Laboratório de Engenharia de Sistemas e Robótica

UFPB

Modelos de Linguagem Larga (LLM)

1. Introdução

Prof.: Alisson Brito (alisson@ci.ufpb.br)

Laboratório de Engenharia de Sistemas e Robótica

UFPB

Objetivo da Disciplina

- Reconhecer a evolução dos modelos de linguagem.
- Analisar arquitetura Transformer e suas variantes.
- Aplicar técnicas de pré-treinamento e fine-tuning.
- Utilizar estratégias de engenharia de prompt.
- Desenvolver aplicações práticas com LLMs.
- Identificar riscos, limitações e aspectos éticos.

Avaliação

- Relatórios teóricos e práticos (60%)
- Participação nas atividades
- Desenvolvimento e apresentação de projeto (40%)
  - Proposta e desenvolvimento de uma aplicação usando LLMs
  - Apresentação dos resultados/comparativos.

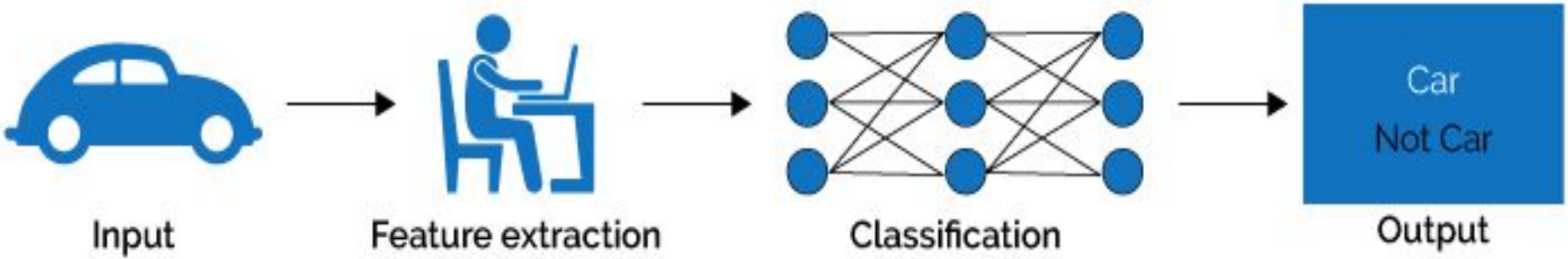
Agenda

- Introdução aos Modelos de Linguagem
- Funcionamento dos Modelos de Linguagem
- História e evolução
- Embeddings e representações vetoriais
- Introdução aos LLMs.
- Prática: Exploração de um LLM via API

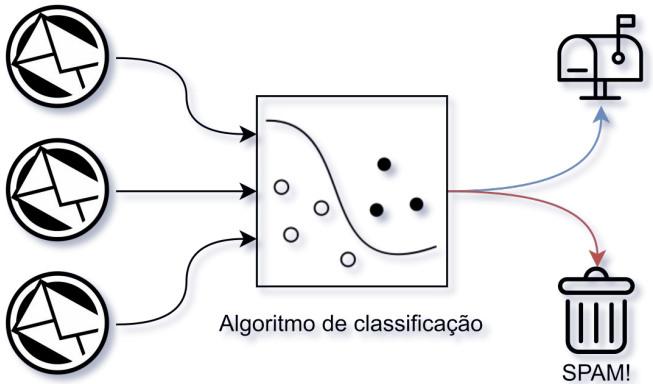
Inteligência Artificial

Sistemas capazes de executar tarefas que normalmente exigiria inteligência humana para tomar decisões

