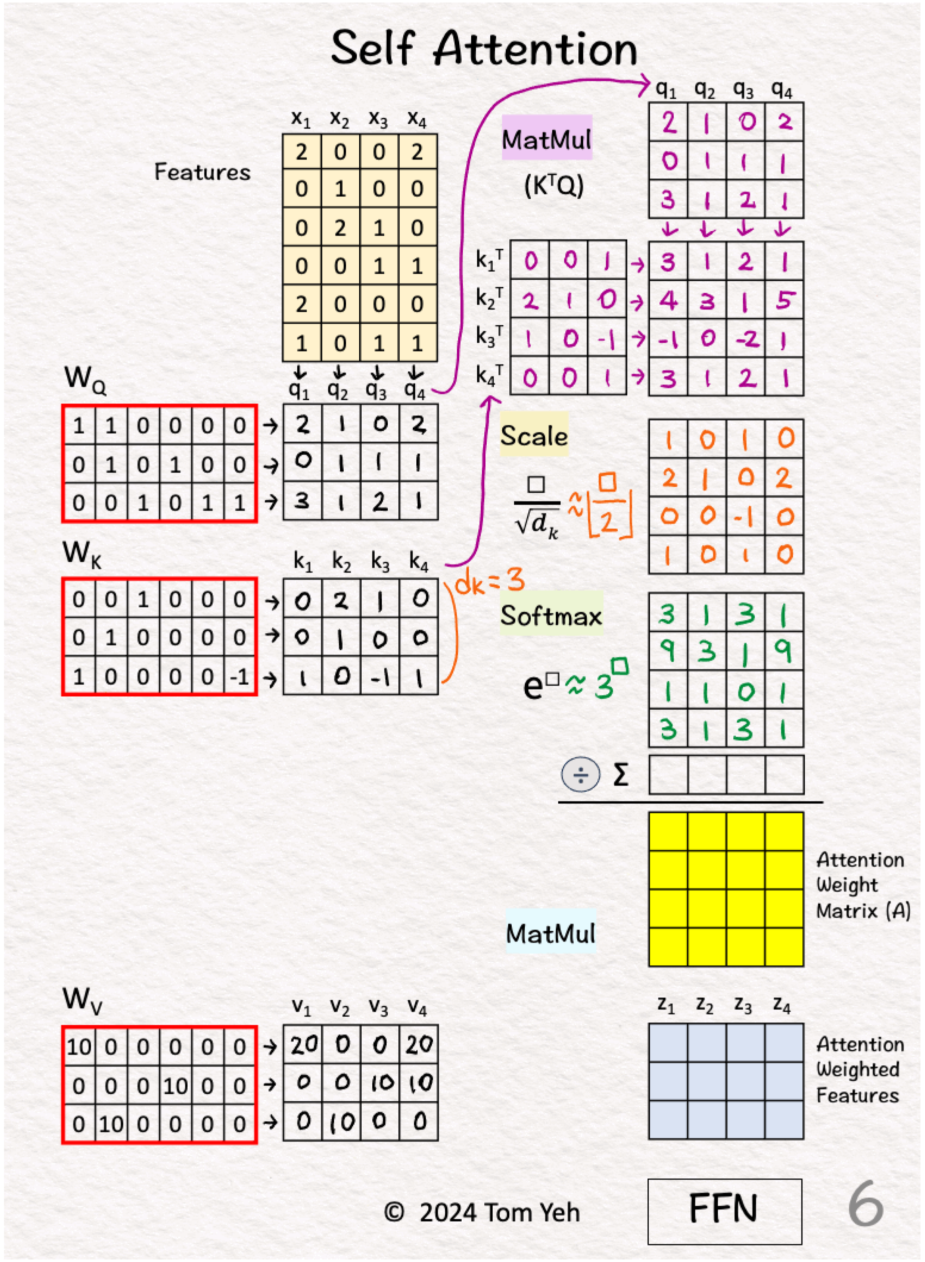
Step 3. Scale each element by the square root of dk, which is the dimension of key vectors (dk=3).

