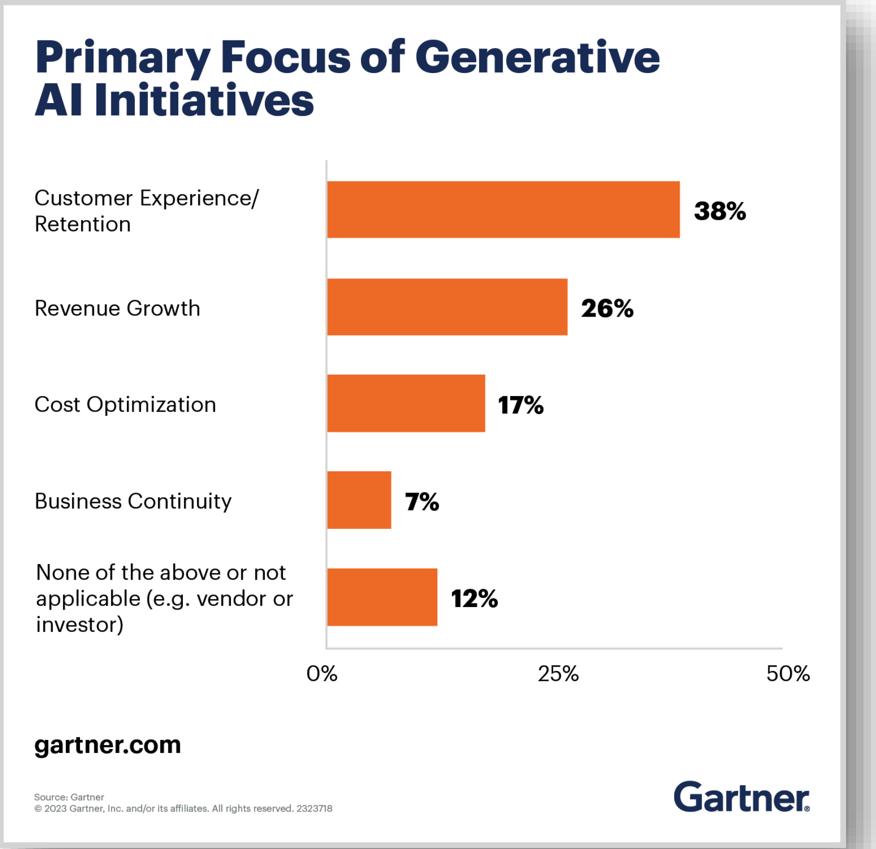
Large Language Models

Jeff Prosise

@jprosise

Why Generative AI?

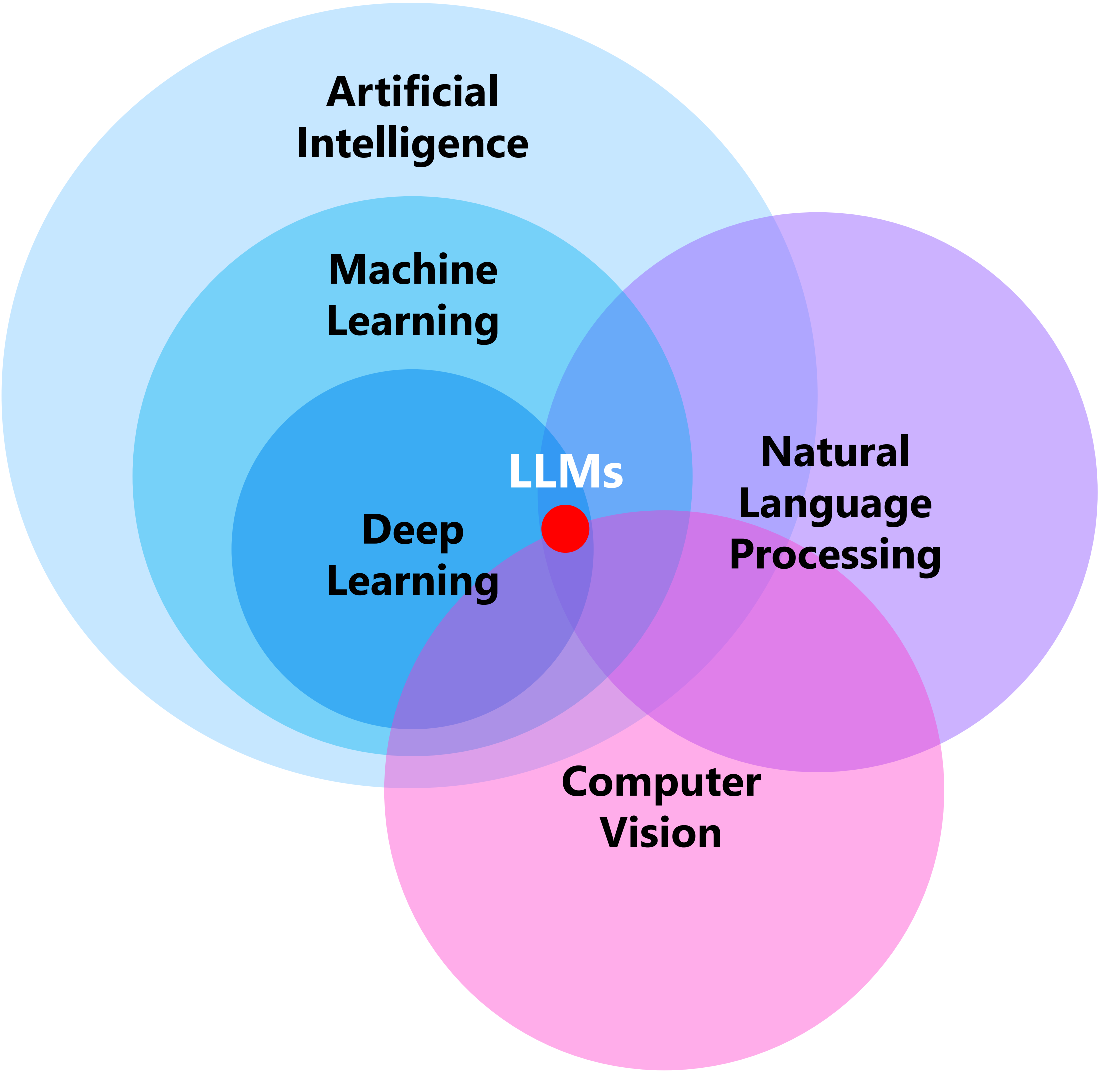


“Gartner sees generative AI becoming a **general-purpose technology** with an impact similar to that of the **steam engine, electricity and the internet**. The hype will subside as the reality of implementation sets in, but the **impact of generative AI will grow** as people and enterprises discover more innovative applications for the technology in daily work and life.”

https://www.gartner.com/en/topics/generative-ai

Large Language Models (LLMs)

- Deep-learning models (neural networks) that generate text
  - All accept text as input
    - "Write a poem about deep learning"
  - Multimodal LLMs (MLLMs) accept text, images, and sometimes audio and video as input
    - "Describe what you see on this map"
- Billions of parameters trained on trillions of words



Generating Text

```
Write a short story that begins with "It was a dark and stormy night."
```

```
It was a dark and stormy night. The wind was howling through the streets, making it
difficult for anyone to stay on their feet. The rain was coming down in sheets,
drenching anyone who dared to step outside. Inside, a young woman sat huddled in a
corner of her small apartment, listening to the storm rage on outside.

She had always been afraid of storms, ever since she was a child. But tonight, there was
something different about this one. It seemed to be getting closer and closer, and she
could feel the panic rising in her chest. She tried to shake it off, telling herself
that it was just a normal storm, but she couldn't shake the feeling that something was
coming.

...
```

Translating Text

```
Translate the following into French: "What time does the train leave for Paris?"
```

```
"À quelle heure part le train pour Paris?"
```

Generating Code

```
Create a Python function that accepts an array of numbers as input, bubble sorts the
numbers, and returns a sorted array
```

```python
def bubble_sort(numbers):
    # Get the length of the array
    n = len(numbers)

    # Traverse through all the array elements
    for i in range(n-1):
        # Last i elements are already in place
        for j in range(0, n-i-1):
            # Traverse the array from 0 to n-i-1
            # Swap if the element found is greater than the next element
            if numbers[j] > numbers[j+1]:
                numbers[j], numbers[j+1] = numbers[j+1], numbers[j]

    return numbers
```

Translating Code

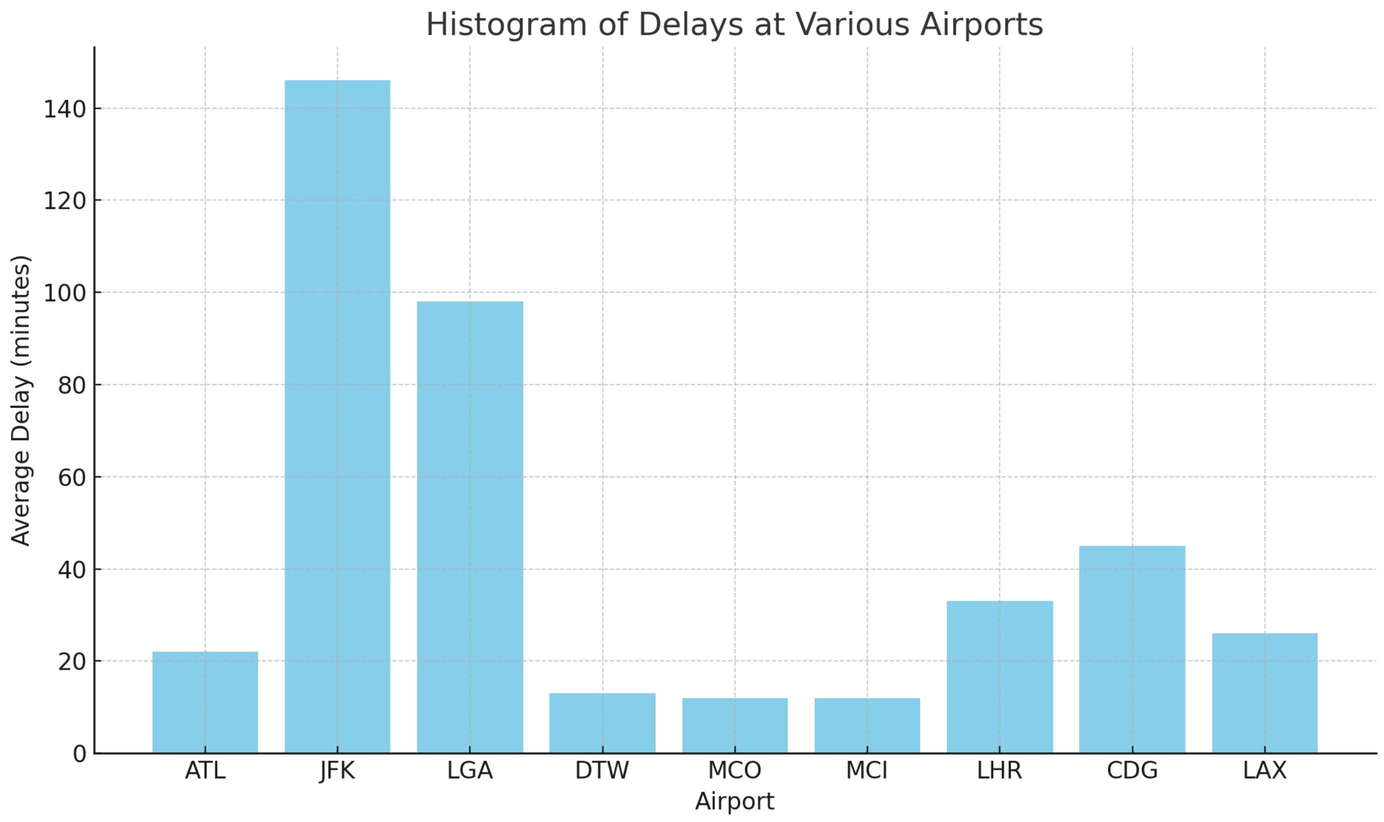
```
Translate the bubble_sort function into FORTRAN
```

```
FUNCTION bubble_sort(arr, n) RESULT(result)
      ...
    DO i = 1, n
        DO j = 1, n-i-1
            IF (arr(j) > arr(j+1)) THEN
                temp = arr(j)
                arr(j) = arr(j+1)
                arr(j+1) = temp
            END IF
        END DO
    END DO

    result = arr
END FUNCTION bubble_sort
```

Generating Charts and Graphs

```
Generate a histogram from the data below showing delays at various airports
```

Airport	Average Delay
ATL	22
JFK	146
LGA	98
DTW	13
MCO	12
MCI	12
LHR	33
CDG	45
LAX	26