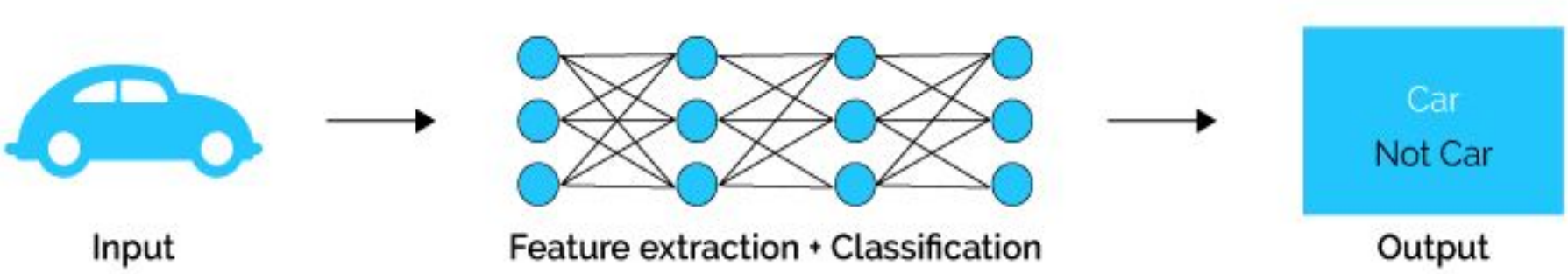
Machine Learning



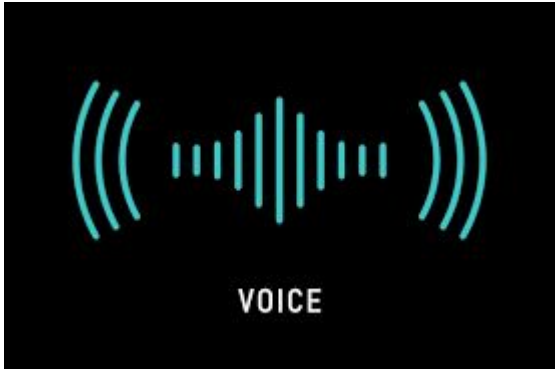
Algoritmos capazes de aprender padrões a partir de dados



Deep Learning



Usa redes neurais artificiais com muitas camadas treinadas para propósitos específicos (IA generativa, reconhecimento de voz, reconhecimento de imagem etc.)



Inteligência Artificial Generativa

criar conteúdo
(texto, imagem, áudio, vídeo)
a partir de padrões aprendidos
nos dados

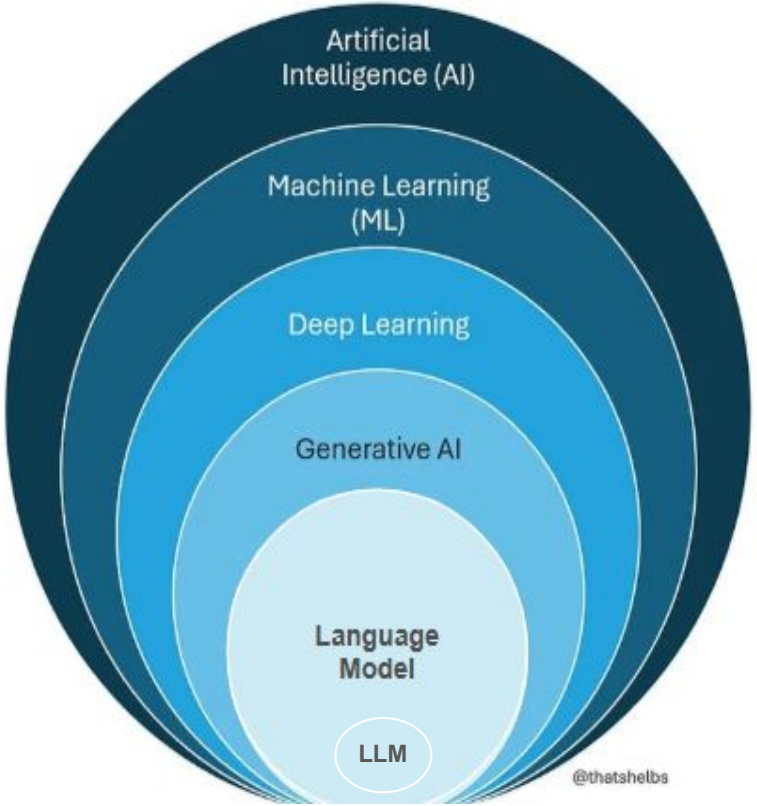
Modelos de Linguagem

LLMs

usam aprendizado profundo e estimam a probabilidade de uma sequência de palavras.

modelo de linguagem em larga escala, treinado com enormes volumes de dados para gerar texto

Uma visão geral



IA: sistemas capazes de executar tarefas que normalmente exigiria inteligência humana para tomar decisões

ML: algoritmos capazes de aprender padrões a partir de dados

DL: usa redes neurais artificiais com muitas camadas

Generative AI: criar conteúdo(texto, imagem, áudio, vídeo) a partir de padrões aprendidos nos dados

LM: usam aprendizado profundo e estimam a probabilidade de uma sequência de palavras.

LLM: modelo de linguagem em larga escala, treinado com enormes volumes de dados para gerar texto