The purpose is to normalize the impact of the dk on matching scores, even if we scale dk to 32, 64, or 128.

To simplify hand calculation, we approximate [□/sqrt(3)] with [floor(□/2)].

Credits: https://www.linkedin.com/posts/tom-yeh_deeplearning-transformer-neuralnetwork-activity-7153234215709360129-9J8y/

Self Attention Mechanism – Matrix Form Example


Self Attention
Features
MatMul
(K^TQ)
Scale
$\frac{\square}{\sqrt{d_k}} \approx \frac{\square}{2}$
$d_k = 3$
Softmax
$e^{\square} \approx 3^{\square}$
÷ Σ
Attention Weight Matrix (A)
MatMul
Attention Weighted Features
FFN
6
© 2024 Tom Yeh

$$\begin{array}{c|cccc} & x_1 & x_2 & x_3 & x_4 \\ \hline & 2 & 0 & 0 & 2 \\ & 0 & 1 & 0 & 0 \\ & 0 & 2 & 1 & 0 \\ & 0 & 0 & 1 & 1 \\ & 2 & 0 & 0 & 0 \\ & 1 & 0 & 1 & 1 \end{array}$$

$$W_Q = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \rightarrow \begin{array}{cccc} q_1 & q_2 & q_3 & q_4 \\ 2 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 3 & 1 & 2 & 1 \end{array}$$

$$W_K = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \rightarrow \begin{array}{cccc} k_1 & k_2 & k_3 & k_4 \\ 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 \end{array}$$

$$W_V = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \\ 20 & 0 & 0 & 20 \\ 0 & 0 & 10 & 10 \\ 0 & 10 & 0 & 0 \end{array}$$

$$\begin{array}{c|ccc} k_1^T & 0 & 0 & 1 \\ k_2^T & 2 & 1 & 0 \\ k_3^T & 1 & 0 & -1 \\ k_4^T & 0 & 0 & 1 \end{array} \rightarrow \begin{array}{cccc} 3 & 1 & 2 & 1 \\ 4 & 3 & 1 & 5 \\ -1 & 0 & -2 & 1 \\ 3 & 1 & 2 & 1 \end{array}$$

$$\text{Scale: } \begin{array}{cccc} 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 2 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 1 & 0 \end{array} \qquad \text{Softmax: } \begin{array}{cccc} 3 & 1 & 3 & 1 \\ 9 & 3 & 1 & 9 \\ 1 & 1 & 0 & 1 \\ 3 & 1 & 3 & 1 \end{array}$$