Analyzing Charts

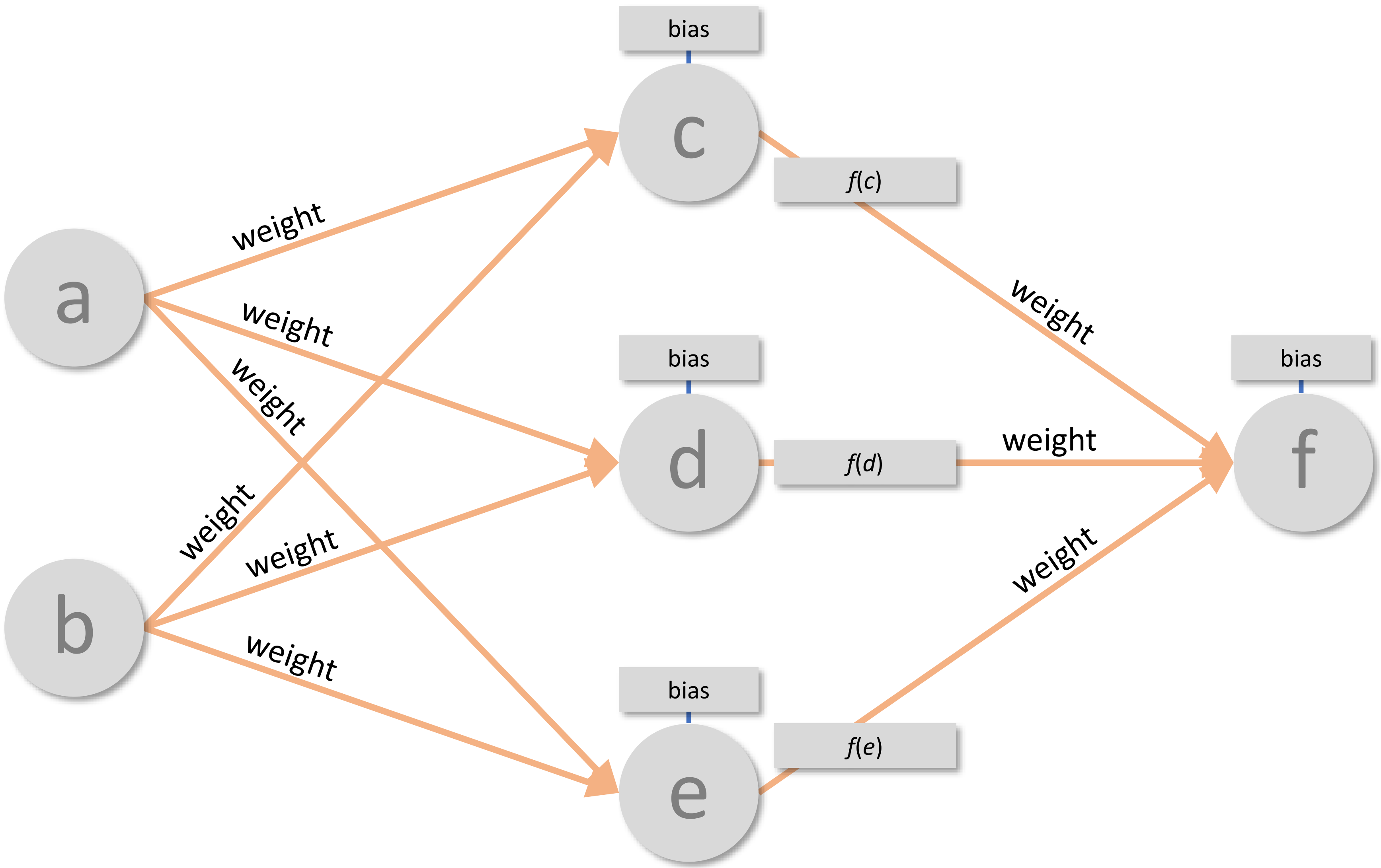
```
Explain the chart on the preceding slide
```

```
This chart is a histogram depicting the average delays (in minutes) at various airports.
Here are some key observations:

- JFK has the highest average delay, approaching 140 minutes.
- LGA follows with a slightly lower average delay, around 100 minutes.
- Most other airports, such as ATL, DTW, MCO, and MCI, have much lower average delays,
  typically around 20 minutes or lower.
- CDG and LHR, two major international airports, show moderate delays (CDG with around
  70 minutes and LHR with about 50 minutes)

The light blue bars indicate the magnitude of the delay for each airport, and the title
of the chart indicates it's a "Histogram of Delays at Various Airports."
```

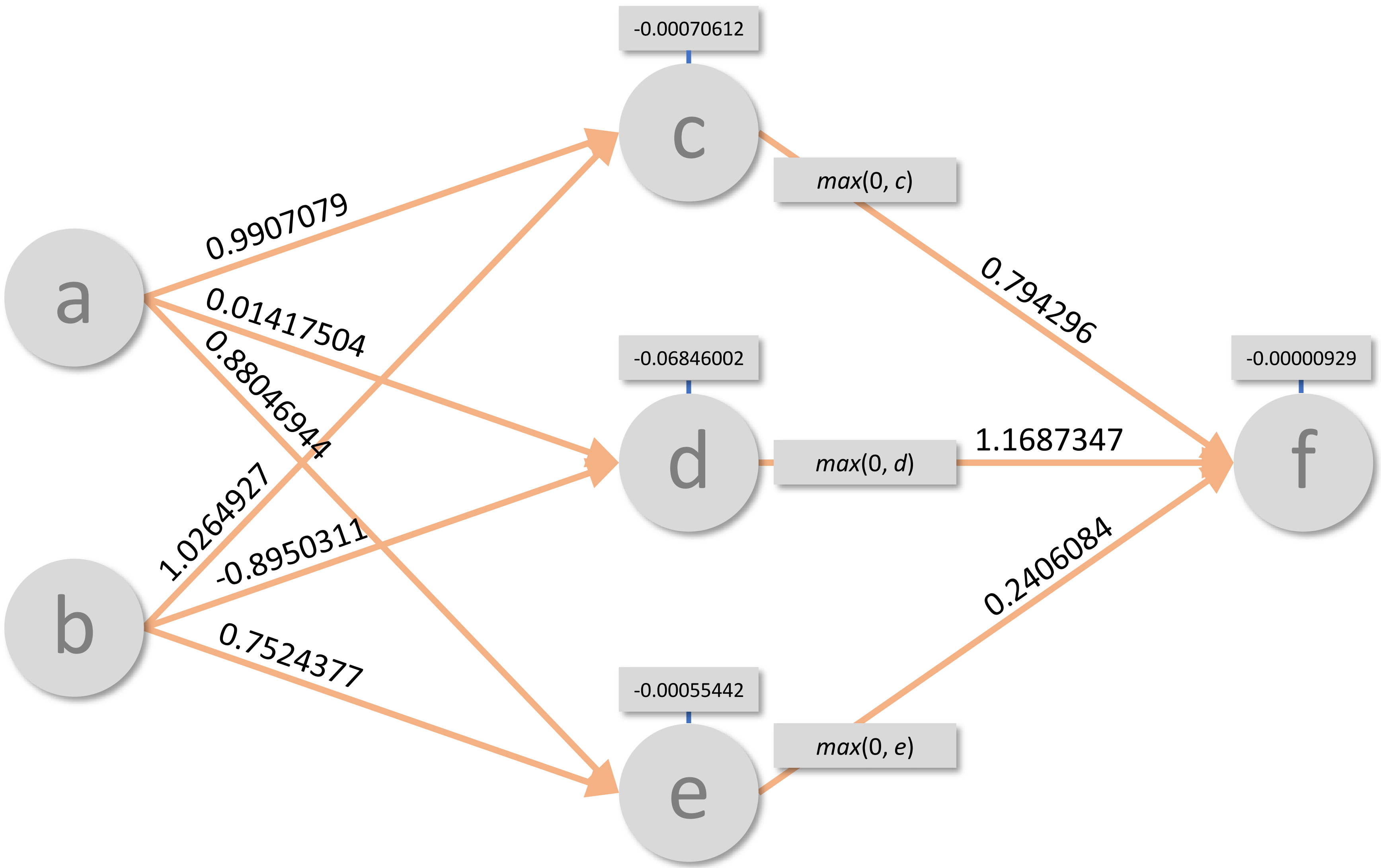
Demo

LLMs in Action

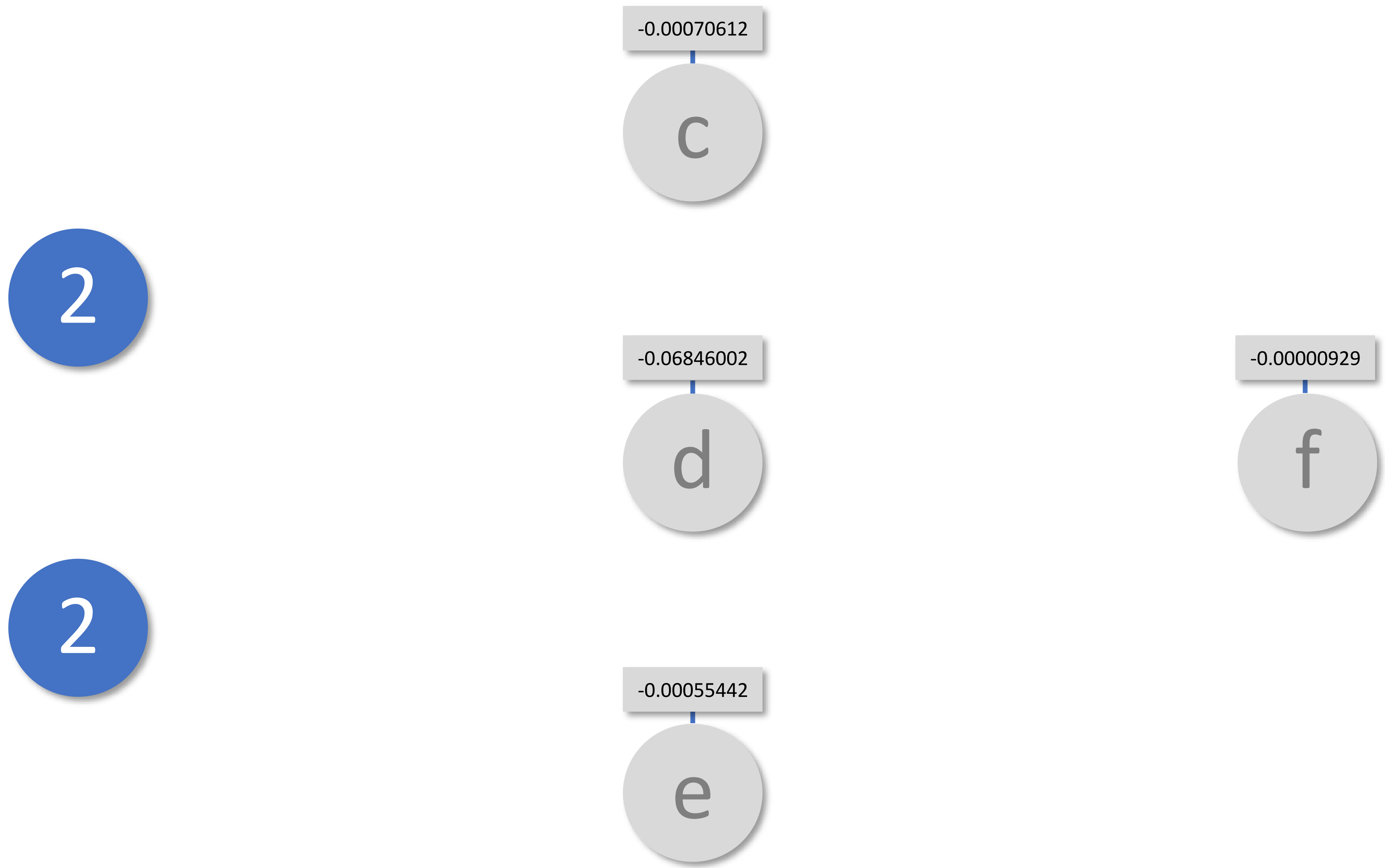
How Neural Networks Work



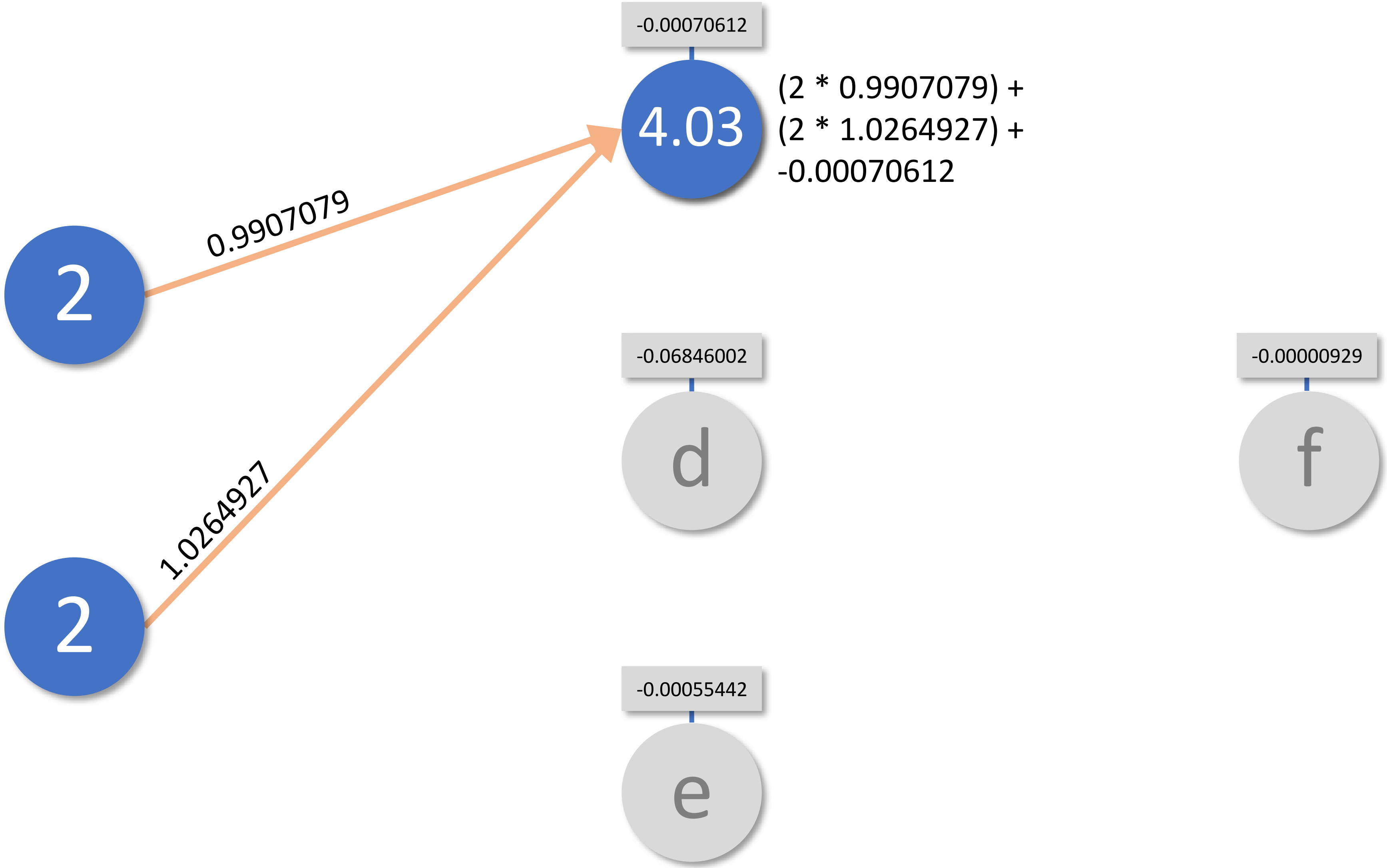
How Neural Networks Work, Cont.



