CREATEQ

Generative AI

Cont.

Overview

CREATEQ

1. **Neural Networks** – Quick Revision
2. **NLP Architectures**
    - Word Embeddings
    - Neural Networks
    - RNNs
    - LSTM Decoder-Encoder
    - LSTM with Attention
    - Transformers
    - BERT and GPT
3. **OpenAI**
4. **DeepSeek**
5. **Gemini**
6. ~~Prompt Engineering~~ – Moved to the next session

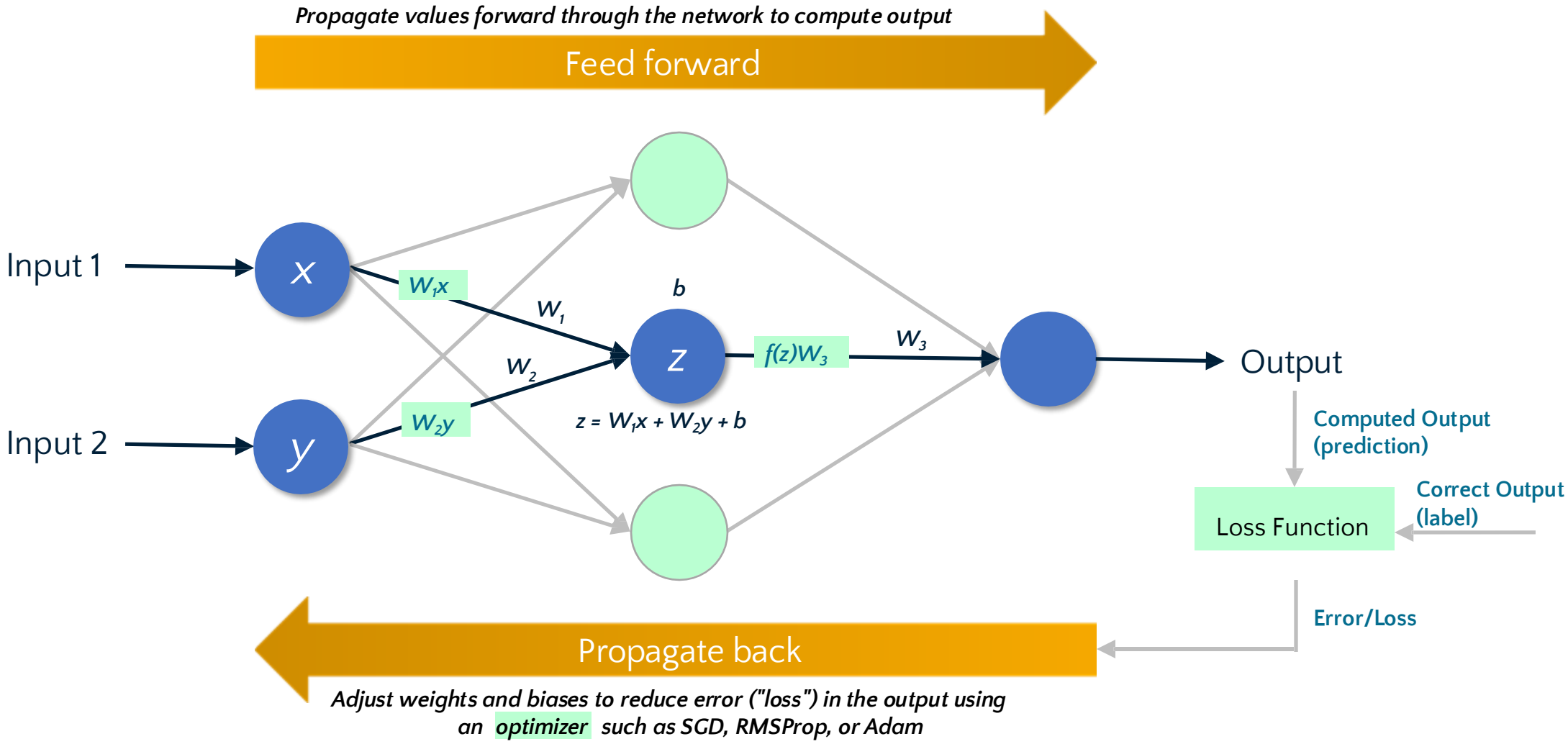
CREATEQ

Neural Networks

Quick Revision

Training Neural Networks

CREATEQ



Activation Functions

CREATEQ

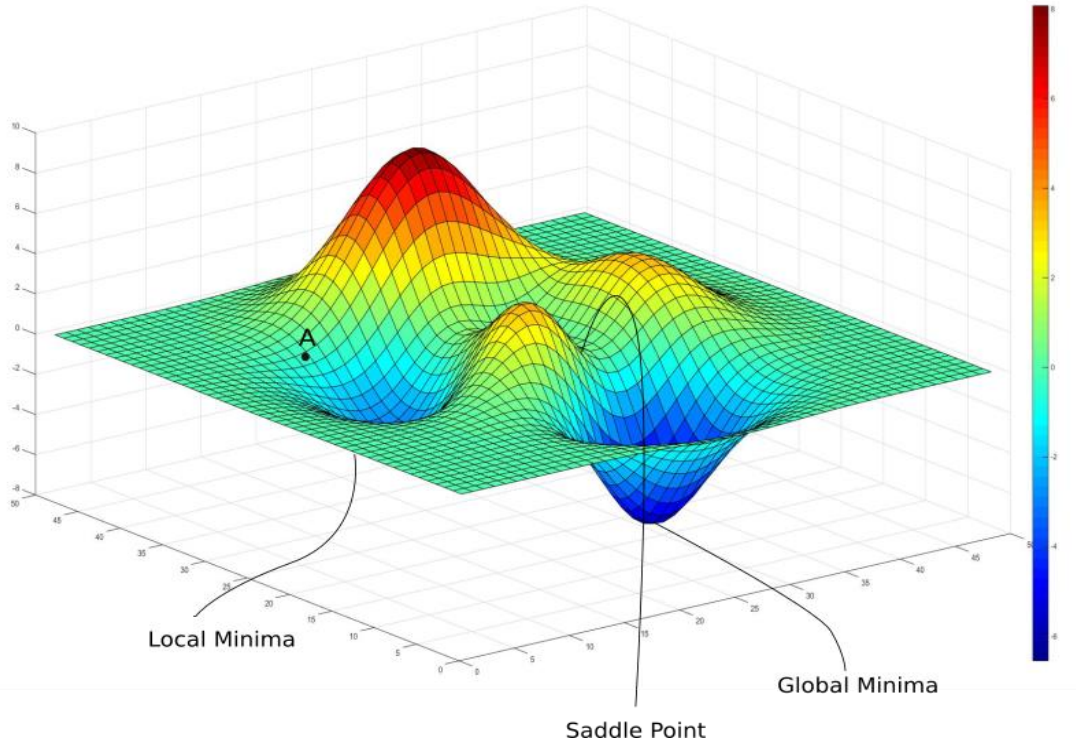
Activation Function	When to use
ReLU	The **most popular** activation for hidden layers. It is simple, fast to compute, and works well for most models. No vanishing gradient.
Sigmoid	For **binary classification**, when you need outputs in $(0,1)$ and logistic regression**.**
Softmax	Used in the output layer, especially for **multi-class classification**. It converts raw scores into **probabilities** for each class, ensuring that the output sums to 1.
Tanh	When you need outputs in $(-1,1)$, that are **zero-centered**
Parametric ReLU	Useful where ReLU's zero output for negative inputs could cause **dead neurons**.
Exponential ReLU	When you want a **smoother**, more **continuous** activation function that avoids **dead neurons** and is less prone to the **vanishing gradient problem**.

Concepts

CREATEQ

- **Loss (Cost) Functions**
  - It expresses the loss (error) by computing the discrepancy between the network's **predicted outputs** and the **actual target values** (ground truth).

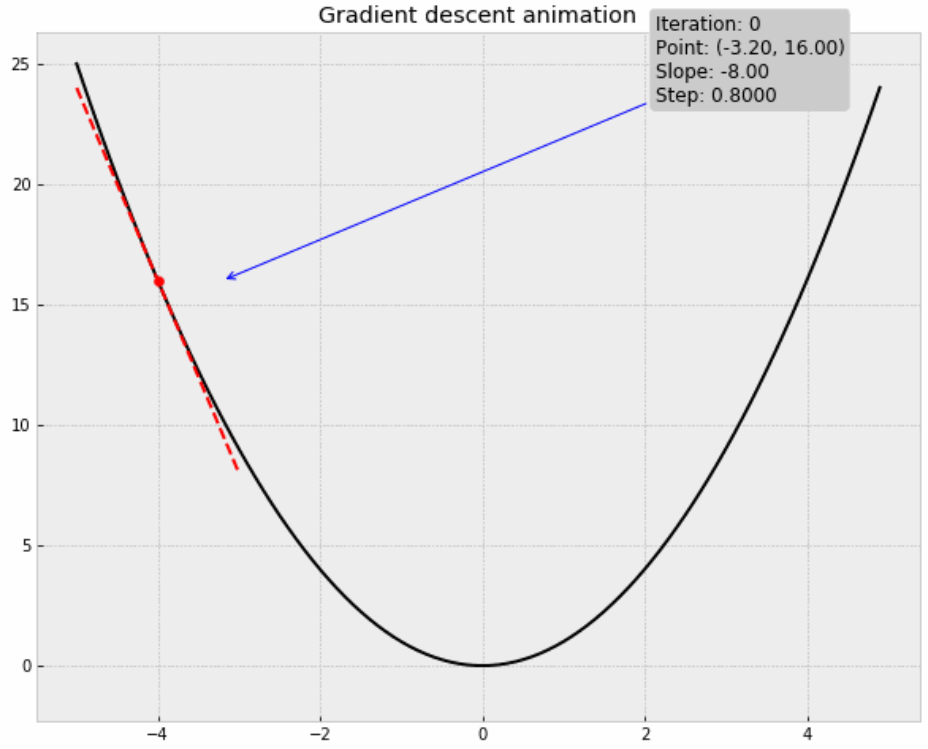
- **Mini-Batch Training**
  - Splits the dataset into **smaller chunks.**
  - Each whole chunk is used to adjust parameters (unlike classic training, in which updating is done after processing each element from the training dataset).