$$\begin{array}{cccc} z_1 & z_2 & z_3 & z_4 \end{array}$$

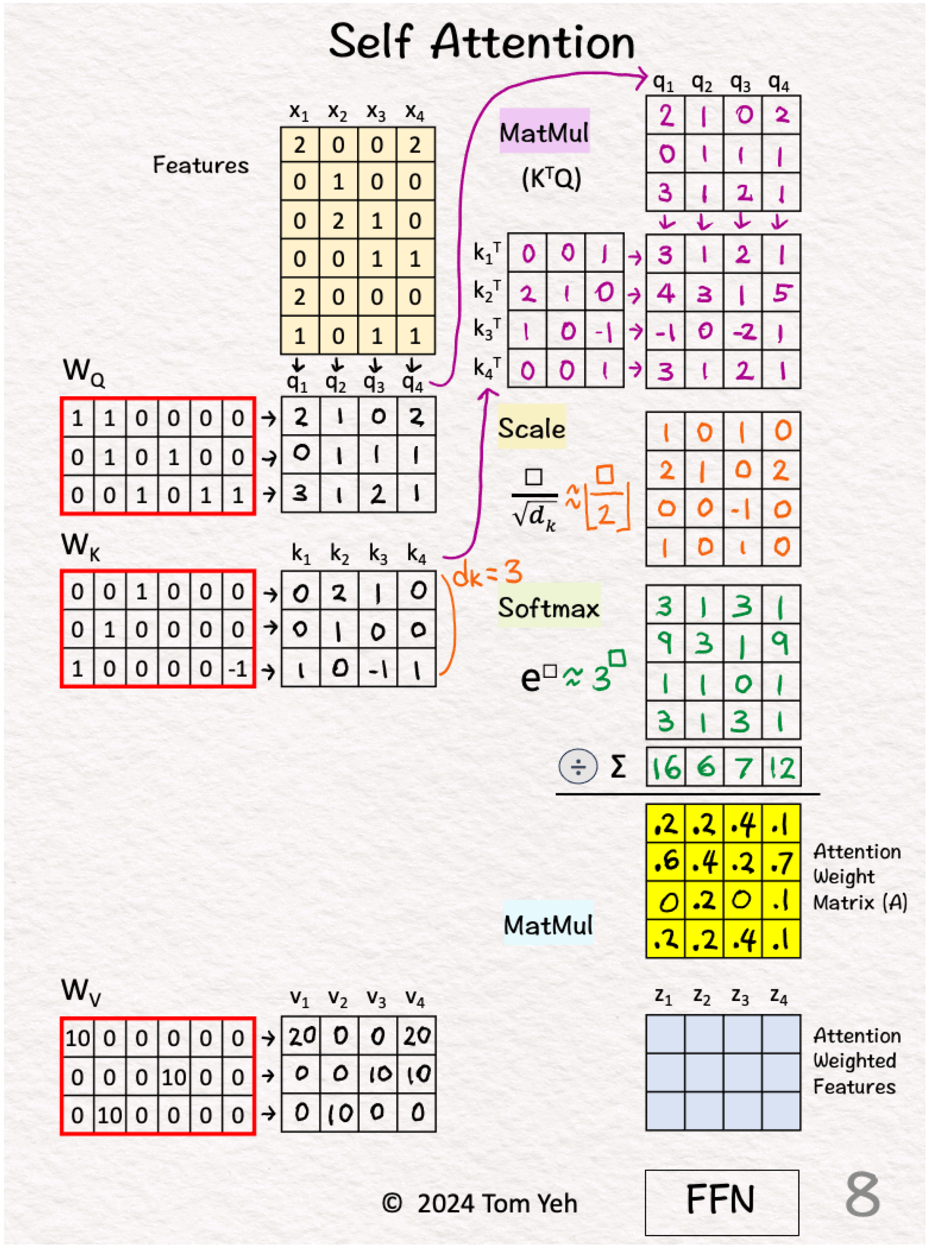
Step 4. Softmax: e^x

- Raise e to the power of the number in each cell

- To simplify hand calculation, we approximate e^□ with 3^□.

Credits: https://www.linkedin.com/posts/tom-yeh_deeplearning-transformer-neuralnetwork-activity-7153234215709360129-9J8y/

Self Attention Mechanism – Matrix Form Example



Step 5. Softmax: e^x

- Raise e to the power of the number in each cell

- To simplify hand calculation, we approximate e^□ with 3^□.