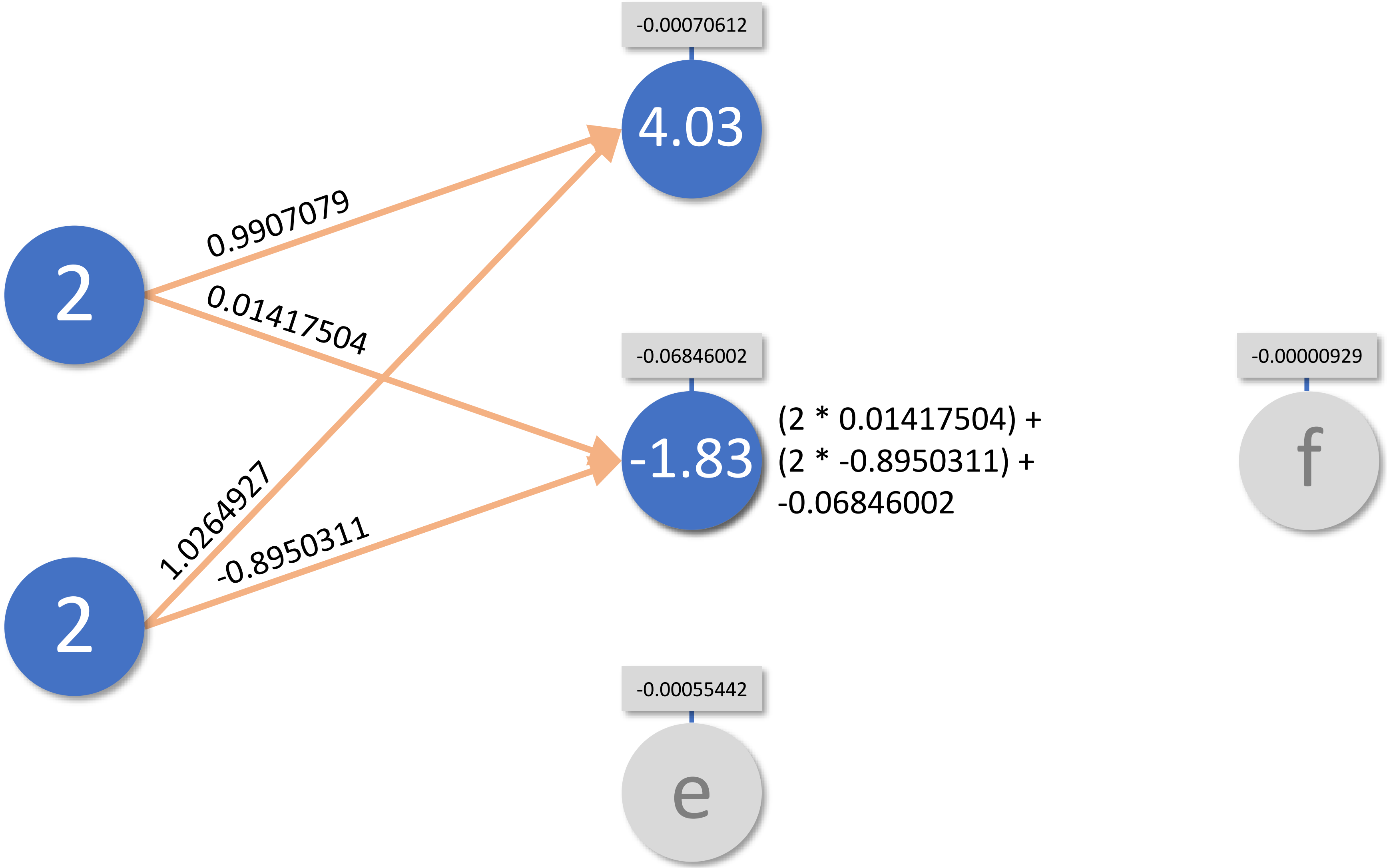
How Neural Networks Work, Cont.



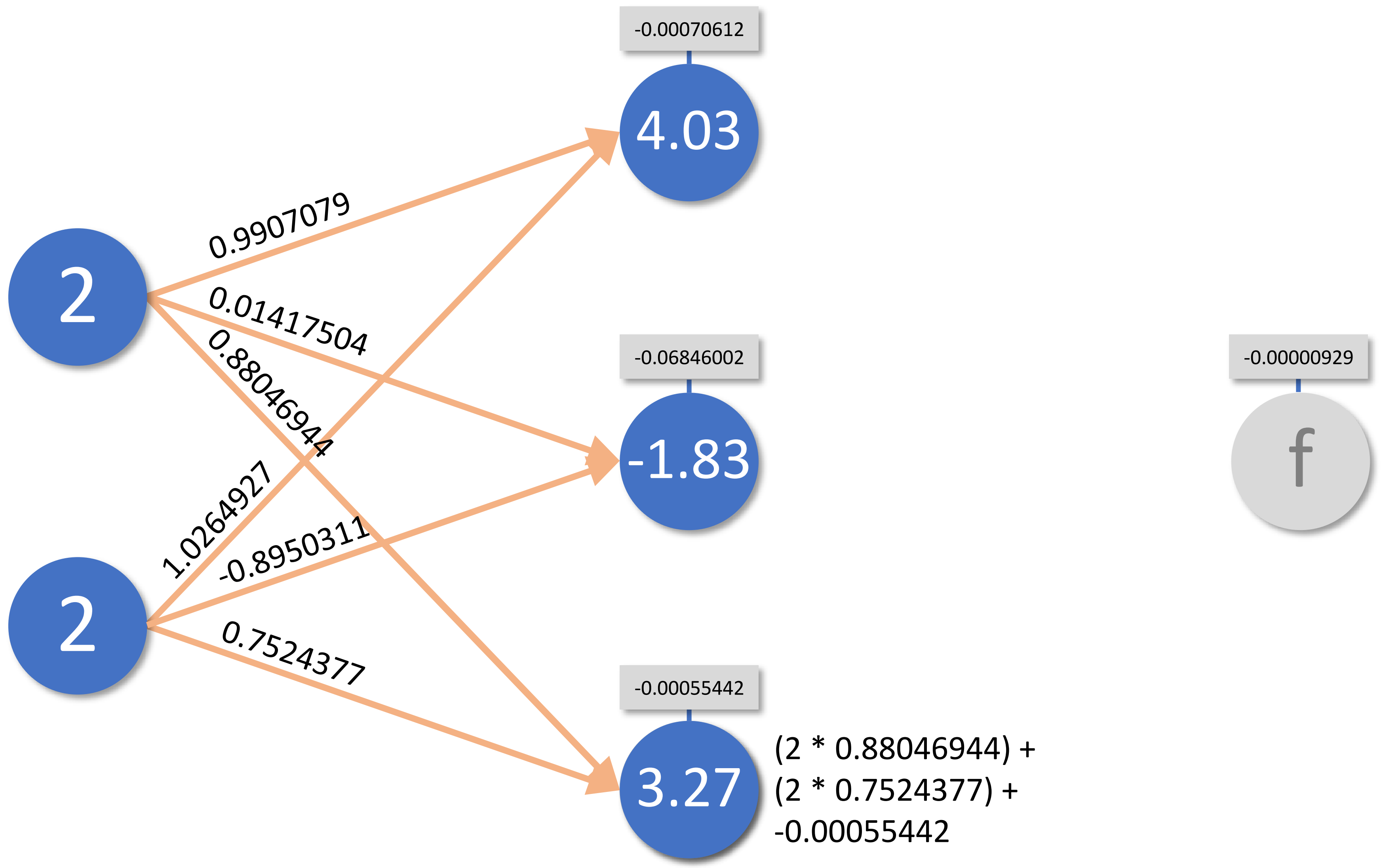
How Neural Networks Work, Cont.



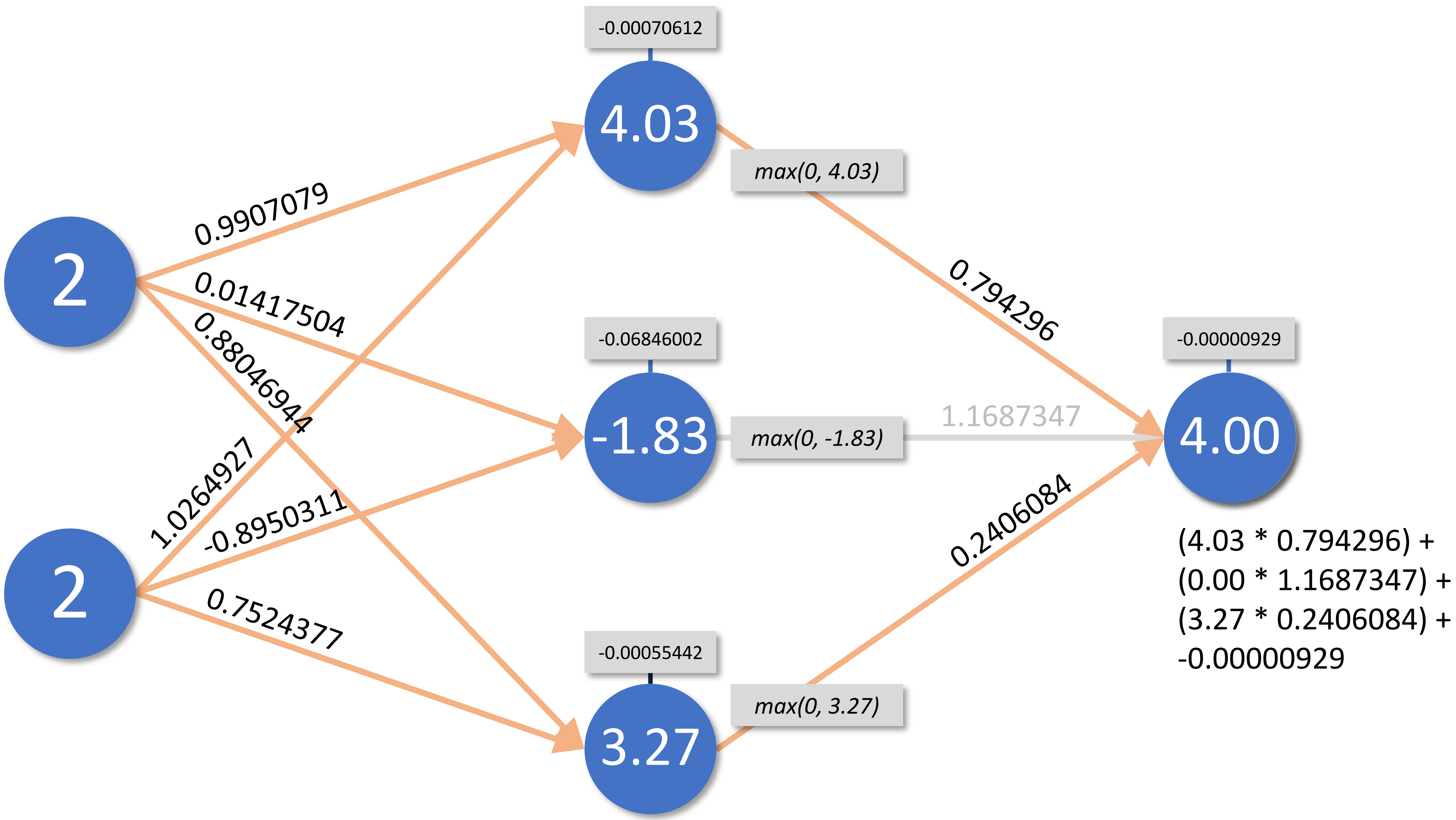
How Neural Networks Work, Cont.



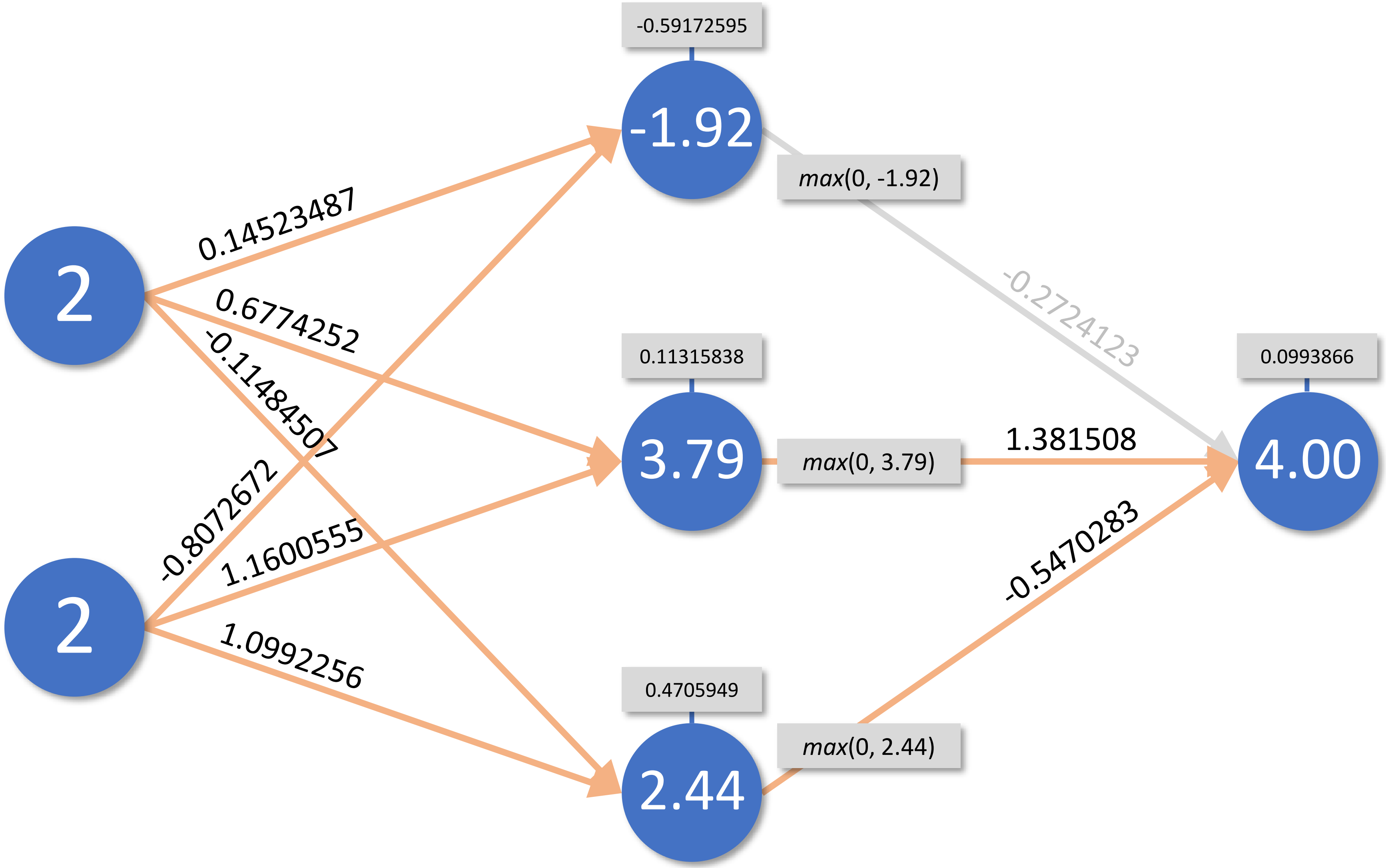
How Neural Networks Work, Cont.