Concepts

CREATEQ

- **Gradient**
  - Another name for the **first derivative**
  - Gradient determines the **slope of tangent** of the graph for a function
  - In other words, it points to the direction of the **greatest rate of increase**
  - By changing the input of the function towards the **opposite direction**, we can reach a **global minimum**
  - **Vanishing Gradient Problem**: during backpropagation, the gradients (used to update weights) get smaller and smaller as they go backward through many layers. This makes the earlier layers learn very slowly or not at all.

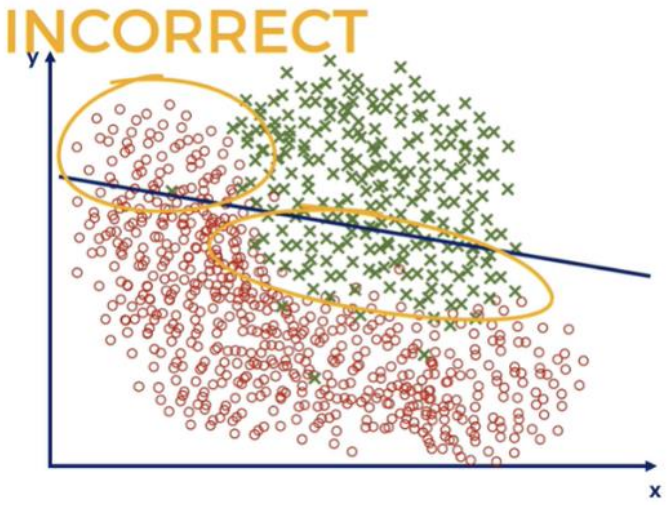


https://www.kaggle.com/code/trolukovich/animating-gradien-descent

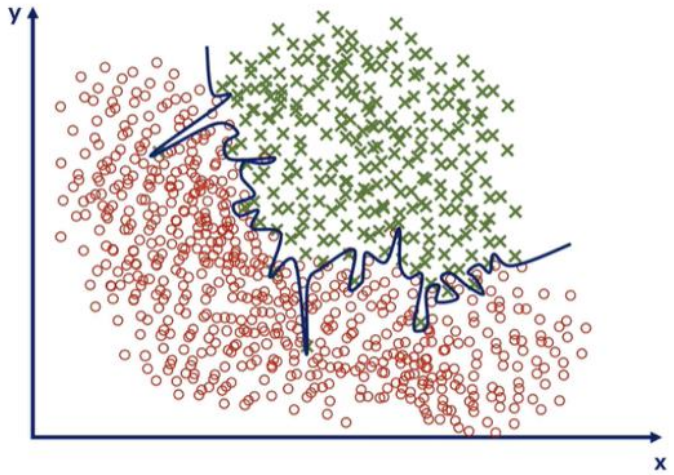
Concepts

CREATEQ

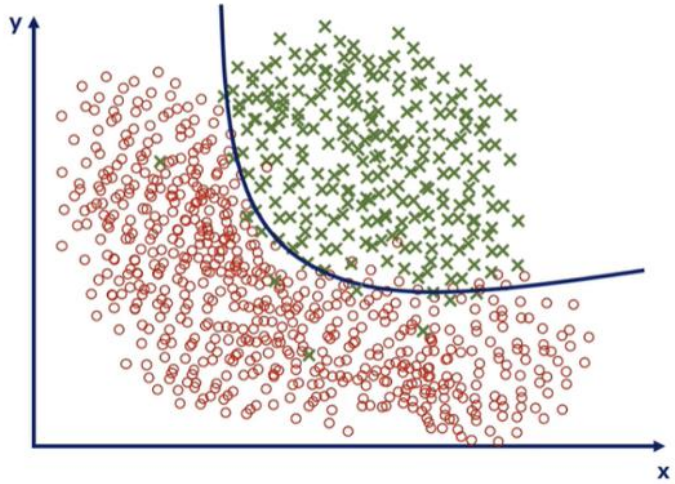
- **Overfitting**
  - Model performs well on training data, but is unable to predict on new, ”unseen” data.
  - Caused by datasets that are too small or not varied enough.



- Underfit model



- Overfit model



- Right fit

Neural Network Types

CREATEQ

Neural Network Type	Description	Uses
Feedforward Neural Networks (FNN)	Simplest neural network with data flowing in one direction.	- Classification and regression tasks. - Didactic
Recurrent Neural Networks (RNN)	Designed for sequential data, with loops allowing information to persist.	- Natural language processing (e.g., text generation). - Time-series analysis. - Speech recognition and translation.
Long Short-Term Memory Networks (LSTMs)	A type of RNN excelling at capturing long-term dependencies.	- Sentiment analysis. - Predictive text and chatbots. - Music composition and sequential data modeling.
Transformer Networks	Utilizes self-attention mechanisms, foundational to modern NLP.	- Large language models (e.g., GPT, BERT).

CREATEQ

Classifying Text

Natural Language Processing Architectures

Word Embeddings

CREATEQ

- Machine learning models take **vectors** (arrays of numbers) as input.
- When working with text, the first step is to vectorize it, before feeding it to the model.

One-Hot Encodings

- Consider the sentence *"The cat sat on the mat"*. The vocabulary (i.e. the set of unique words) in this sentence is: (*cat*, *mat*, *on*, *sat*, *the*).
- Encode each word by creating a *zero* vector with equal length to the vocabulary set, then place a *one* in the index coresponding to that word.
- Encoding the whole sentence = concatenating the vectors for each word
- **Downside**: this approach is **memory inefficient** because it is sparse (e.g. for a *10000* word vocabulary, *99.99%* of the elements are *zero.*

	cat	mat	on	sat	the
the	0	0	0	0	1
cat	1	0	0	0	0
sat	0	0	0	1	0
on	0	0	1	0	0
the	0	0	0	0	1
mat	0	1	0	0	0

Word Embeddings

CREATEQ

Integer Encoding

- Encode each word from the vocabulary with a **unique number**.
- This approach is efficient because the vector is **dense**.
- **Downside**: the encoding is arbitrary, as it does not capture any relationship between words, therefore being challening for a model to interpret.

Word Embeddings

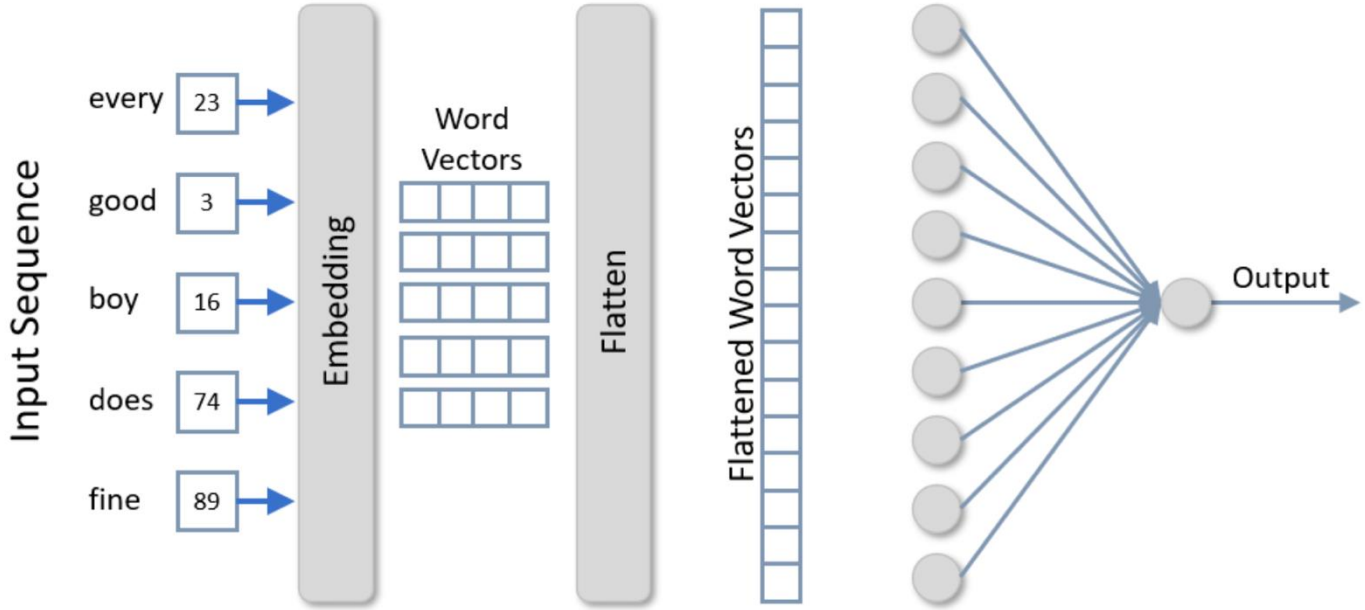
- Dense representation in which **similar words have similar encoding**.
- An embedding is a dense vector of **floating point values** (the vector length is fixed, specified as a parameter). These values are **trainable parameters** (just like weights in a neural network).
- Larger datasets require larger vector lengths (up to 1024). A higher dimensional embedding can capture fine-grained relationships between words.
- Sequences of words -> sequences of vectors.

cat	1.2	-0.1	4.3	3.2
mat	0.4	2.5	-0.9	0.5
on	2.1	0.3	0.1	0.4
sat	0.15	-1.1	3.3	0.6
the	-0.3	0.4	1.5	2.3

Classifying Text – Neural Network

CREATEQ

- Lines of text are input as **sequences**, which are arrays of integers representing individual words (e.g., indices into a dictionary).
- An **embedding layer** transforms integers, representing words into **word vectors**, or arrays of floating-point numbers.
- Word vectors encode information about **meaning of words**, such as the fact that both *"excellent"* and *"amazing"* express positive sentiment.
- The vectors are flattened to match the dimension that the network expects.