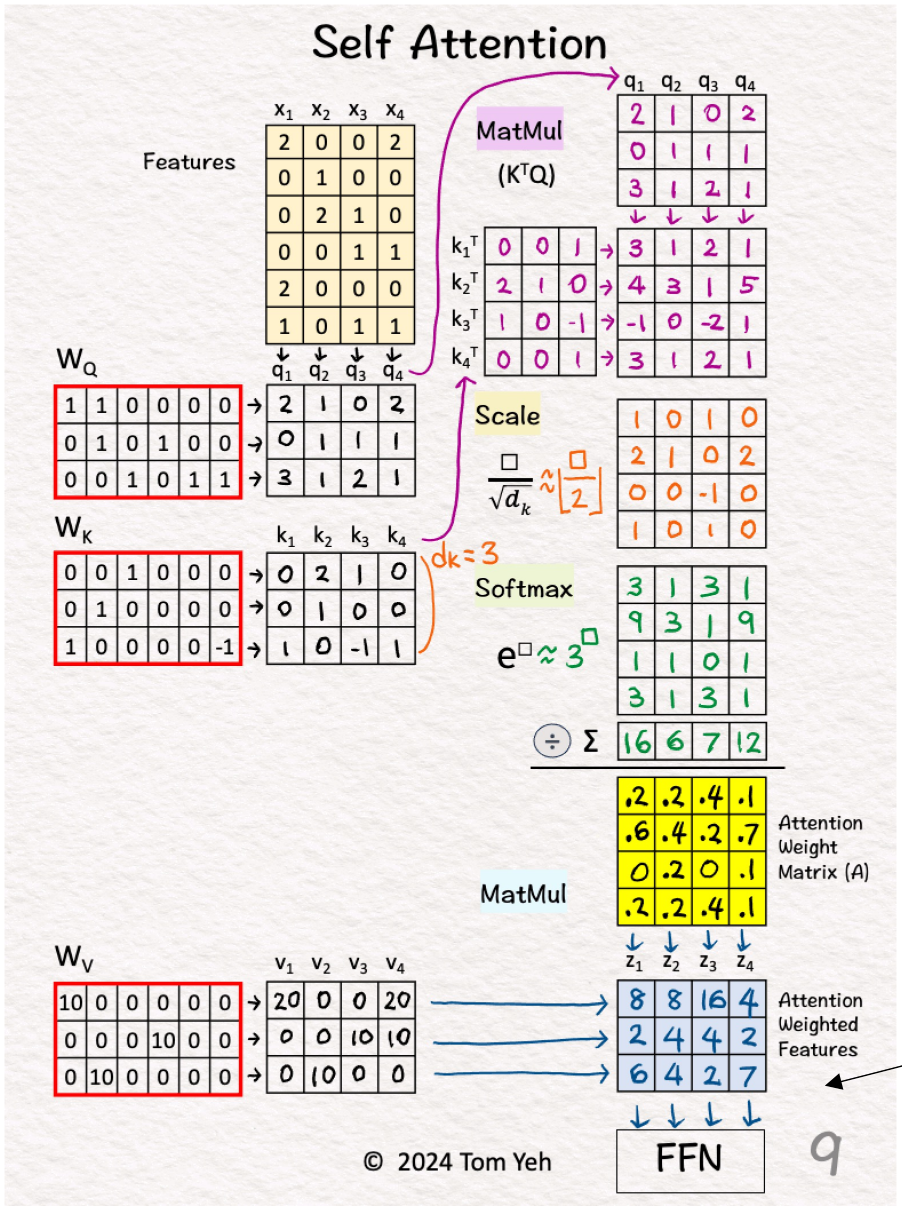
For each column, divide each element by the column sum

The purpose is normalize each column so that the numbers sum to 1. In other words, each column is a probability distribution of attention, and we have four of them.

The result is the Attention Weight Matrix (A) (yellow)

Credits: https://www.linkedin.com/posts/tom-yeh_deeplearning-transformer-neuralnetwork-activity-7153234215709360129-9J8y/

Self Attention Mechanism – Matrix Form Example



Step 7. MatMul
Multiply the value vectors (Vs) with the Attention Weight Matrix (A)

The results are the attention weighted features Zs.

They are fed to the position-wise feed forward network in the next layer.

In:

2	0	0	0	2	1

Out:

8	2	6

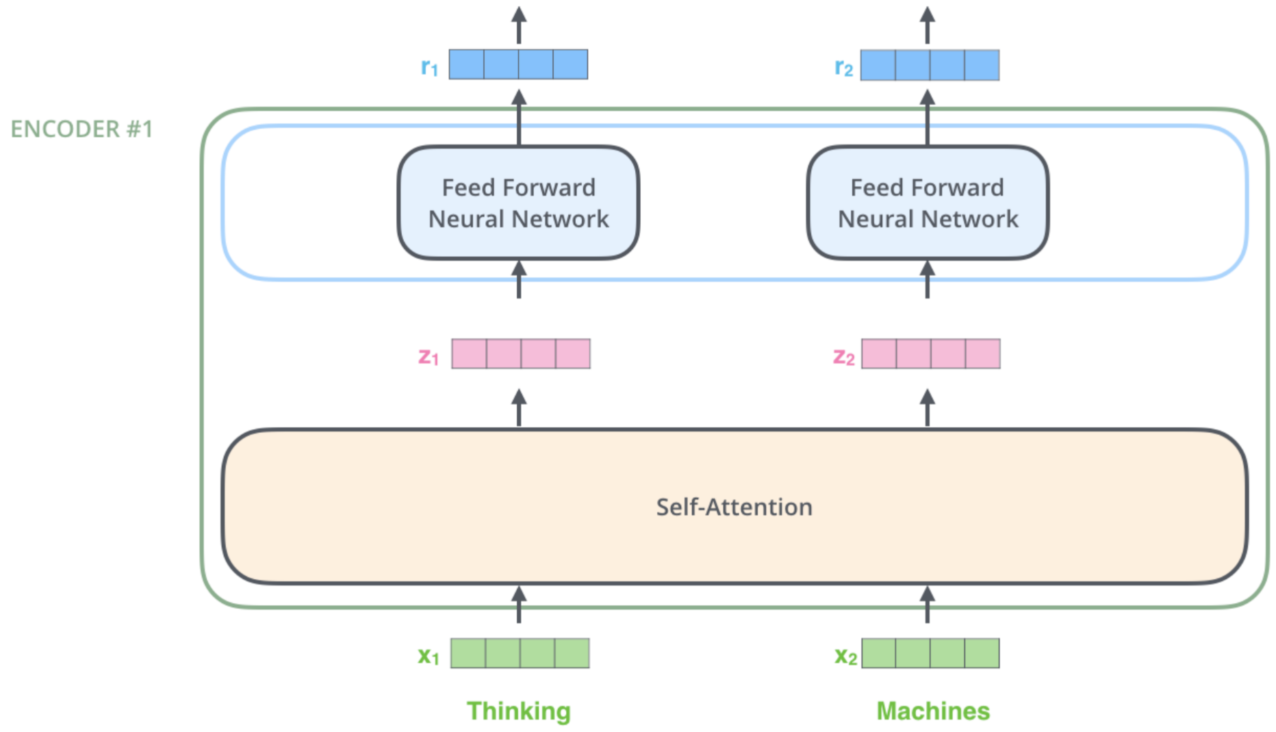
In practice, the first value vector (v_1) pays attention (is replaced by…) to 0.2 v_1 + 0.6 v_2 + 0.2 v_3

This yields one row of the $\boldsymbol{V}_{\text{encdec}}$ matrix (next...)

Credits: https://www.linkedin.com/posts/tom-yeh_deeplearning-transformer-neuralnetwork-activity-7153234215709360129-9J8y/