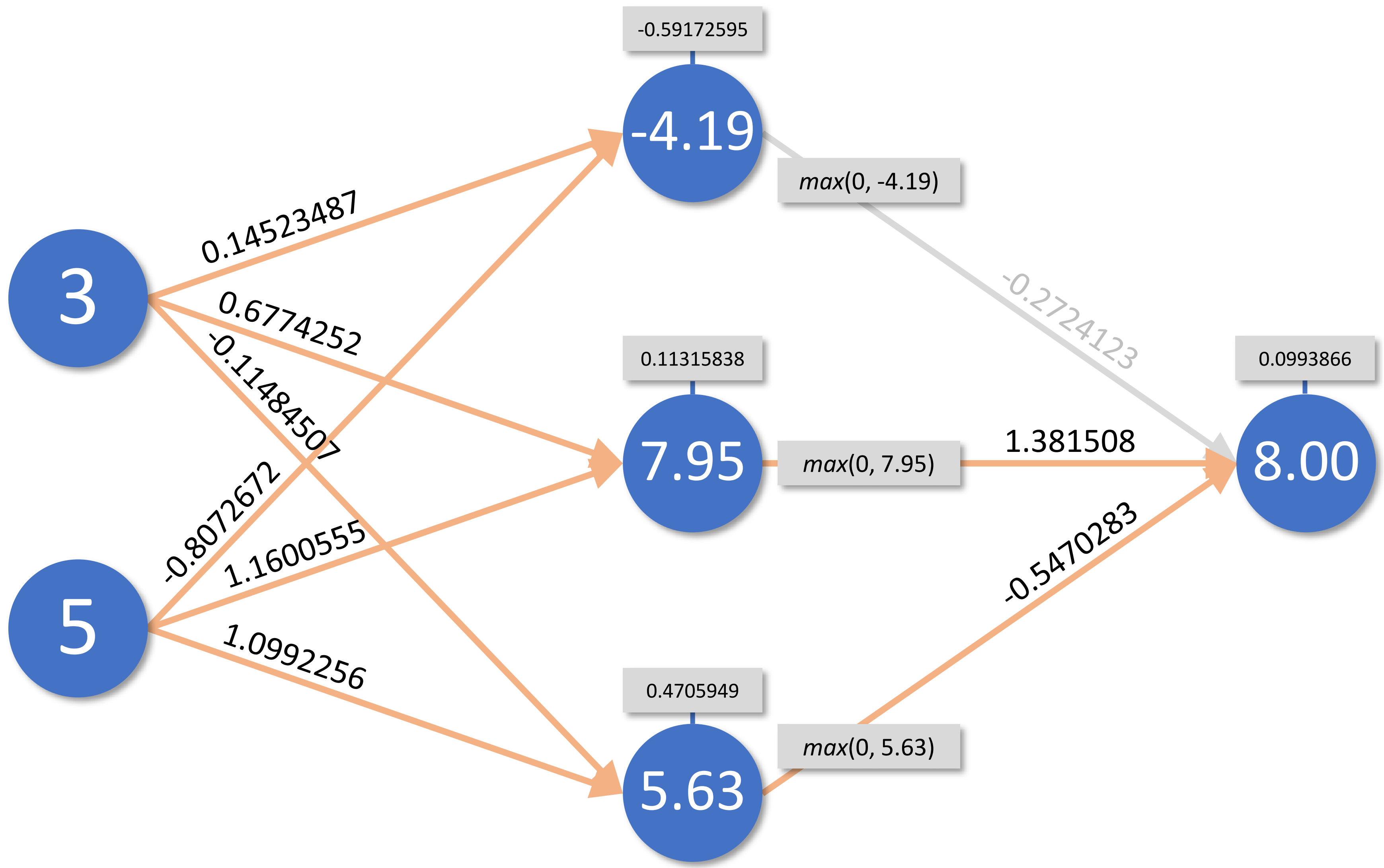
How Neural Networks Work, Cont.



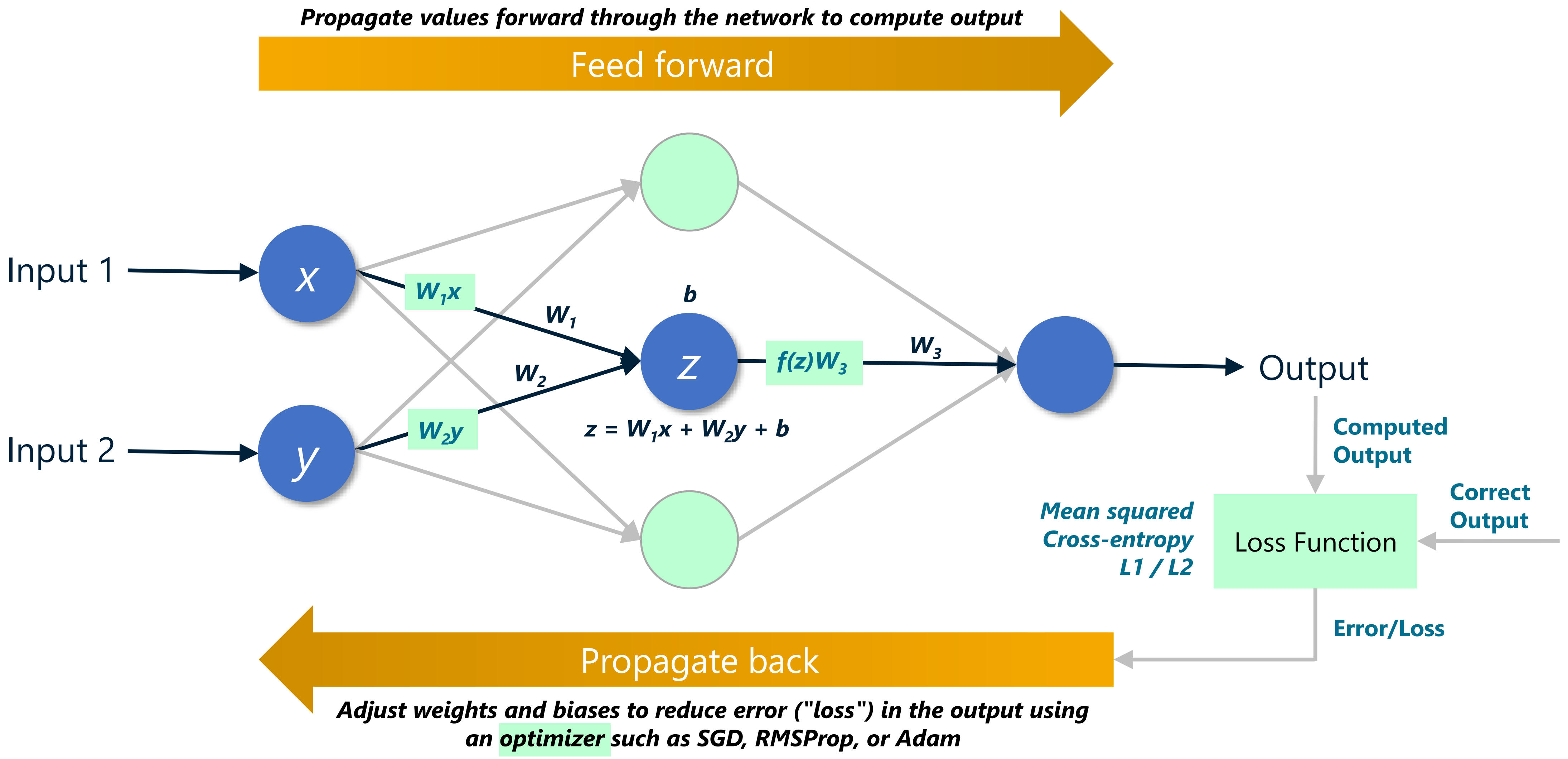
Different Weights and Biases, Same Result



Changing the Inputs

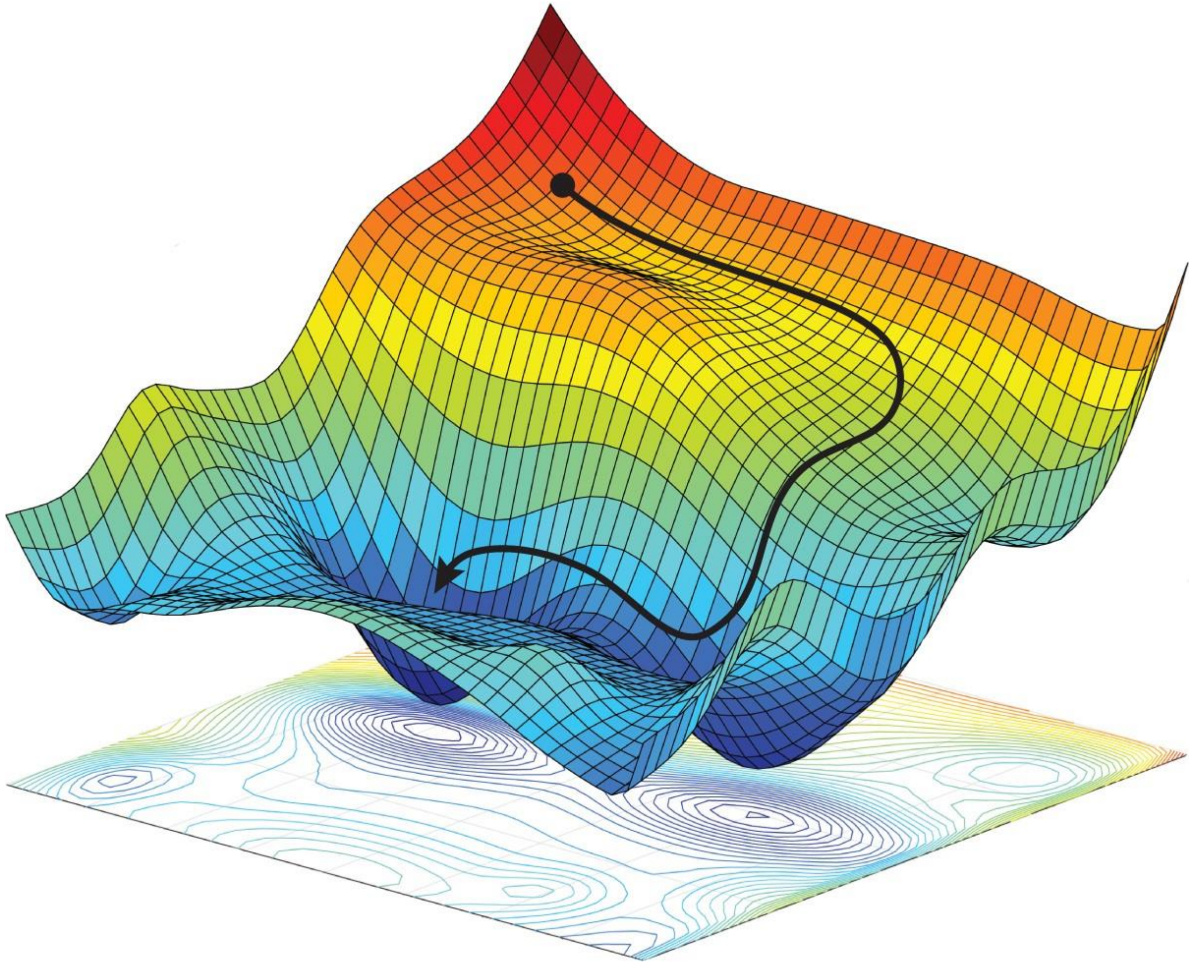


Backpropagation



Gradient Descent

- Training involves navigating multi-dimensional loss landscape searching for global minimum
- Weights and biases are adjusted using gradients computed from partial derivatives
  - Partial derivatives of loss function with respect to individual weights
- Gradients are multiplied by learning rate to make steps incremental