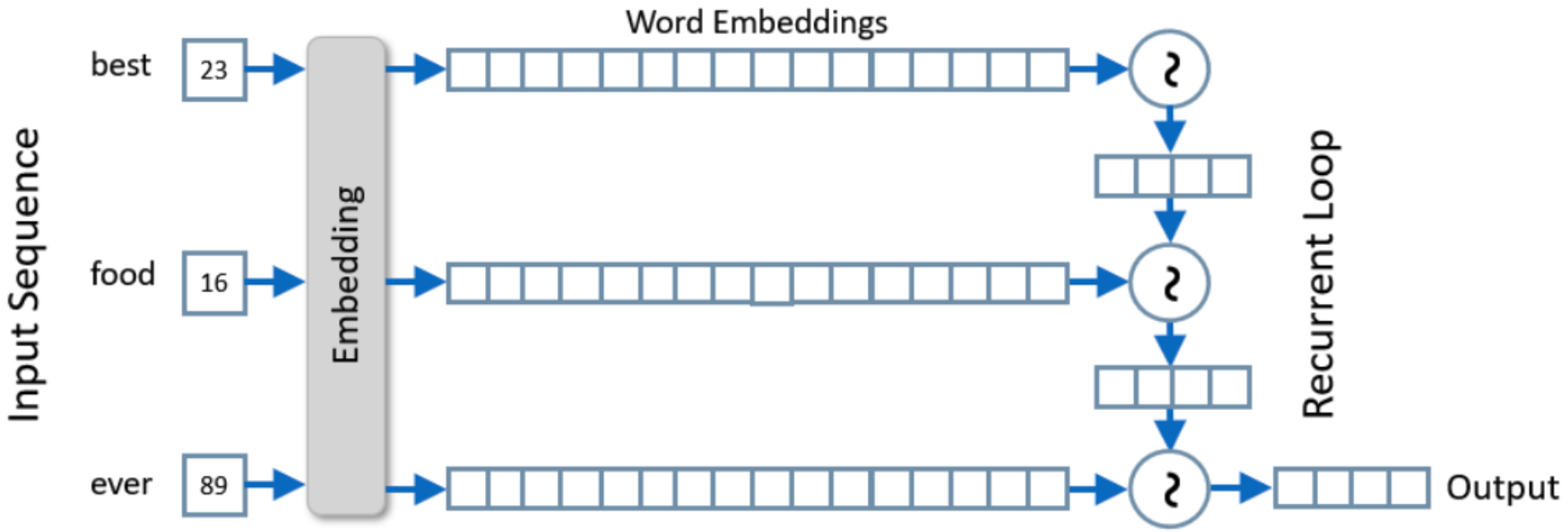
Classifying Text – Neural Network

CREATEQ

- The flattened word vectors is not a sequence anymore, as it doesn't account for the **order of the words**.
- Even if we input the words one at a time, like elements from a training dataset, simple MLPs adjust only from individual words, regardless of the order.
- We need an architecture that implements **memory**, such that, when processing a word, it remembers the **previous words**.

Recurrent Neural Networks (RNNs)

CREATEQ



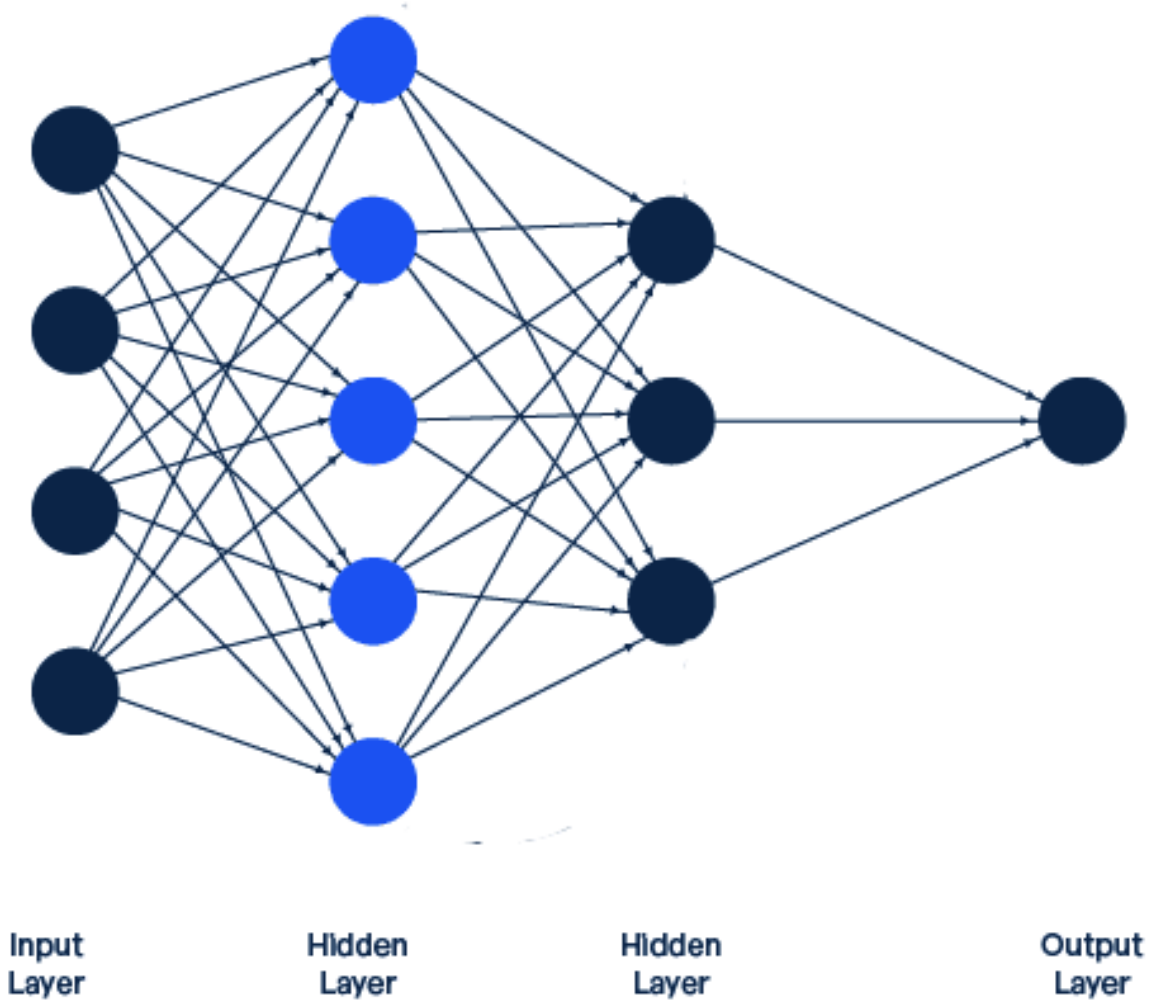
Tokenized phrase is input to the embedding layer as a **sequence of word indexes**

Each token is converted into a **word embedding**.

A recurrent network **loops** through the word embeddings in the sequence.