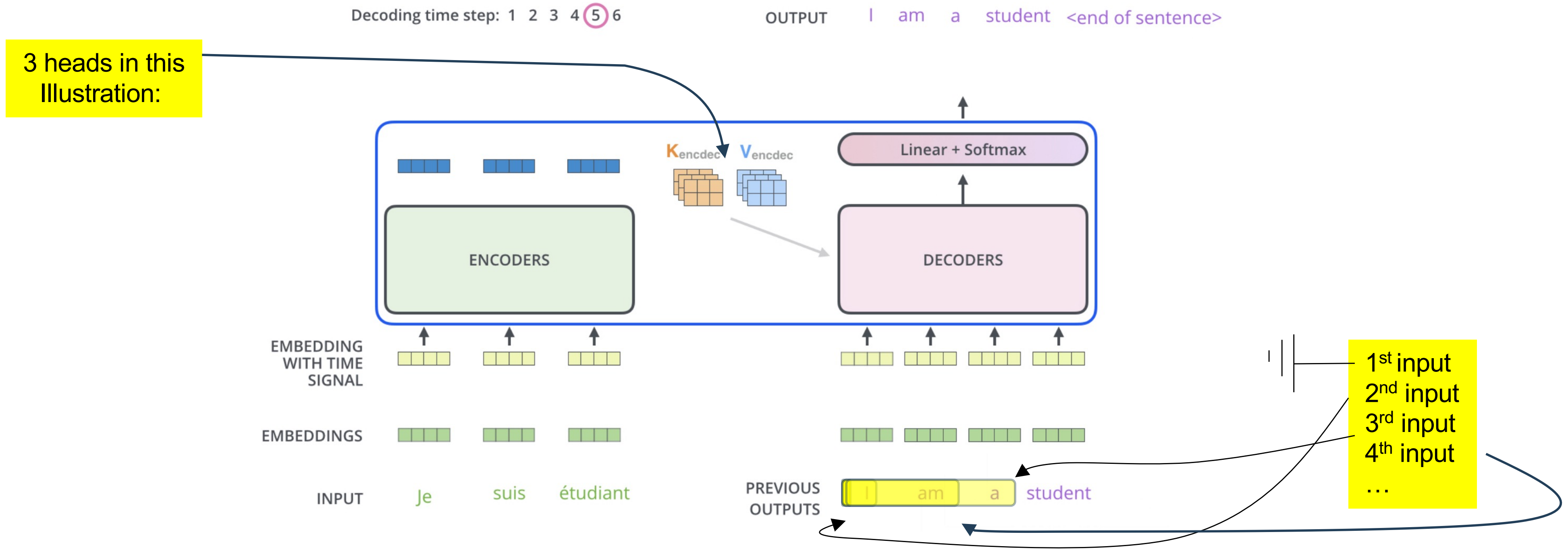
Self Attention Mechanism - Encoder

- The described process allows to map the embedding representations of each input v_i element into the attention vectors z_i.
- The final encoding encoding step consists of passing the z_i elements through a feed forward (dense) layer.



Self Attention Mechanism - Decoder

- Using multiple heads, the Value matrices representations (obtained as previously Illustrated) are concatenated into ($\boldsymbol{K}_{\text{encdec}}$ and $\boldsymbol{V}_{\text{encdec}}$) (with as many elements as the number of heads used).
  - They represent the features of the whole input sequence.
  - Are used in the second multi-head attention module of the decoder to relate the input sequence to the masked output of the first multi-head decoder.
- Then, the decoder starts to produce its outputs, until a special element (<END>) indicates that the process must be stopped.
  - During the first iteration, only the "<start>" token is additionally given
  - At each iteration, the set of previous outputs is also given as input.
  - The self attention layers are only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to "$-\infty$") before the softmax step in the self-attention calculation.