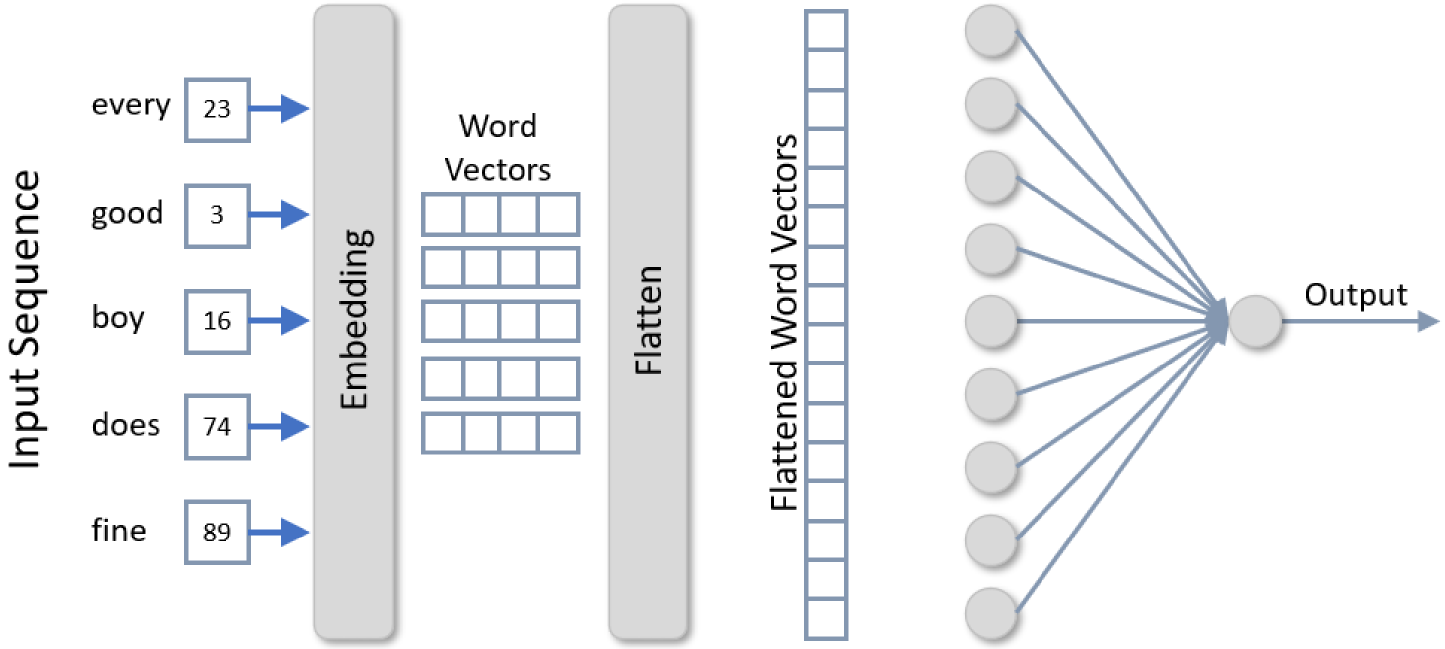


Source: https://arxiv.org/pdf/1805.04829 ("Spatial Uncertainty Sampling for End-to-End Control")

Classifying Text

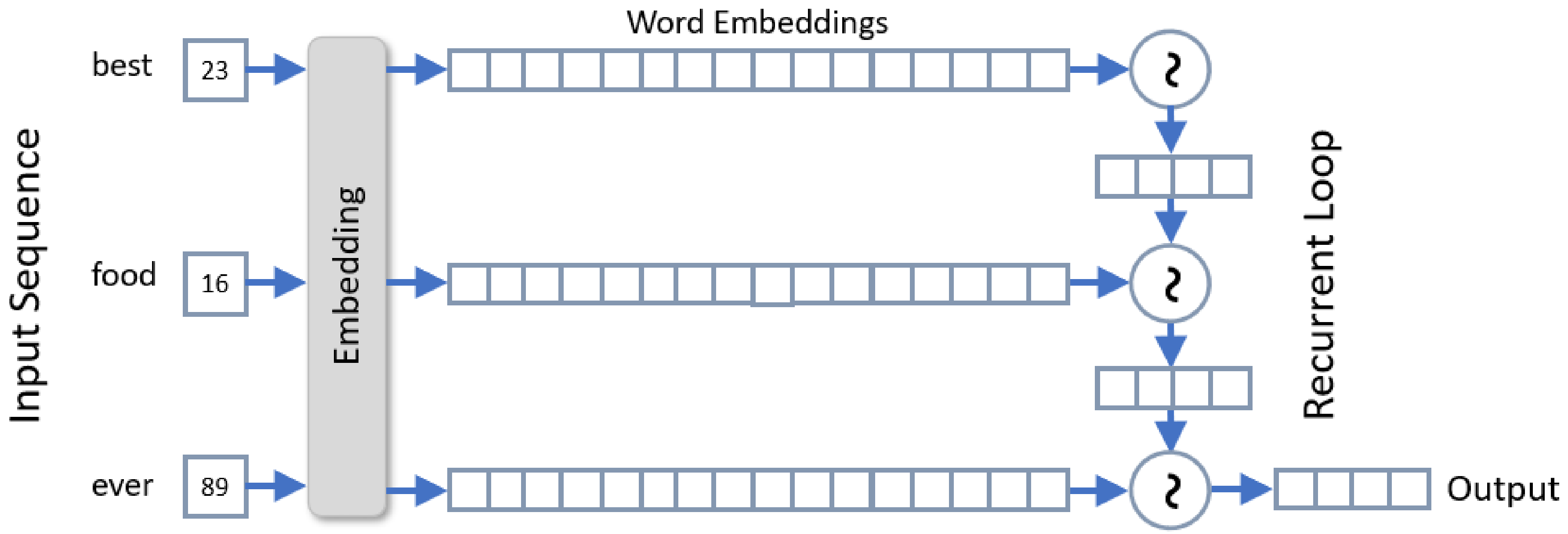
- Embedding layers turn sequences of word tokens into arrays of *word vectors*, which encode information about relationships between words
- Word vectors are classified using multilayer perceptron (MLP)

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 **relationships between words**, such as the fact that both "excellent" and "amazing" express positive sentiment.



Recurrent Neural Networks (RNNs)

- Factor word order into the vectors they produce

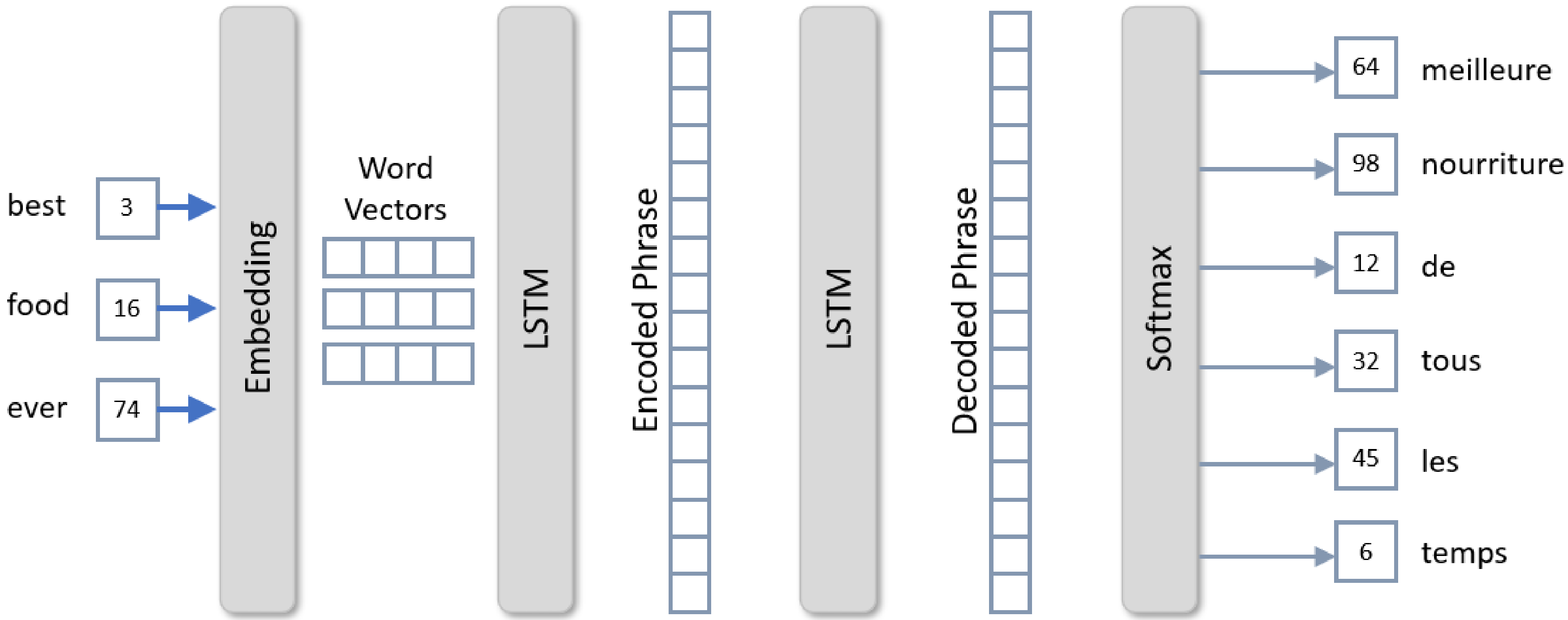


Tokenized phrase is input to the embedding layer as a **sequence of word indexes**

Each token is converted into a **word embedding**, which is an array (vector) of floating-point values modeling each word's **relationship to other words**

A recurrent layer **loops** through the word embeddings in the sequence, computing a value for each that **factors in the output from the previous iteration**

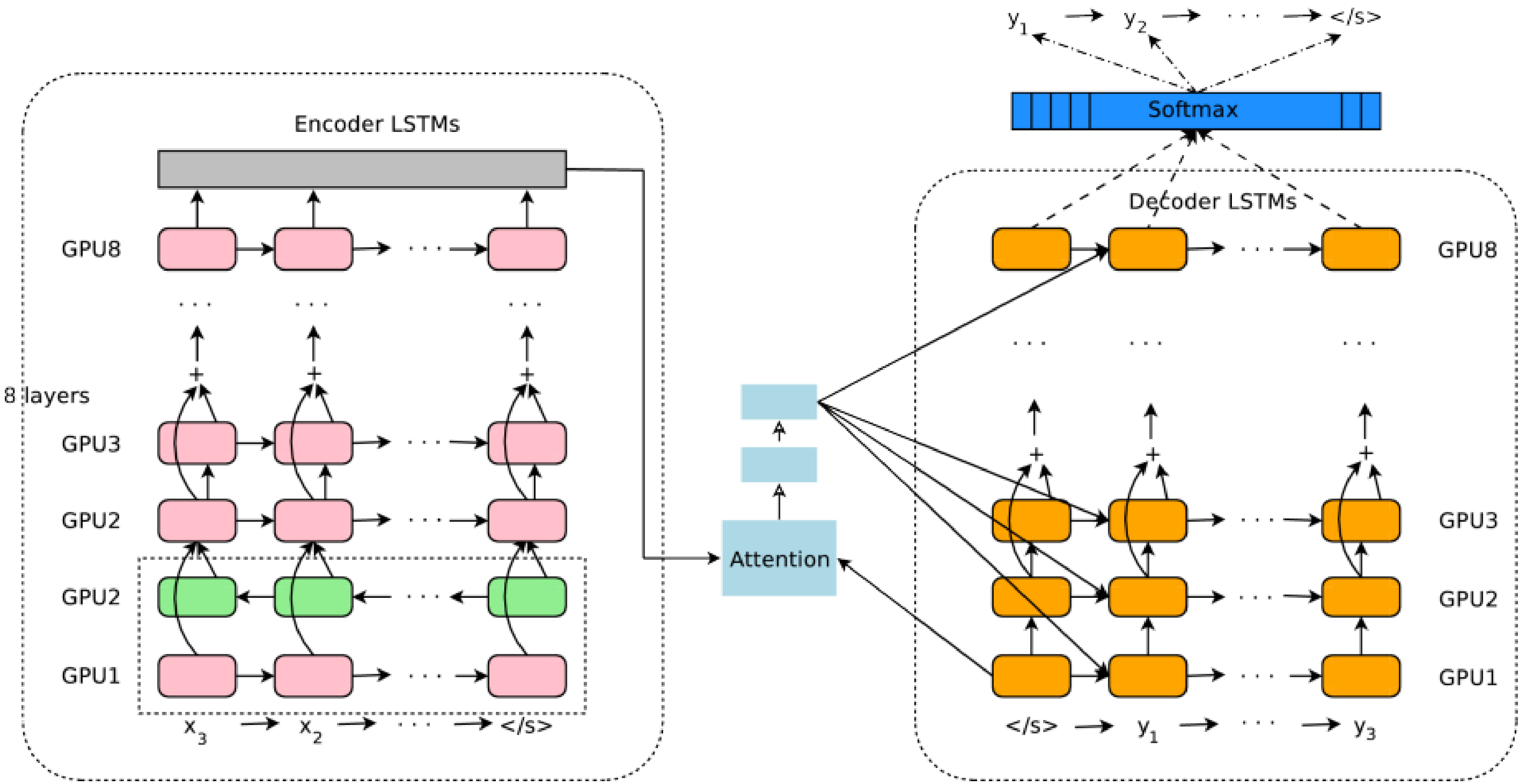
LSTM Encoder-Decoder Architecture



Tokenized input is **transformed to word vectors** by an embedding layer. Word vectors are input to an **LSTM encoder** that produces a dense vector representation of the input phrase.

An **LSTM decoder** transforms the vector generated by the encoder into a dense vector representing the translated phrase. A **softmax output layer** translates the vector into a **set of probabilities**. The word assigned to each position in the output sequence is the word in the vocabulary **assigned the highest probability**.