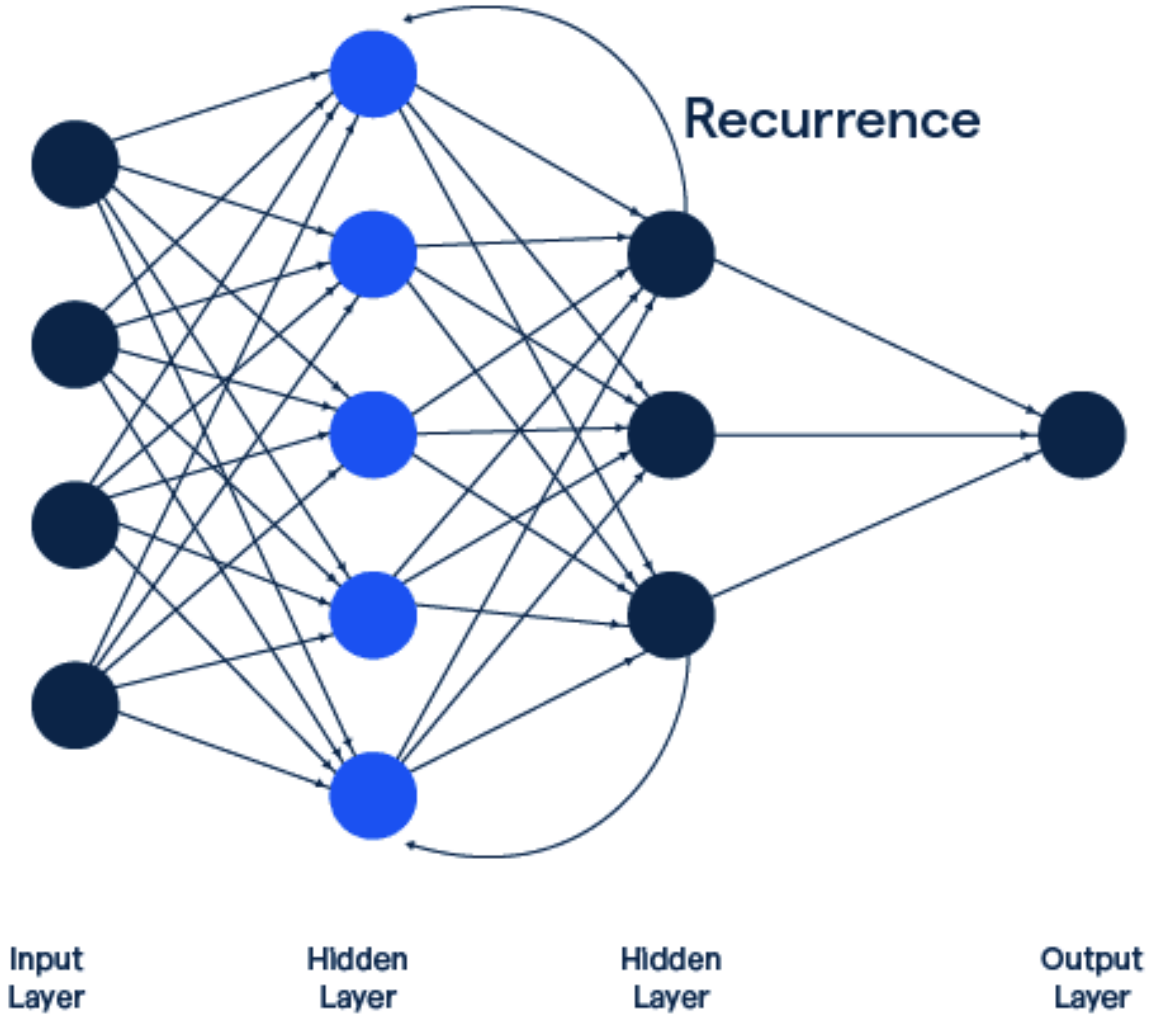
Recurrent Neural Networks (RNNs)

CREATEQ



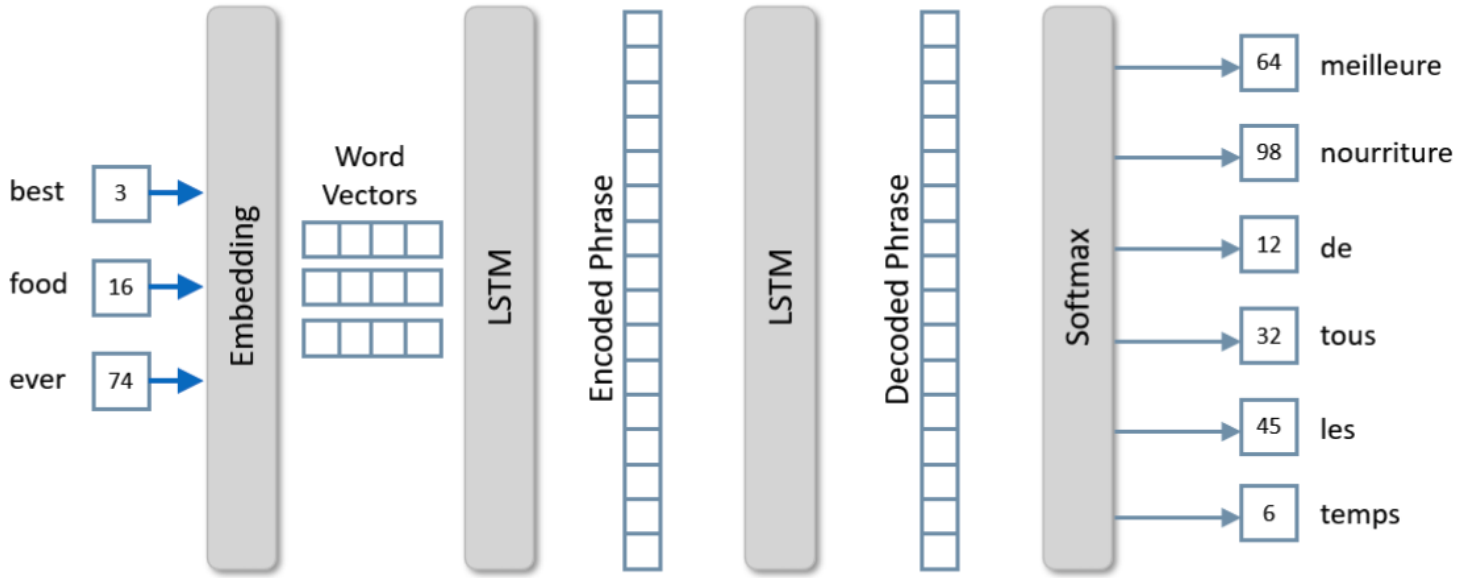
Recurrent Neural Networks (RNNs)

CREATEQ



LSTM Encoder-Decoder Architecture

CREATEQ



Embedding Layer

- Integer word encodings are mapped to dense **word vectors** (embeddings), which capture the semantic meaning of the words.

LSTM (Long Short-Term Memory) Encoder

- Processes the entire sequence, producing a fixed-size **encoded phrase** (**context vector**), summarizing the whole sequence (i.e. a compressed meaning).

Decoder (LSTM)

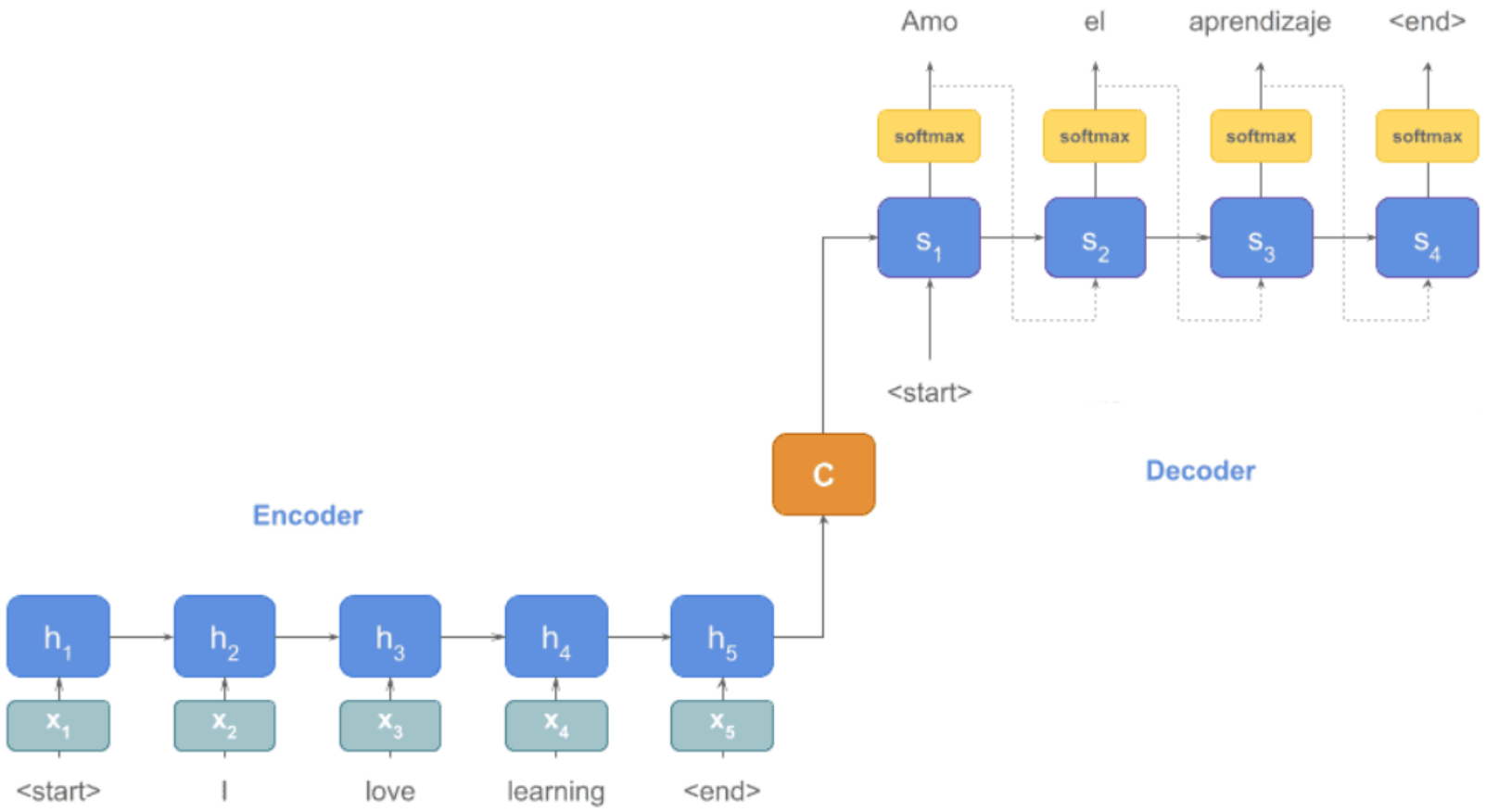
- The encoded phrase is passed to the **decoder LSTM**, which generates the target sequence step by step (word by word, or token by token)

Softmax Layer

- The decoder outputs scores over the target vocabulary at each step (for each word). Applying **Softmax** on the scores allows us to select the most likely token (e.g., meilleure, nourriture, etc.).