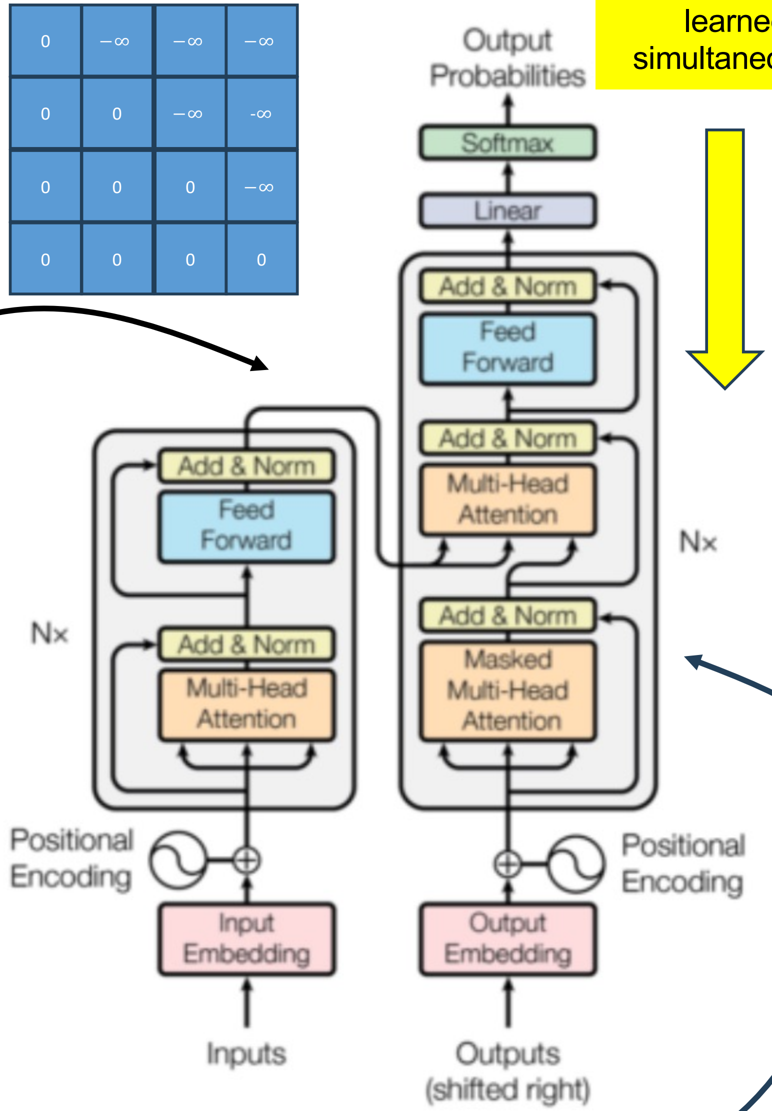
Self Attention Mechanism - Decoder

Gradients flow across the whole network. i.e., all matrices are learned simultaneously

- Note that the decoder has an extra level of complexity. The masked multi-head attention layer, that avoids to pay attention to "future words"

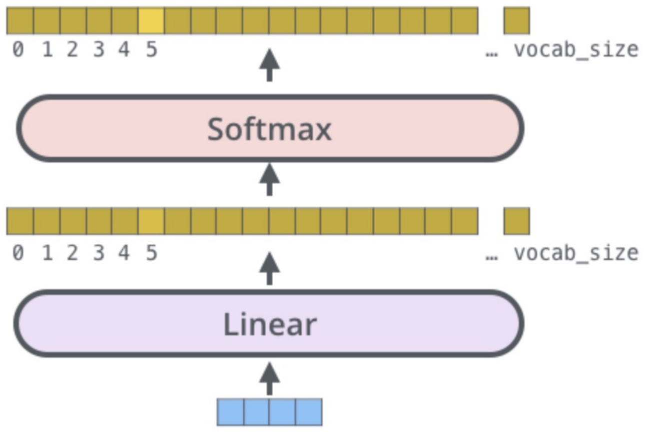
$$\begin{bmatrix} 12 & 3 & 5 & 2 \\ 4 & 9 & 3 & 5 \\ 2 & 3 & 7 & 2 \\ 3 & 4 & 1 & 9 \end{bmatrix} + \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- The output of this layer is then fed to the "Multi-head Attention" layer that uses the Key $\boldsymbol{K}_{\text{encdec}}$ and Value $\boldsymbol{V}_{\text{encdec}}$ outputs given by the encoder.



Self Attention Mechanism - Decoder

- The final part of the decoder works pretty much as a standard "classification" CNN, returning a vector with as many entries as the number of elements in the dictionary. After a "softmax()" layer, the index o the maximum element is found and the corresponding entry in the dictionary returned.



- Real-Life Well-Known Transformers:

  - Google's 2018 release of **BERT**, an open-source natural language processing framework, revolutionized NLP with its unique bidirectional training. Pre-trained on Wikipedia, excels in various NLP tasks, prompting Google to integrate it into its search engine for more natural queries.

  - **LaMDA** (Language Model for Dialogue Applications) is a Transformer-based model developed by Google, designed specifically for conversational tasks, and launched during in 2021. They are designed to generate more natural and contextually relevant responses, enhancing user interactions in various applications.

  - **ChatGPT**, developed by OpenAI, are advanced generative models known for their ability to produce coherent and contextually relevant text. They are suitable for content creation, conversation, language translation, ... GPT's architecture enables it to generate text that closely resembles human writing, making it useful in applications like creative writing, customer support, and even coding assistance.