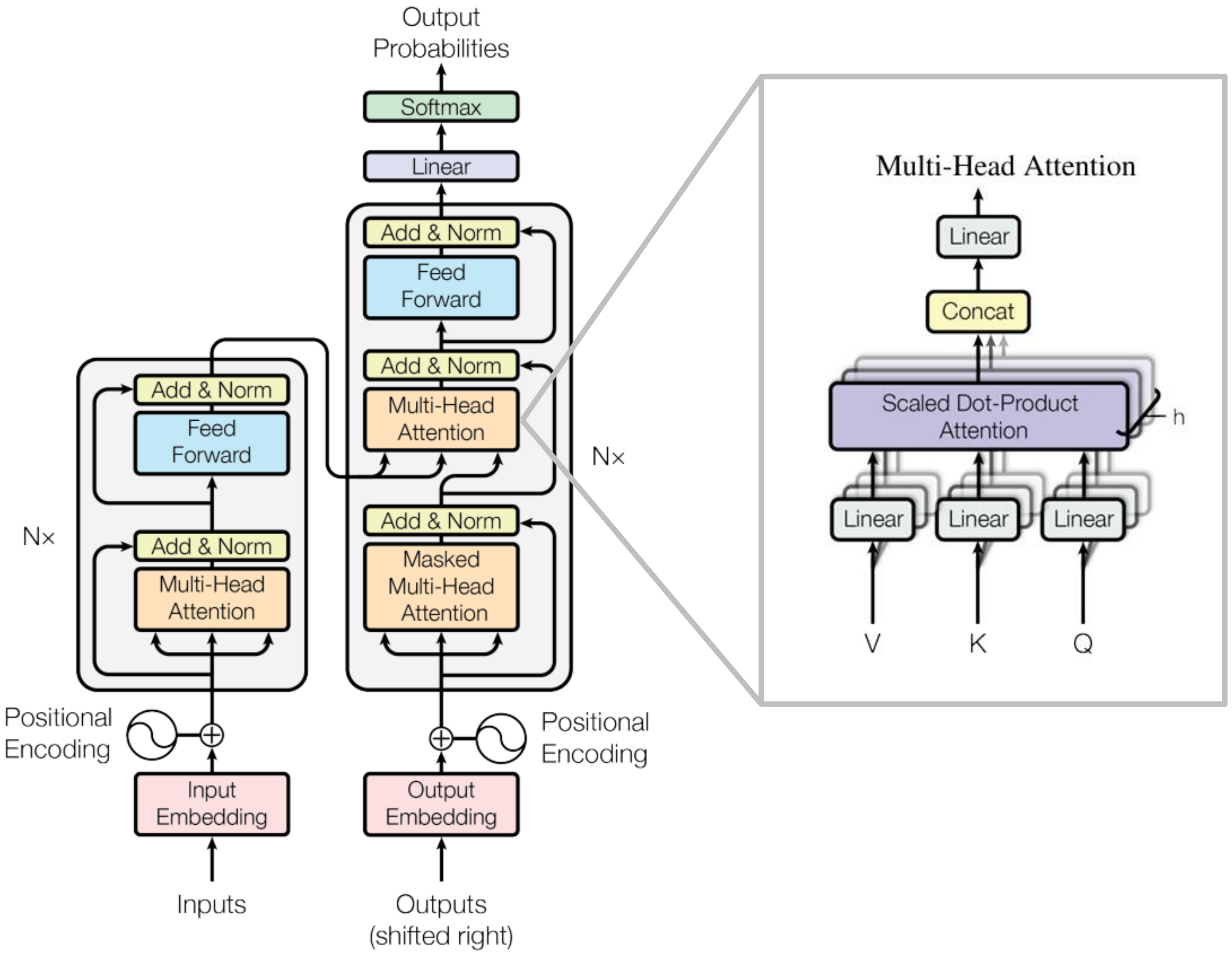
Google Translate circa 2016



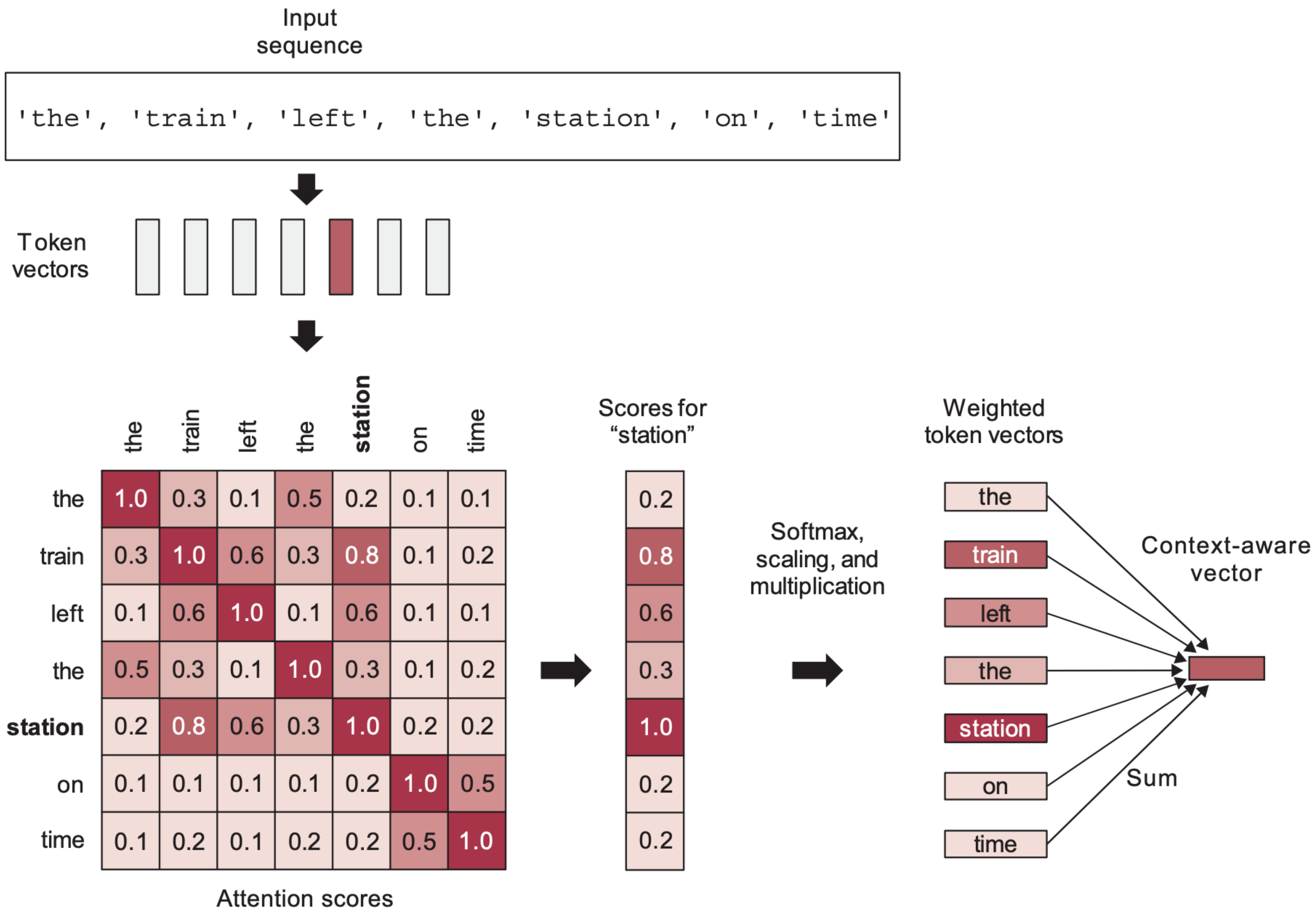
"Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation" (https://arxiv.org/abs/1609.08144)

Transformers

- Introduced in 2017 paper "Attention is All You Need"
- Multi-head attention layers extract meaning from text
  - Discern between different meanings of the same word (polysemy)
  - Connect pronouns to subjects
- Train in parallel and connect words independent of distance

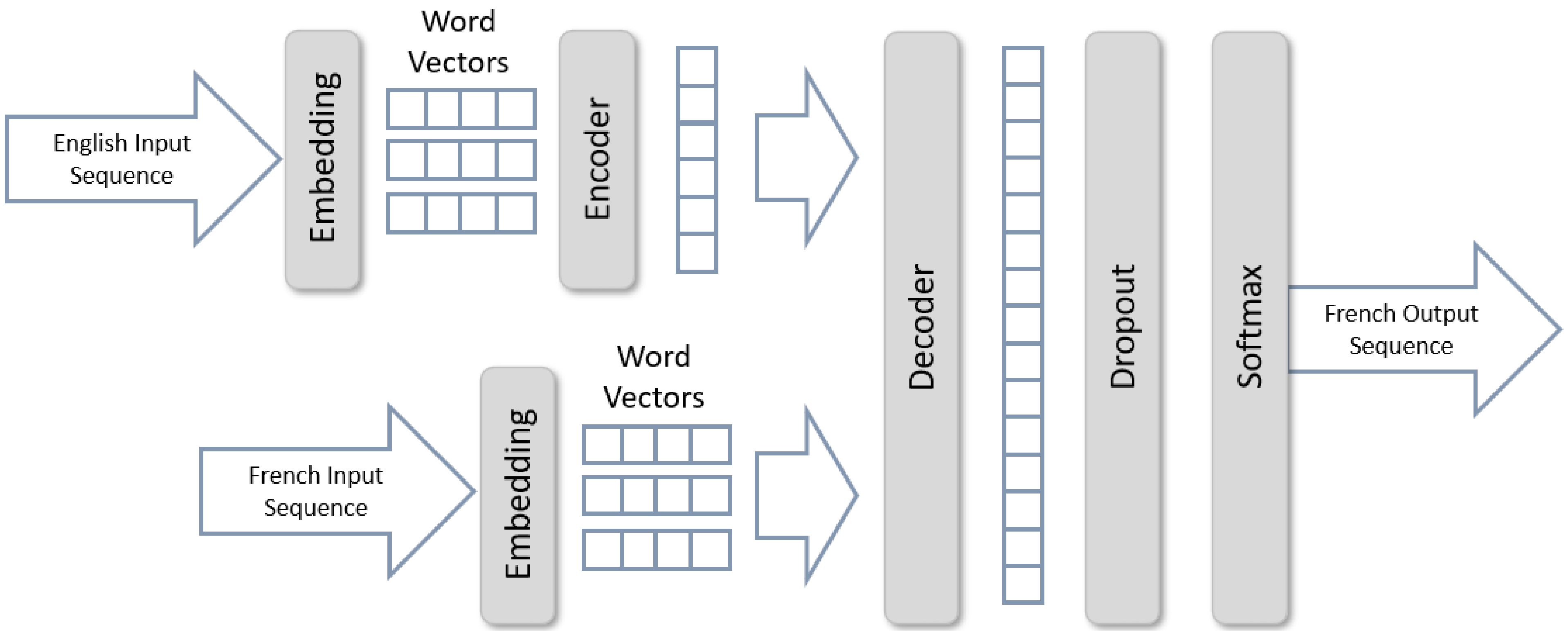


Self-Attention



Source: "Deep Learning with Python" by Francois Chollet (2021, Manning Publications)

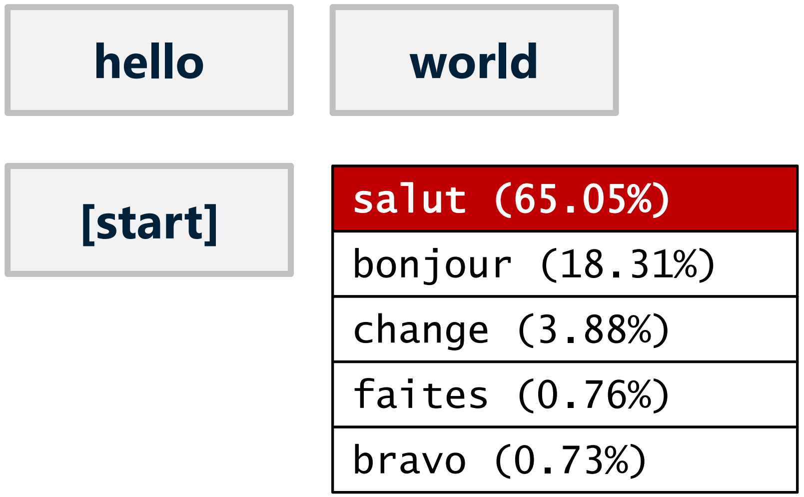
Transformer Encoder-Decoder Architecture



The model has **two inputs**: one for **tokenized English input**, and another for **tokenized French input**. Embedding layers translate each into word vectors.

The decoder translates the inputs into an output sequence, and a softmax output layer predicts the **next word in the French sequence**. The next word is appended to the text predicted thus far and **fed back into the model** to predict the **next word**. The cycle repeats until the **entire English phrase** has been translated.

Translating Text



Translating Text, Cont.

hello **world**

[start] **salut**

le (83.57%)
les (5.76%)
des (3.77%)
du (1.69%)
[end] (1.61%)

Translating Text, Cont.

hello world

[start] salut le

monde (98.47%)
ton (0.13%)
credule (0.08%)
bon (0.08%)
record (0.06%)

Translating Text, Cont.

hello world

[start] salut le monde

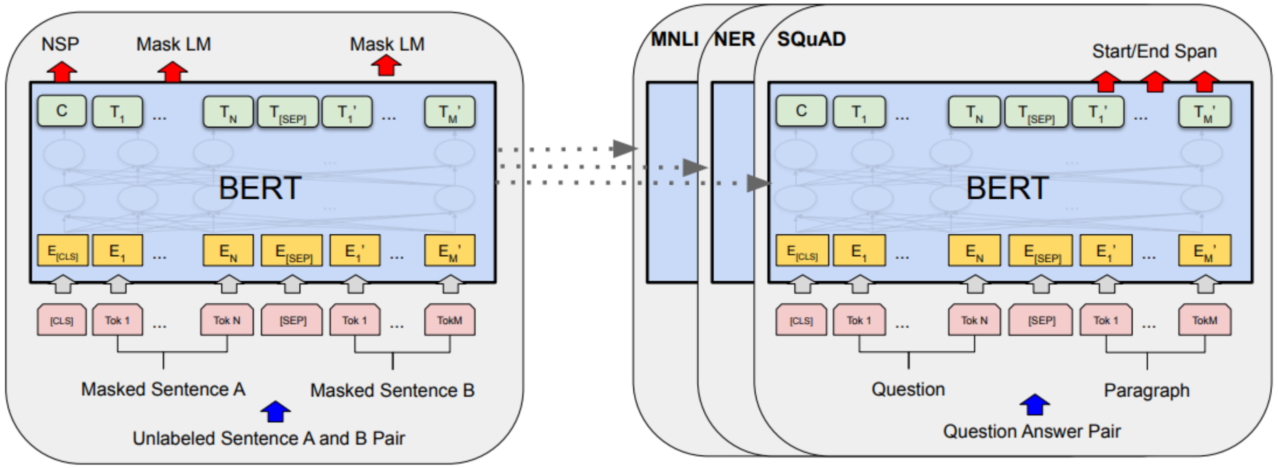
[end] (99.18%)
est (0.53%)
se (0.08%)
a (0.04%)
le (0.03%)

Demo

Neural Machine Translation

BERT

- Bidirectional Encoder Representations from Transformers (BERT)
- Built by Google and instilled with language understanding by pretraining with billions of words and phrases
- Can be fine-tuned to perform a variety of NLP tasks



Pre-training

Fine-Tuning

Masked Language Modeling (MLM)

- Turns unlabeled text into training ground for language structure
- During training, a specified percentage (usually 15%) of the tokens are randomly masked (dropped) from the training sequences, and the model is trained to predict the missing words

?	and	seven	years	?	our
fathers	?	forth	on	this	continent
a	new	?	conceived	in	liberty
and	dedicated	?	the	proposition	?