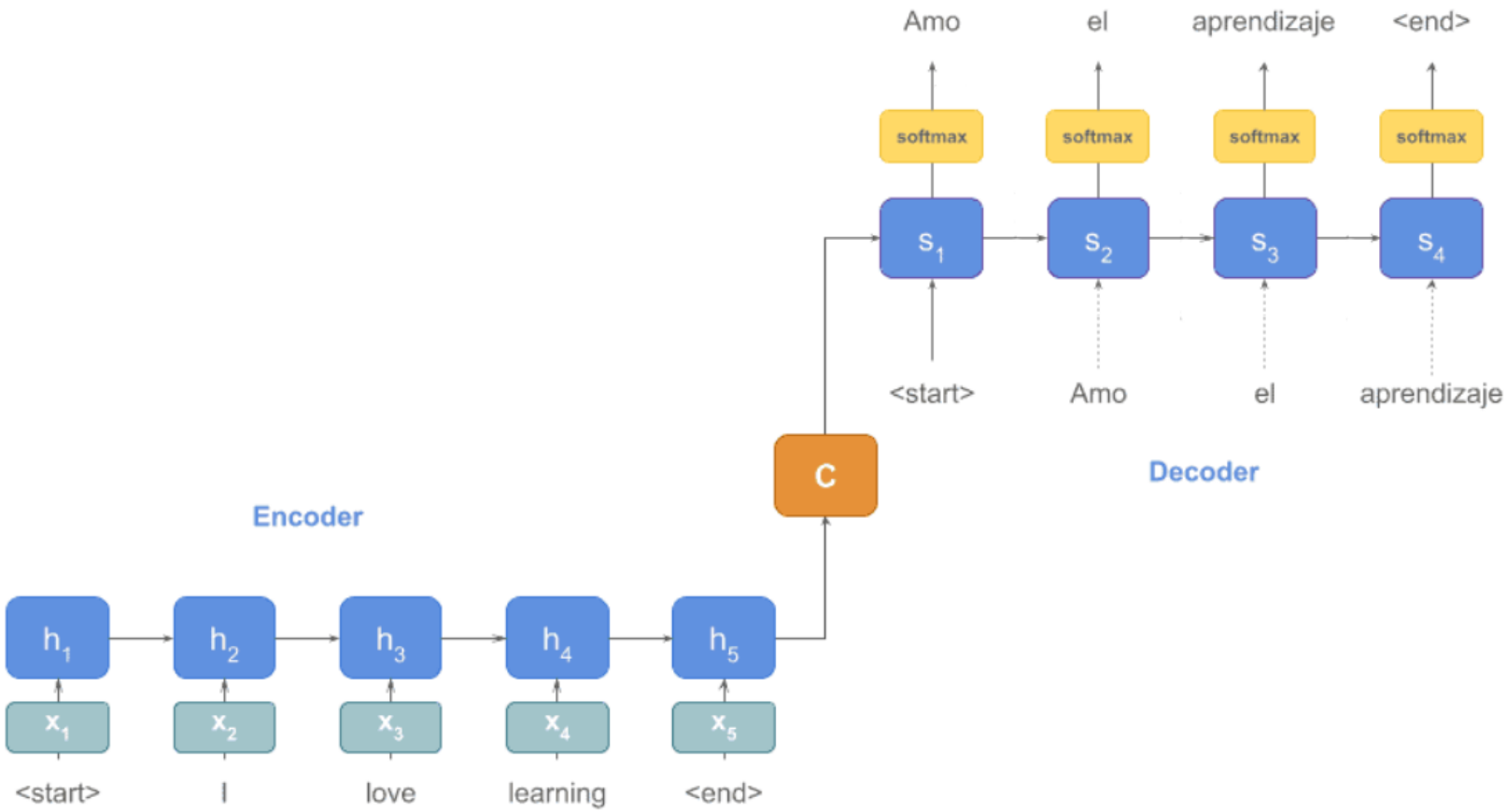
LSTM Encoder-Decoder - Inference

CREATEQ



LSTM Encoder-Decoder - Train

CREATEQ



LSTM Encoder-Decoder

CREATEQ

- RNNs often struggle with the **vanishing gradient problem**, making it difficult to learn dependencies over long sequences. LSTM modules still use recurrent operations, but they mitigate this problem.

- The Encoder-Decoder framework enables the processing of **variable-length** input and output sequences, making it versatile for tasks like machine translation and text summarization.

- RNNs may face challenges when compressing information into **fixed-size vectors**, especially with long sequences.

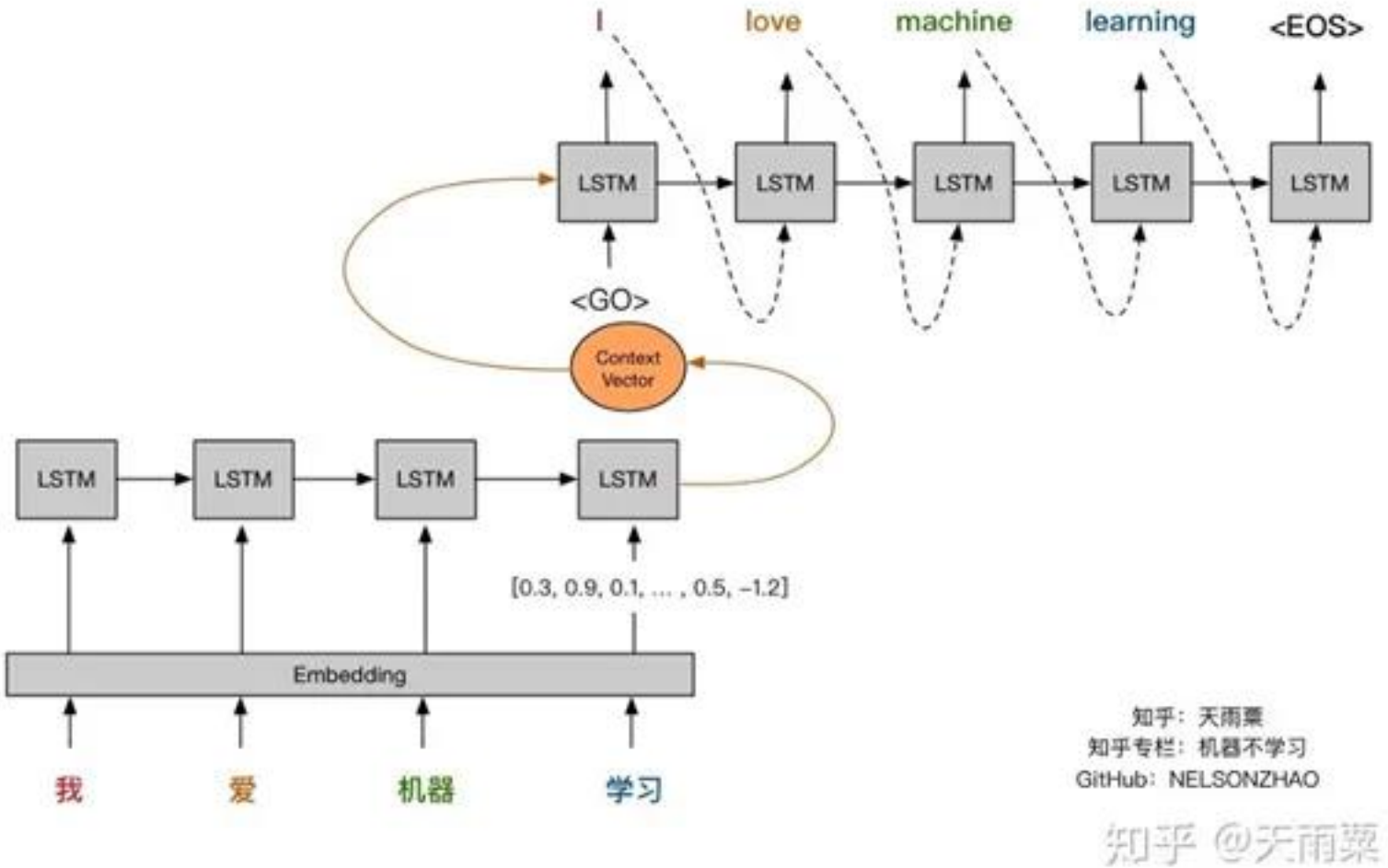
LSTM with Attention

CREATEQ

- One of the main drawbacks of this network is its **inability to extract strong contextual relations from long semantic sentences.**
- The **Attention Mechanism** is an upgrade aiming to provide
  - a more weighted/signified context to de decoder
  - a learning mechanism where the decoder can interpret were to add more focus
- Fixed-size context vector at the end of word sequence -> a **sequence of context vectors**, generated at each time-step
- These outputs are accompanied by **weighted constraints**, which represent contexts which are getting attention and therefore being trained on eventually predicting the desired results