OpenAI GPT Models

- Available by REST API with an OpenAI account
- SDKs available for Python, Node.js, .NET, Java, Rust, and more

GPT-3.5 Turbo

Groundbreaking LLM with **175B parameters**. Fast and cost-effective. Also known as **ChatGPT**.

GPT-4o

Multimodal model ("o" for "omni") with **128K context window**. Knowledge cutoff of October 2023.

GPT-4o mini

Distilled version of GPT-4o with **128K context window**. Costs more than **30 times less than GPT-4o**.

GPT-4o1

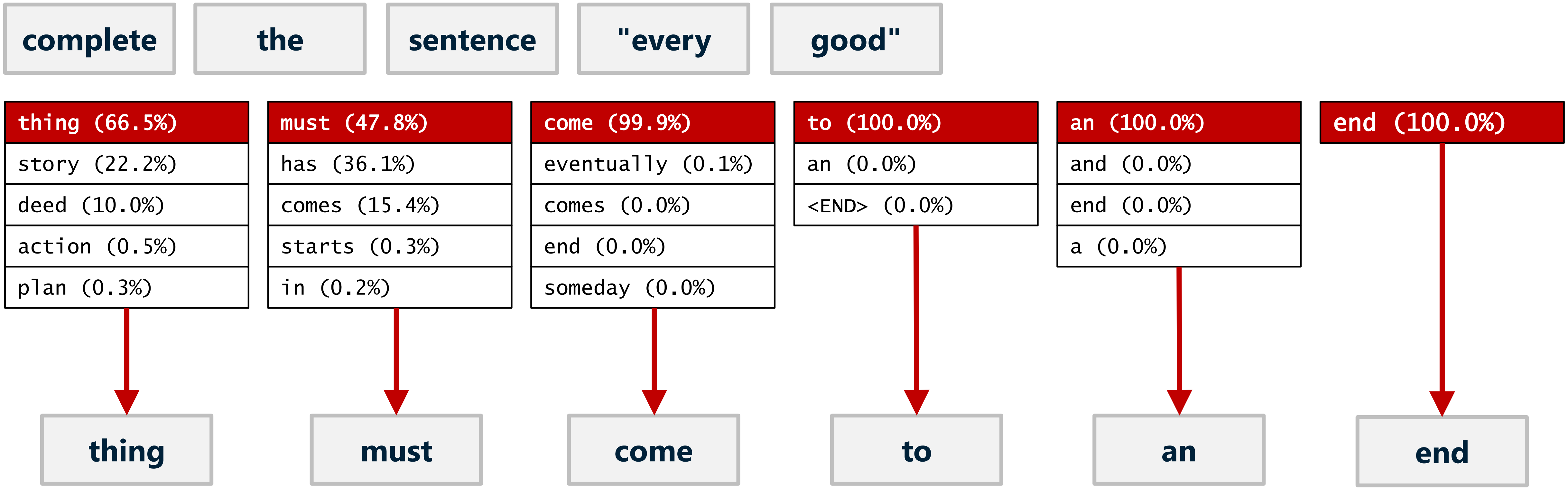
Latest GPT model capable of **complex problem solving**. Currently in preview.

GPT-4o1 mini

Distilled version of GPT-4o1. Currently in preview.

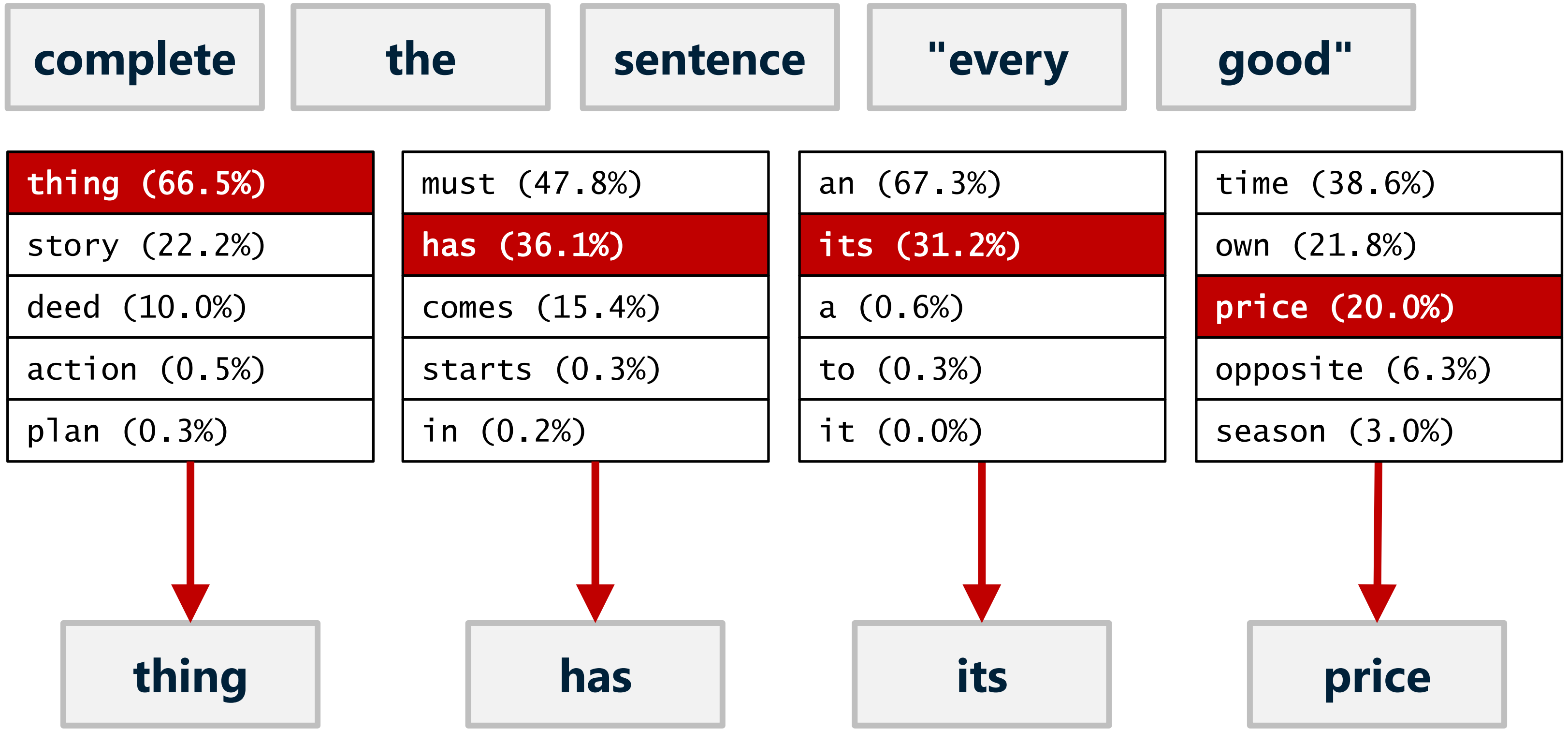
How GPT Models Work

`temperature=0.0`



How GPT Models Work, Cont.

`temperature=0.7`



Generating Text with GPT-4o

```python
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

```
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

```
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

```
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

```python
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)
```

Google Gemini

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

Gemini 1.5 Flash

Fastest, most cost-efficient model that accepts **text, images, audio, and video** as input. Features a **1 million token context window** and an estimated 20 billion parameters.

Gemini 1.5 Pro

Google's **most advanced multimodal LLM** with an estimated 120 billion parameters. Features a context window that accepts up to **2 million tokens**.

Generating Text with Gemini

```
import google.generativeai as genai

genai.configure(api_key='GOOGLE_API_KEY')
model = genai.GenerativeModel('gemini-1.5-flash')
response = model.generate_content('Why is the sky blue?')
print(response.text)
```

Streaming the Response

```
import google.generativeai as genai

genai.configure(api_key='GOOGLE_API_KEY')
model = genai.GenerativeModel('gemini-1.5-flash')
response = model.generate_content('Why is the sky blue?', stream=True)

for chunk in response:
    print(chunk, end='')
```

Specifying the Temperature

```python
import google.generativeai as genai

genai.configure(api_key='GOOGLE_API_KEY')
model = genai.GenerativeModel('gemini-1.5-flash')
config = genai.types.GenerationConfig(temperature=0.2)

response = model.generate_content(
    'Why is the sky blue?',
    generation_config=config
)

print(response.text)
```

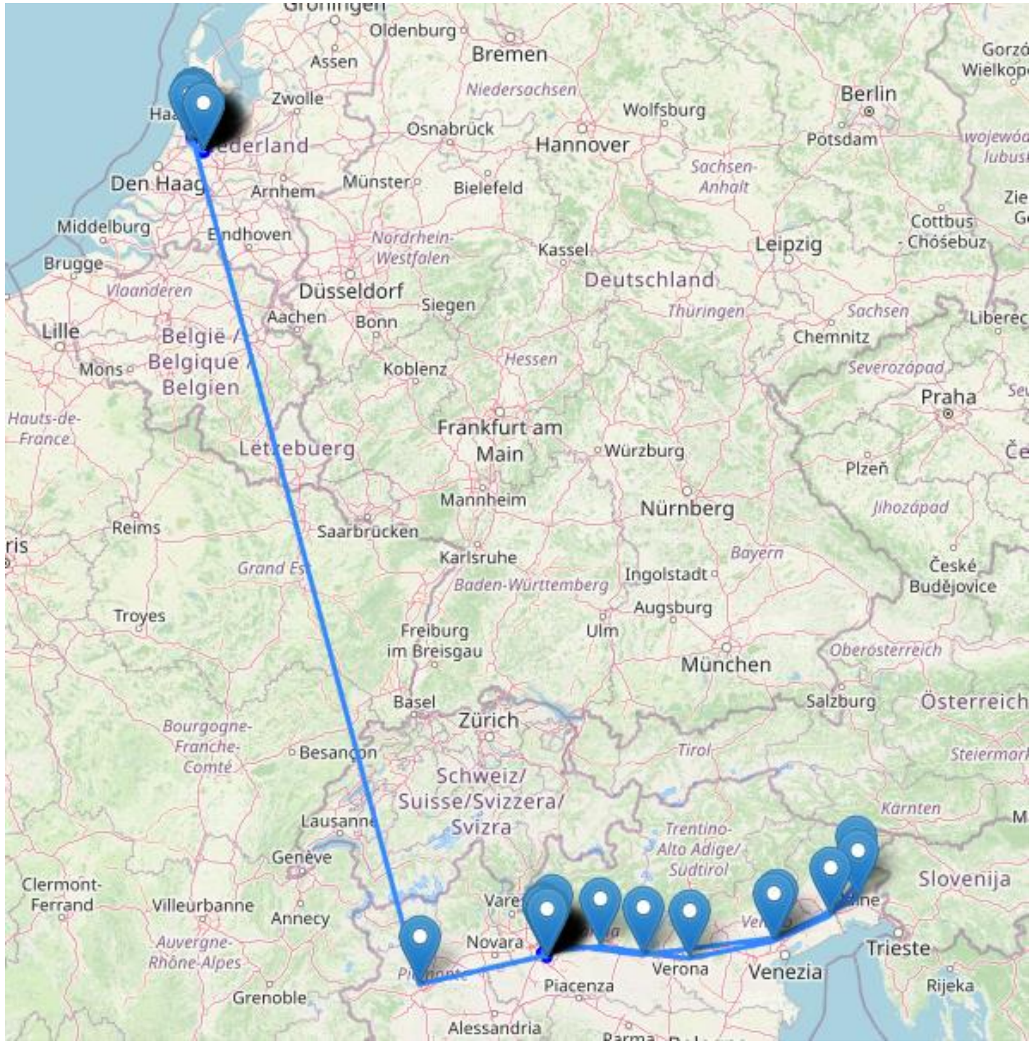
Generating JSON Output

```python
prompt = '''
    Which books comprise the Harry Potter series? Please respond in JSON
    using the following format for each book. Do not output markdown.
    {
        "title": "Harry Potter and the Prisoner of Azkaban",
        "year": 1999
    }
    '''

genai.configure(api_key='GOOGLE_API_KEY')
model = genai.GenerativeModel('gemini-1.5-flash')
response = model.generate_content(prompt)
print(response.text)
```

Demo

Text Generation

Multimodal LLMs (MLLMs)




```
Describe the shipment plotted on this map.
```

```
The shipment began in the Netherlands and made its way
to northeastern Italy using a combination of air and
ground transportation.

The initial segment went by air, originating from a
location near the coast of the Netherlands, possibly
near Rotterdam, and ending in northeastern Italy,
likely near Milan, as the shipment appears to fly
across the Netherlands, over Germany, and into Italy.

The subsequent legs of the journey were traveled by
road. The shipment passed through several cities in
northern Italy, eventually reaching its destination
near Venice.

The map does not specify exact highways, but the route
within Italy likely involved major roads such as the
A4 motorway, which connects Milan to Venice.
```