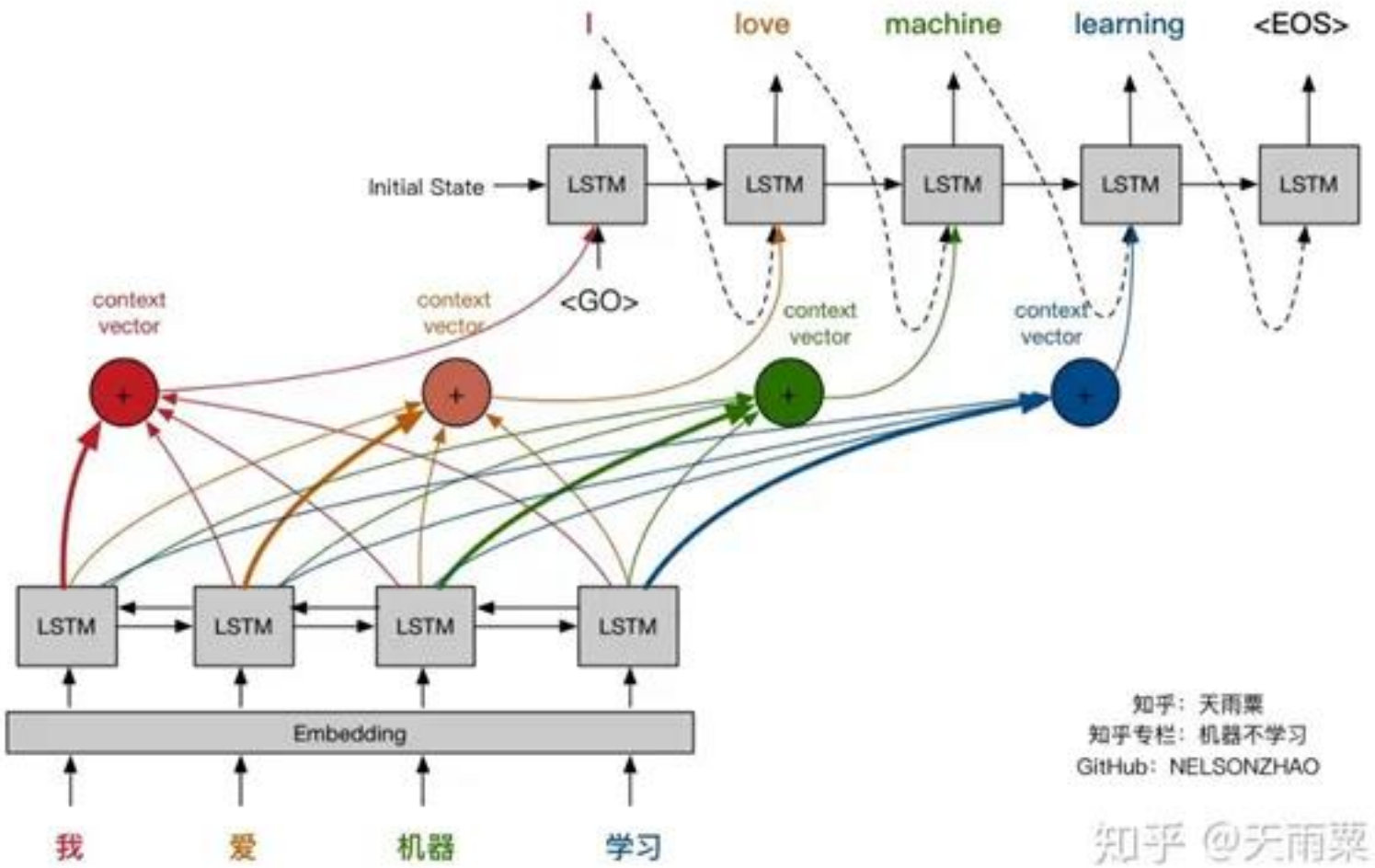
LSTM with Attention

CREATEQ



LSTM with Attention

CREATEQ



LSTM with Attention

CREATEQ

"I poured water from the bottle into the cup until **it** was **full**."

"I poured water from the bottle into the cup until **it** was **empty**."

LSTM with Attention

CREATEQ

"I poured water from the bottle into the cup until **it** was **full**."

it -> cup

"I poured water from the bottle into the cup until **it** was **empty**."

it -> bottle

Transformers

CREATEQ

- The **Transformer** (from the 2017 paper *Attention Is All You Need*) is a **sequence-to-sequence** model that relies entirely on **attention mechanisms**, discarding recurrent (LSTM) operations.
- The foundation of LLMs like **GPT** or **BERT**.

Feature	LSTM Encoder-Decoder + Attention	Transformer Encoder-Decoder
Handling Long Sequences	Processes input **sequentially**, making long sequences challenging. Attention helps, but still limited by recurrence.	Uses **Multi-Head Self-Attention**, allowing tokens to attend to all others **in parallel**, making it better at long-range dependencies; **Multi-Head** means multiple attention "heads" learn different aspects of the sequence in parallel.
Training Speed & Parallelization	Sequential processing slows down training; GPUs/TPUs **cannot fully parallelize** recurrent steps.	Entire sequences are processed in **parallel** using attention, leading to significantly faster training.
Model Complexity & Scalability	Recurrent models become cumbersome as they scale, exposing the model to **vanishing/exploding gradient** risk.	Uses **stacked attention layers**, **residual connections**, and **layer normalization**, making deep models easier to train.
Positional Information	Sequence order is inherently captured through recurrence.	**Positional encodings** are added to the **token embeddings**; This decoupling of position from content allows the model to learn flexible patterns (out-of-order, long-distance relations

Google Translate

CREATEQ

- In 2016, Google introduced the **Google Neural Machine Translation (GNMT)** system, transitioning **Google Translate** from statistical methods to neural networks.
- GNMT employed an **LSTM-based Encoder-Decoder architecture**:
  - **Encoder:** An 8-layer bidirectional LSTM network processed input sentences into context vectors.
  - **Decoder:** Another 8-layer LSTM network generated translations from these context vectors.
- This architecture improved translation fluency and accuracy by considering entire sentences rather than isolated phrases.
- By 2020, Google Translate adopted models based on the **Transformer architecture**, which offered enhanced performance and efficiency.
- The new system combined a Transformer-based encoder with an RNN-based decoder.

Google BERT

CREATEQ

- **BERT** (***B**idirectional **E**ncoder **R**epresentations from **T**ransformers*) is built on the **encoder** part of the original Transformer architecture
- It only uses **stacked encoder layers** (multi-head self-attention + feed-forward layers) and discards the decoder component.

- **Bidirectional Context**
  - Traditional language models (even Transformer-based ones like GPT) read text **left-to-right** (or right-to-left) to predict the next token.
  - **BERT** processes text **in both directions simultaneously**, letting it capture context from both sides of a word.

- **Masked Language Modelling**
  - For supervised training it is often expensive to get labelled data.
  - In order to generate training data, some tokens are randomly masked, and the model must predict these missing tokens from the surrounding context. -> **Self-Supervised Learning**

Google BERT

CREATEQ

- BERT uses a **next sentence prediction (NSP)** task during pretraining: it sees two unlabelled sentences and predicts whether the second sentence **follows** the first in the original text.

- **Pretraining + Fine-Tuning Paradigm**
  - BERT is first **pretrained** on large corpora (like Wikipedia, BookCorpus) using MLM and NSP.
  - Then it's **fine-tuned** on specific NLP tasks (e.g., sentiment classification, question answering) by adding a small output layer on top of the pretrained BERT.

- Because of its **bidirectional attention** and **large-scale pretraining**, BERT typically achieves **strong performance**, capturing deeper contextual information compared to unidirectional or shallow models.

GPT

CREATEQ

- **GPT** (**G**enerative **P**re-trained **T**ransformer) is built on the **decoder** component of the Transformer architecture.

- **Auto-Regressive Language Modeling**
  - GPT is trained to predict the next token given all previously generated tokens.
  - At each step, the model sees the existing sequence and tries to guess the next word.

- **Unidirectional Context**
  - Because GPT is predicting the next token, it only looks at the **left context** (the tokens before the current position).
  - This makes it well-suited for **generative tasks**, such as **text completion** and **story generation**.

GPT

CREATEQ

- **Pretraining**
  - GPT is pretrained on large text corpora (e.g., BooksCorpus, WebText) in an **unsupervised** manner using the next-token prediction objective.
  - This allows the model to learn grammar, facts, and general world knowledge from massive data.

- **Fine-Tuning or Prompting**
  - After pretraining, GPT can be **fine-tuned** on specific tasks (e.g., summarization, question answering) by adding a task-specific output layer or by using **prompting** and letting GPT generate answers directly.
  - Its **generative nature** makes it flexible for tasks that require producing new text.

BERT vs GPT

CREATEQ

Aspect	GPT	BERT
Core Architecture	**Transformer Decoder** only	**Transformer Encoder** only
Directionality	**Unidirectional** (left-to-right)	**Bidirectional** (attends to both left and right)
Primary Objective	**Next Token Prediction** (auto-regressive learning)	**Masked Language Modeling + Next Sentence Prediction** (self-supervised learning)
Context Handling	Looks **only** at preceding tokens	Looks at **entire sentence** context (both sides)
Strengths	**Text Generation** (stories, completions, etc.)	**Language Understanding** (classification, QA, etc.)

BERT vs GPT

CREATEQ

Task Category	Example Tasks	Best Model	Why?
Text Generation	Story writing, article generation, dialogue generation	**GPT**	GPT is trained to predict the next word, making it ideal for generating coherent text.
Text Completion	Autocomplete, code completion (e.g., GitHub Copilot)	**GPT**	GPT's auto-regressive nature enables it to complete sentences logically.
Summarization	News/article summarization (extractive & abstractive)	**GPT** (abstractive), **BERT** (extractive)	GPT generates new summaries; BERT extracts key information.
Conversational AI	Chatbots, virtual assistants	**GPT**	GPT understands context over long interactions, making it good for conversational AI.
Text Classification	Sentiment analysis, spam detection	**BERT**	BERT captures bidirectional context, useful for understanding meaning in text.

BERT vs GPT

CREATEQ

Task Category	Example Tasks	Best Model	Why?
Question Answering (QA)	Extracting answers from a paragraph	**BERT**	BERT sees the full sentence, allowing it to pinpoint precise answers.
Machine Translation	English → French, Spanish → German	**GPT**	GPT's generative nature allows it to translate fluently.
Paraphrasing	Rewriting text with different words	**GPT**	GPT can generate alternative ways to express the same meaning.
Text-based Reasoning	Logical inference, multi-step reasoning	**BERT** (structured reasoning), **GPT** (creative reasoning)	BERT excels at fact-based reasoning; GPT is better at open-ended inference.
Named Entity Recognition (NER)	Identifying names, locations, dates	**BERT**	BERT's bidirectional encoding helps recognize entities accurately.

GPT Temperature

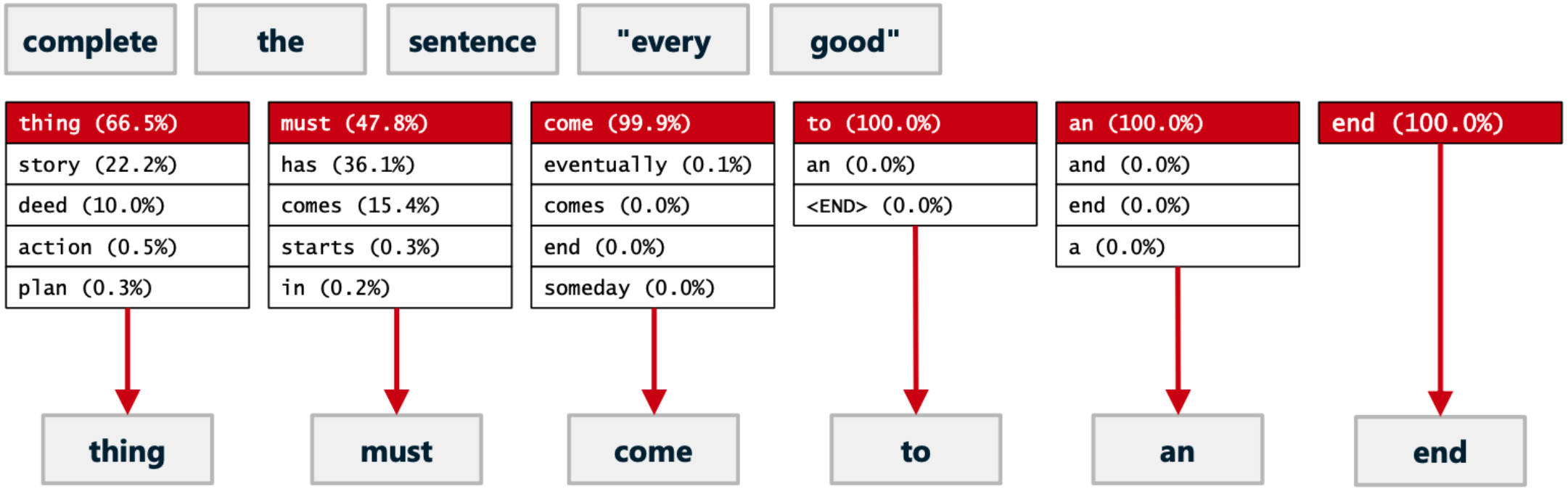
CREATEQ

- **Temperature** is a parameter that controls the randomness of a GPT model's responses. It determines how confident or creative the model should be when generating text.
- **Low Temperature** (e.g., 0.1 - 0.3) → More **deterministic** and **focused** responses. The model picks the **most likely words**, making the output precise and repetitive (good for factual answers).
- **High Temperature** (e.g., 0.7 - 1.5) → More **diverse** and **creative** responses. The model is more likely to choose **less probable words**, making the output more varied but potentially less accurate (good for storytelling or brainstorming).

How GPT Models Work

CREATEQ

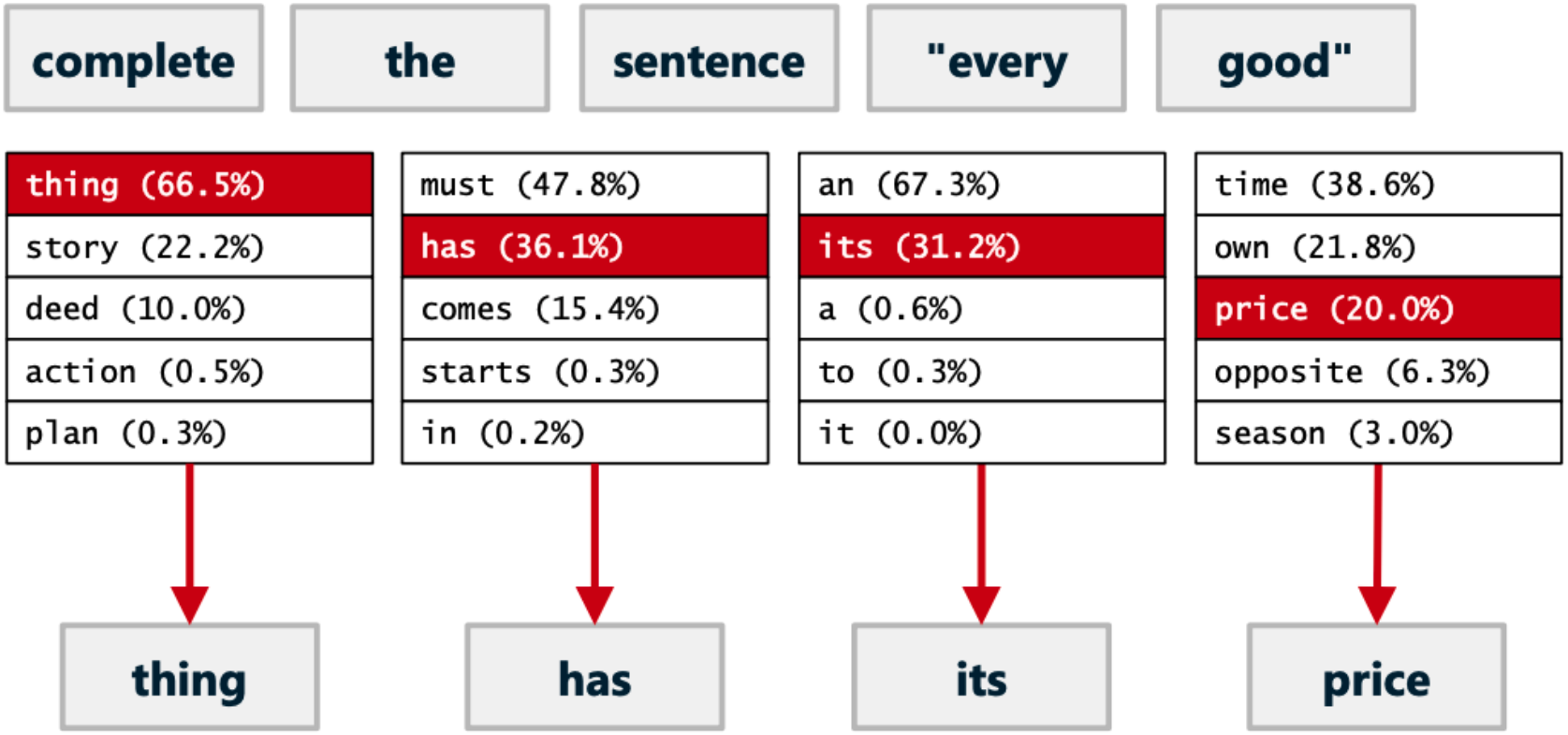
temperature = 0.0



How GPT Models Work

CREATEQ

temperature = 0.7



CREATEQ

OpenAI

Overview, Models, API

OpenAI Models - Evolution

CREATEQ

- GPT chat models:

Model	Year	Parameters	Training Dataset Size	Context Window	Capabilities
GPT-1	2018	117M	~5GB (BooksCorpus)	~512 tokens	Basic text generation (token predicted by previous tokens)
GPT-2	2019	1.5B	~40GB (WebText)	1024 tokens	More fluent, **zero-shot learning**
GPT-3	2020	175B	~570GB (Common Crawl, Wikipedia, books)	2048 tokens	Few-shot learning, better **reasoning**
ChatGPT (GPT-3.5)	2022	~175B (est.)	GPT-3 + RLHF training	4096 tokens	Fine-tuned for **conversation**, better maintenance of context in extended exchanges, less hallucinations
GPT-4	2023	~1.8T (est.)	~Trillions of words (undisclosed)	32K tokens	More accurate, **multimodal** (text & image)
GPT-4o	2024	???	Massive multimodal dataset	128K tokens	Optimized, efficient, more performance, easier to adopt