Submitting an Image to GPT-4o

```
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
)
```

Submitting Multiple Images to GPT-4o

```
messages = [{
    'role': 'user',
    'content': [
        { 'type': 'text', 'text': 'Describe similarities between these images' },
        { 'type': 'image_url', 'image_url': { 'url': 'IMAGE_URL_1' }},
        { 'type': 'image_url', 'image_url': { 'url': 'IMAGE_URL_2' }}
    ]
}]

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

Submitting an Image to Gemini

```
image = Image.open('IMAGE_PATH')
prompt = 'Describe what you see in this image'
response = model.generate_content([image, prompt])
```

Submitting Multiple Images to Gemini

```
image1 = Image.open('IMAGE_PATH_1')
image2 = Image.open('IMAGE_PATH_2')
prompt = 'Describe similarities between these images'
response = model.generate_content([image1, image2, prompt])
```

Demo

Multimodal LLMs

Small Language Models (SLMs)

- Language models with fewer parameters for faster inference and smaller footprint but with most of the power of foundation models
- Models are open-source ("open weights") and can be run locally

Phi-3

Available in **3.8B, 7B, and 14B** versions. Variants include **Phi-3 Vision**, which is multimodal.

Gemma 2

Available in **2B, 9B, and 27B** versions. Variants include **PaliGemma**, which is multimodal, and **CodeGemma**.

Llama 3

Based on Llama 3 foundation model. Available in **1B, 3B, 11B, and 90B** versions. 11B and 90B are multimodal.

Mistral

Available in **7B and 12B** versions. Variants include **Pixtral**, which is multimodal, and **Codestral Mamba**.

Generating Text with Phi-3

```python
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load the model and tokenizer
model_name = 'microsoft/Phi-3-mini-4k-instruct'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Tokenize the input
prompt = 'Why is the sky blue?'
tokens = tokenizer(prompt, return_tensors='pt')

# Generate the output
outputs = model.generate(**tokens, max_length=512)
print(tokenizer.decode(outputs[0]))
```

Generating Text with Gemma 2

```
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load the model and tokenizer
model_name = 'google/gemma-2-2b-it'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Tokenize the input
prompt = 'Why is the sky blue?'
tokens = tokenizer(prompt, return_tensors='pt')

# Generate the output
outputs = model.generate(**tokens, max_length=512)
print(tokenizer.decode(outputs[0]))
```

Streaming Text with Gemma 2

```
from transformers import AutoTokenizer, AutoModelForCausalLM, TextStreamer

# Load the model and tokenizer
model_name = 'google/gemma-2-2b-it'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Tokenize the input
prompt = 'Why is the sky blue?'
tokens = tokenizer(prompt, return_tensors='pt')

# Stream the output
streamer = TextStreamer(tokenizer, skip_prompt=True) # Writes to stdout
_ = model.generate(**tokens, max_length=512, streamer=streamer)
```

Demo

Small Language Models

Code Generation

- Flagship LLMs such as GPT-4o, Gemini 1.5, and Llama 3 are adept at generating code in 40+ languages
  - Write code (convert natural language into code) and unit tests
  - Comment and explain code (convert code into natural language)
  - Convert code from one programming language to another
  - Complete code, edit/refactor code, and find bugs
- SLMs optimized for code generation are available, too
  - Code Llama, available in 7B, 13B, 34B, and 70B parameter versions
  - CodeGemma, available in 2B and 7B parameter versions

Generating Code

```python
content = '''
    Create a Python function that accepts an array of numbers as
    input, bubble sorts the numbers, and returns a sorted array
    '''

messages = [{ 'role': 'user', 'content': content }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages,
    temperature=0 # Use a low temperature setting for code generation
)
```

Converting Code to Natural Language

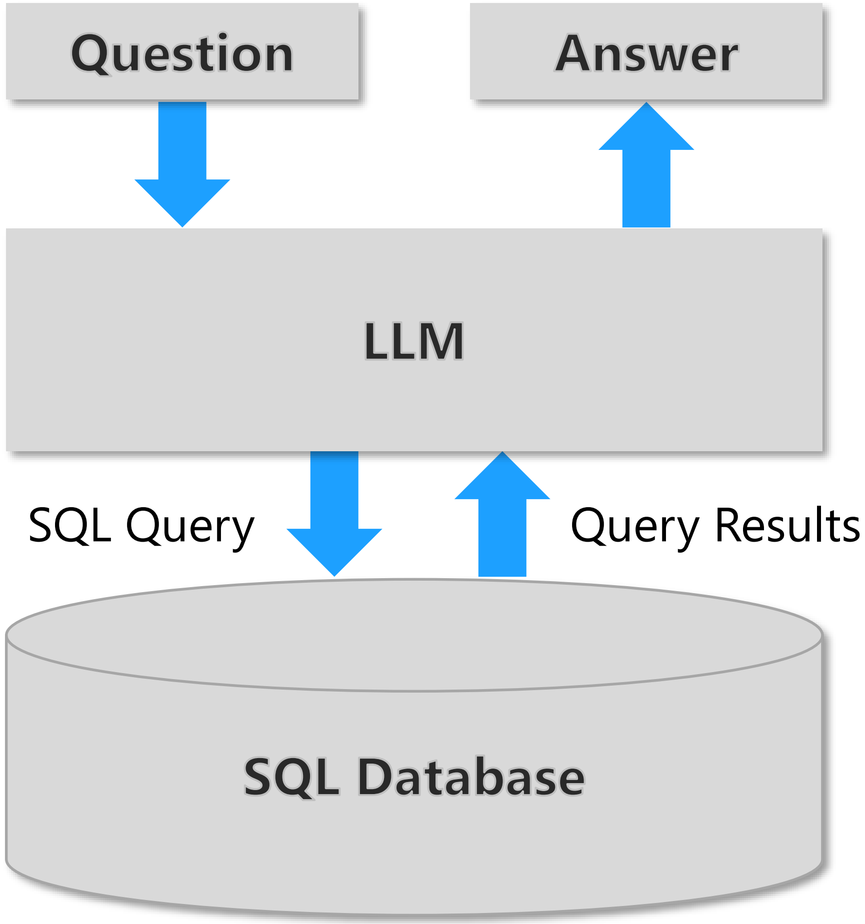
```
content = '''
    Explain what the following code does:

    def bubble_sort(arr):
        n = len(arr)
        for i in range(n):
            for j in range(0, n-i-1):
                if arr[j] > arr[j+1]:
                    arr[j], arr[j+1] = arr[j+1], arr[j]
        return arr
    '''

messages = [{ 'role': 'user', 'content' : content }]

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

Translating Code

```
content = '''
    Convert the following Python code into FORTRAN:

    def bubble_sort(arr):
        n = len(arr)
        for i in range(n):
            for j in range(0, n-i-1):
                if arr[j] > arr[j+1]:
                    arr[j], arr[j+1] = arr[j+1], arr[j]
        return arr
    '''

messages = [{ 'role': 'user', 'content' : content }]

response = client.chat.completions.create(
    model='gpt-4o',
    messages=messages,
    temperature=0
)
```

SQL-Augmented Generation (SAG)

- Use LLM to implement a natural-language interface to databases
  - Convert natural-language questions into SQL queries
  - Execute the queries against the database
  - Use LLM to phrase query results
- Use CREATE TABLE statements in prompts to define schemas
- Optionally provide sample questions and resultant SQL queries in prompts



Generating a SQL Query

```
prompt = f'''
    Assume the database has a table that is defined as follows:
    {table_definition}
    Generate a well-formed SQL query from the following prompt:
    {question}
    '''

response = client.chat.completions.create(
    model='gpt-4o',
    messages=[{ 'role': 'user', 'content': prompt }],
    temperature=0
)
```

Generating a Natural-Language Response

```
prompt = f'''
   Given the following question and query result, phrase the answer
   in terms that a human can understand.
   Question: {question}
   Result: {result}
   '''

response = client.chat.completions.create(
    model='gpt-4o',
    messages=[{ 'role': 'user', 'content': prompt }]
)
```

Demo

LLMs Over Databases

Answering Questions

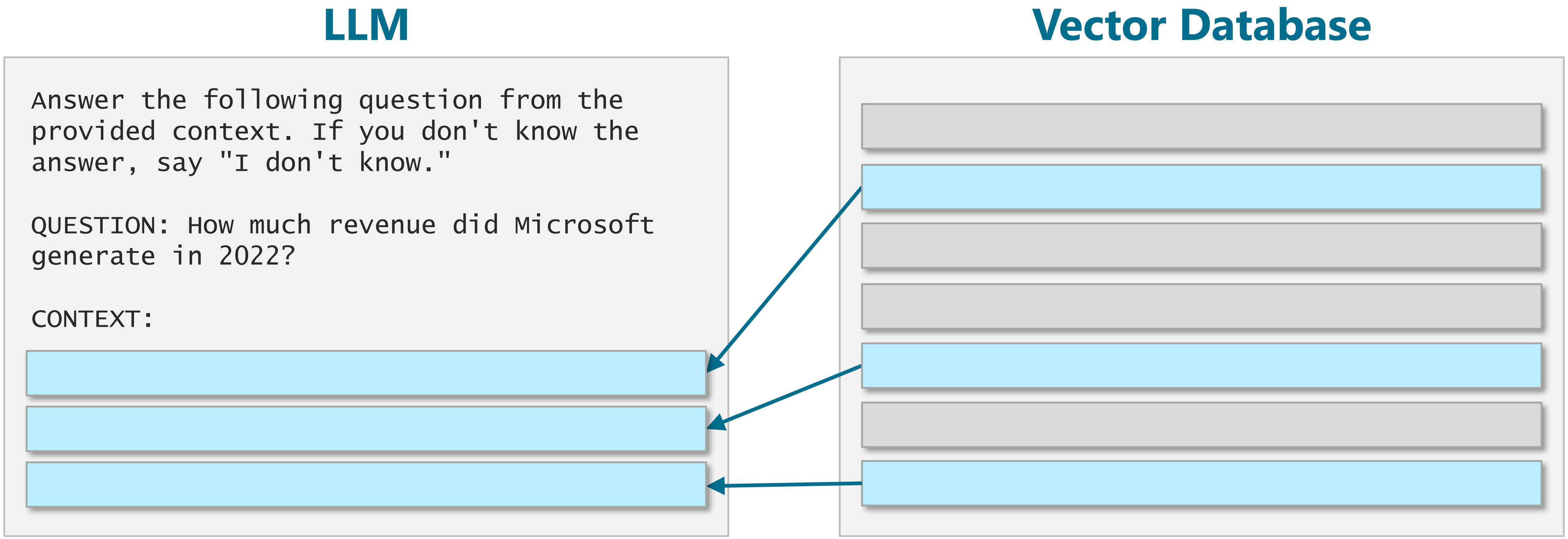
```
content = f'''
    Answer the following question, and if you don't know the answer, say "I don't know:"
    Question: Who was the first president of the United States?
    Answer:
    '''

messages = [{ 'role': 'user', 'content': content }]

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

Answering Contextual Questions

```
content = f'''
    Answer the following question using the provided context, and if the answer isn't
    contained in the context, say "I don't know:"
    Context: {context} # Insert text to be searched for an answer
    Question: Who was the first president of the United States?
    Answer:
    '''

messages = [{ 'role': 'user', 'content': content }]

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

Retrieval-Augmented Generation (RAG)

- Use LLM to answer questions using documents as context
- "Chunk" documents and use embeddings to identify relevant chunks



OpenAI Embeddings API

- Generates high-quality embedding vectors from text
  - `text-embedding-3-small` model is best and most cost-effective
  - Generates vectors of 1,536 floating-point numbers and is applicable to long and short text samples, replacing earlier models that were sensitive to text length
- Compute similarity between two text samples by generating embeddings for each and computing the dot product
  - Useful for semantic-search systems, recommender systems, deduplication systems, and other similarity-based systems
- Combine with vector databases such as **Pinecone** or **ChromaDB** to create systems that scale to millions of embedding vectors

Generating an Embedding Vector

```python
from openai import OpenAI

client = OpenAI(api_key='OPENAI_API_KEY')

response = client.embeddings.create(
    model='text-embedding-3-small',
    input='Four score and seven years ago our fathers brought forth, ' \
          'upon this continent, a new nation, conceived in liberty, and ' \
          'dedicated to the proposition that all men are created equal'
)

embedding = response.data[0].embedding

# [0.018447233363986015, 0.041993703693151474, 0.026396041736006737, ...]
```

Comparing Embedding Vectors

```
x = client.embeddings.create(
    model='text-embedding-3-small',
    input='What was the average MKT for shipments that went through LHR?'
).data[0].embedding

y = client.embeddings.create(
    model='text-embedding-3-small',
    input='Average Mean Kinetic Temperature for shipments via Heathrow'
).data[0].embedding

similarity = np.dot(np.array(x), np.array(y))
# 0.864652407906532
```

Do LLMs Represent AGI?

Grady Booch
@Grady_Booch

LLMs do not think or reason.

LLMs interpolate within a high dimensional latent space. They are undeniably capable of generating remarkably coherent results (which is their strength) but do not in any sense of the word understand (which is their weakness). As such, they are inherently biased by the nature of their training corpus and when they attempt to extrapolate beyond their training, they tend to confabulate.

Human thinking and human understanding are not mere statistical processes as are LLMs and to assert that they are represents a profound misunderstanding of the exquisite uniqueness of human cognition. To paraphrase Sagan, the brain does much more than predict. It generates abstractions; it builds theories of the world, of others, and of itself; it has agency that is tempered by a complex interaction of its subjective experience, its goals, its desires, and its feelings.

I have reason to believe that the mind is computable, that sentience and sapience are beautiful consequences of the laws of physics. Someday, we may understand how to build a mind.

But this is not the day, and likely not even the generation.

LLMs are not the path. They are a small part of the journey.