OpenAI Models

CREATEQ

- GPT chat models:

Model	Usage	Input Support	Output Support	Context Window	Max Output Tokens	Price / 1M Tokens (Input/Output)
GPT-4.5	Largest and most capable GPT model	Text, Images	Text	128,000	16,384	$75 / $150
GPT-4o	Intelligent, flexible GPT model	Text, Images	Text	128,000	16,384	$2.50 / $10
GPT-4o mini	Cost-effective, very fast GPT model	Text, Images	Text	128,000	16,384	$0.15 / $0.60
GPT-4 Turbo	Older high-intelligence GPT model	Text, Images	Text	128,000	4096	$10 / $30

- GPT reasoning models:

Model	Usage	Input Support	Output Support	Context Window	Max Output Tokens	Price / 1M Tokens (Input/Output)
o1	High-intelligence reasoning model	Text, Images	Text	200,000	100,000	$15 / $60
o3-mini	High-intelligence, very fast, cost-effective reasoning model	Text	Text	200,000	100,000	$1.10 / $4.40
o1-mini	Older reasoning model	Text	Text	128,000	65,536	$1.10 / $4.40

Generating Text with GTP-4o

CREATEQ

```
from openai import OpenAI

client = OpenAI(api_key='OPENAI_API_KEY') # Or use OPENAI_API_KEY environment variable
messages = [{ 'role': 'user', 'content': 'Why is the sky blue?' }]

response = client.chat.completions.create(
        model='gpt-4o',
        messages=messages
)

print(response.choices[0].message.content)
```

Generating Text Asynchronously

CREATEQ

```
from openai import AsyncOpenAI

client = AsyncOpenAI(api_key='OPENAI_API_KEY')
messages = [{ 'role': 'user', 'content': 'Why is the sky blue?' }]

response = await client.chat.completions.create(
        model='gpt-4o',
        messages=messages
)

print(response.choices[0].message.content)
```

Streaming the Response

CREATEQ

```
client = OpenAI(api_key='OPENAI_API_KEY')
messages = [{ 'role': 'user', 'content': 'Why is the sky blue?' }]

response = client.chat.completions.create(
        model='gpt-4o',
        messages=messages,
        stream=True
)

for chunk in response:
        content = chunk.choices[0].delta.content
        if content is not None:
                print(content, end='')
```

Specifying the Temperature

CREATEQ

```
client = OpenAI(api_key='OPENAI_API_KEY')
messages = [{ 'role': 'user', 'content': 'Why is the sky blue?' }]

response = client.chat.completions.create(
        model='gpt-4o',
        messages=messages,
        temperature=0.2
)

print(response.choices[0].message.content)
```

Generating JSON Output

CREATEQ

```
prompt = '''
        Which books comprise the Harry Potter series? Please respond
        in JSON using the following format for each book:
        {
                "title": "Harry Potter and the Prisoner of Azkaban",
                "year": 1999
        }
        '''
client = OpenAI(api_key='OPENAI_API_KEY')
messages = [{ 'role': 'user', 'content': prompt }]

response = client.chat.completions.create(
        model='gpt-4o',
        messages=messages,
        response_format={ 'type': 'json_object' }
)

print(response.choices[0].message.content)
```

Submitting an Image to GPT-4o

CREATEQ

```
messages = [{
        'role': 'user',
        'content': [
                { 'type': 'text', 'text': 'Describe what you see in this image' },
                { 'type': 'image_url', 'image_url': { 'url': 'IMAGE_URL' }} # Can be a data URL
        ]
}]

response = client.chat.completions.create(
        model='gpt-4o',
        messages=messages
)
```

CREATEQ

DeepSeek

Overview, Models, API

DeepSeek

CREATEQ

- Launched the chatbot on 10th January 2025, based on **DeepSeek-R1** reasoning model.
- By 27th of January, it surpassed ChatGPT as the **most downloaded free app** on iOS App Store in US.
- Caused a significant crash in US Stock Market: market value of Nvidia dropped by *$600bn* in one day.
- **DeepSeek-V3** claimed training cost $5.5m est. VS GPT-4o training cost $100m est.
- **DeepSeek-V3** claimed training cost is often called into question by experts.
- **DeepSeek-R1** (**open-source**, undisclosed training cost) matches or exceeds GPT-4o in tasks requiring logical reasoning or problem solving.

DeepSeek Architecture

CREATEQ

- **DeepSeek** models use significantly fewer resources compared to others, thanks to the **Mixture-of-Experts** (**MoE**) Framework.
- The large models is split into **multiple smaller "expert" network,** each specialized for specific applications.
- A separate **Gating Network** determines which expert(s) to activate for each input.
- This leads to a **sparse activation**, meaning that most of the parameters remain inactive.
- For **DeepSeek-R1**, usually approx. *37b* parameters out of *671b* are activated.

Models & Pricing

CREATEQ

Model		deepseek-chat	deepseek-reasoner
Context Length		64K	64K
Max Cot Tokens		-	32K
Max Output Tokens		8K	8K
Standard Price (UTC 00:30-16:30)	1m Tokens Input (Cache Hit)	$0.07	$0.14
	1m Tokens Input (Cache Miss)	$0.27	$0.55
	1m Tokens Output	$1.10	$2.19
Discount Price (UTC 16:30-00:30)	1m Tokens Input (Cache Hit)	$0.035（50% OFF）	$0.035（75% OFF）
	1m Tokens Input (Cache Miss)	$0.135（50% OFF）	$0.135（75% OFF）
	1m Tokens Output	$0.550（50% OFF）	$0.550（75% OFF）

- The **deepseek-chat** model points to **DeepSeek-V3**.
- The **deepseek-reasoner** model points to **DeepSeek-R1**.

Chat Completion

CREATEQ

- The DeepSeek API uses an API format **compatible with OpenAI**.
- Just use the OpenAI SDK, with modified config, to access the DeepSeek API.

```
from openai import OpenAI

# for backward compatibility, use `https://api.deepseek.com/v1` as `base_url`.
client = OpenAI(api_key="<your API key>", base_url="https://api.deepseek.com")

response = client.chat.completions.create(
        model="deepseek-chat",
        messages=[
                {"role": "system", "content": "You are a helpful assistant"},
                {"role": "user", "content": "Hello"},
        ],
        max_tokens=1024,
        temperature=0.7,
        stream=False
)

print(response.choices[0].message.content)
```

FIM (Fill-In-The-Middle) Completion (Beta)

CREATEQ

- For **Fill-in-the-Middle** completion, users can provide a **prefix** and a **suffix** (optional), and the model will complete the content in between.

```
from openai import OpenAI

# set `base_url="https://api.deepseek.com/beta"` for this feature.
client = OpenAI(
        api_key="<your API key>",
        base_url="https://api.deepseek.com/beta",
)
response = client.completions.create(
        model="deepseek-chat",
        prompt="def fib(a):",
        suffix=" return fib(a-1) + fib(a-2)",
        max_tokens=128)

print(response.choices[0].text)
```

CREATEQ

Google Gemini

Overview, Models, API

Google Gemini

CREATEQ

- Available by REST API with Google account
- SDKs available for Python, Node.js, Go, Dart, Swift, and more

Model	Usage	Input Support	Output Support	Context Window	Max Output Tokens	Price / 1M Tokens (Input/Output)
Gemini 2.0 Pro	Most powerful Gemini model	Text, Images, Video, Audio	Text	2,048,576	8,192	-
Gemini 2.0 Flash	Next generation features, speed, thinking, realtime streaming, and multimodal generation	Text, Images, Video, Audio	Text, Images, Audio	1,048,576	8,192	$0.10 (text, image, video) $0.70 (audio) / $.30
Gemini 2.0 Flash Lite	Optimized for cost efficiency and low latency	Text, Images, Video, Audio	Text	1,048,576	8,192	$0.075 / $0.30

Google Gemini

CREATEQ

- Gemini has been trained on a massive corpus of multilingual and multimodal data sets.
- Architecture is **Transformer-based**.
- It uses **Cross-Modal Attention**, which allows it tot integrate and process information from multiple modalities (i.e. learns relationships and dependencies between different types of data).
- Google Gemini supports interleaved sequences of **audio**, **image**, **text** and video as inputs and can produce interleaved text and image outputs.

Text Generation

CREATEQ

```
from google import genai

client = genai.Client(api_key="GEMINI_API_KEY")

response = client.models.generate_content(
        model="gemini-2.0-flash",
        contents=["How does AI work?"])

print(response.text)
```

Generate a text stream

CREATEQ

```
from google import genai

client = genai.Client(api_key="GEMINI_API_KEY")

response = client.models.generate_content_stream(
        model="gemini-2.0-flash",
        contents=["How does AI work?"])

for chunk in response:
        print(chunk.text, end="")
```

Generate text from text-and image input

CREATEQ

```
from PIL import Image
from google import genai


client = genai.Client(api_key="GEMINI_API_KEY")

image = Image.open("/path/to/organ.png")
response = client.models.generate_content(
        model="gemini-2.0-flash",
        contents=[image,"Tell me about this instrument"])

print(response.text)
```

CREATEQ

Danke

Mulțumesc

Thank You!

Hvala!

createq.space