Introdução aos Modelos de Linguagem

Modelos de Linguagem: Funcionamento

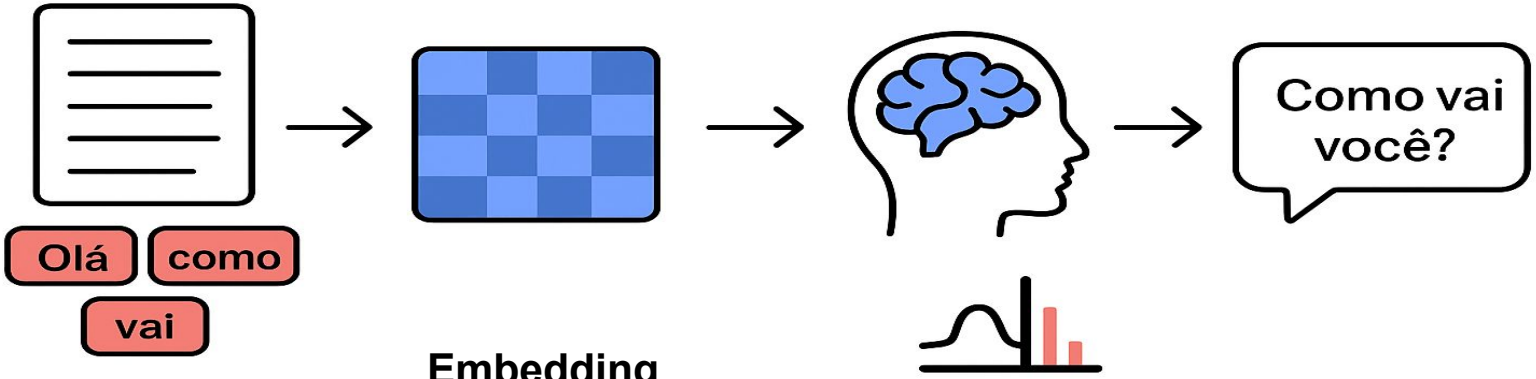
1. **Tokenização**: O texto de entrada é dividido em unidades menores chamadas tokens (palavras, subpalavras ou caracteres).

2. **Representação (Embedding)**: Cada token é representado em um formato numérico que o computador pode processar, como vetores.

3. **Aprendizado**: O modelo aprende a probabilidade de ocorrência de diferentes sequências de tokens com base em exemplos de treinamento.

4. **Geração**: Ao receber um texto de entrada, o modelo prevê a próxima palavra ou sequência de palavras, gerando texto coerente.

Modelos de Linguagem: Funcionamento



Tokenização

O texto de entrada é dividido em unidades menores chamadas tokens (palavras, subpalavras ou caracteres)

Embedding Representação

Cada token é representado em um formato numérico que o computador pode processar, como vetores

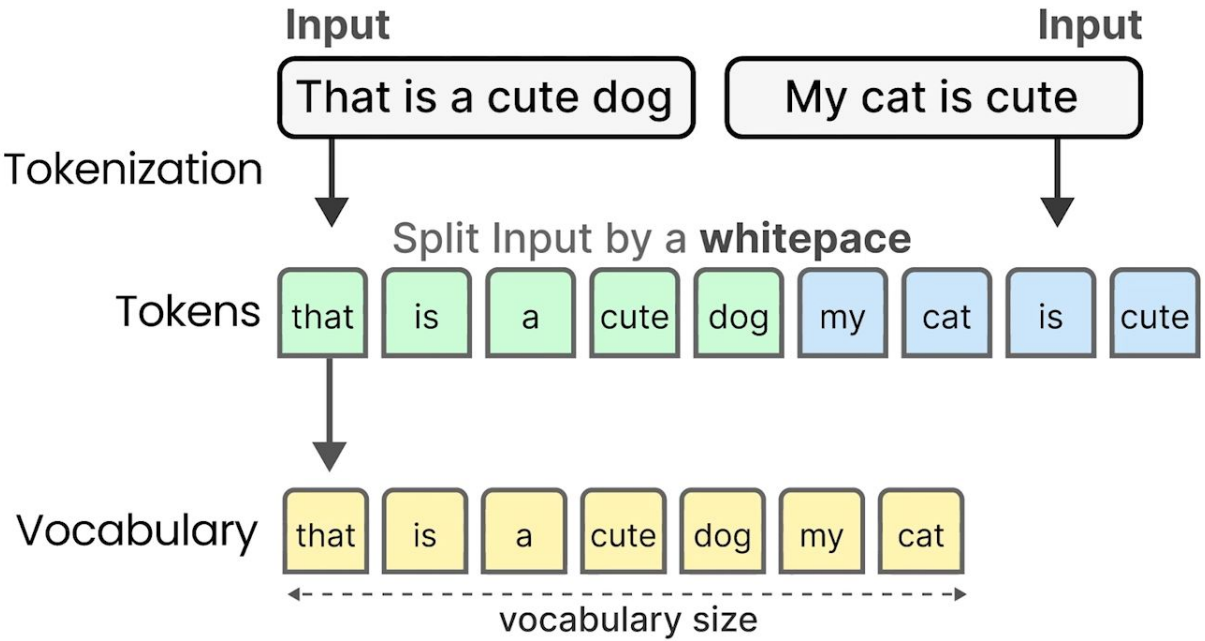
Aprendizado

O modelo aprende a probabilidade de ocorrência de diferentes sequências de tokens com base em exemplos de treinamento

Geração

Ao receber um texto de entrada, o modelo prevê a próxima palavra ou sequência de palavras, gerando texto coerenté

Tokenização



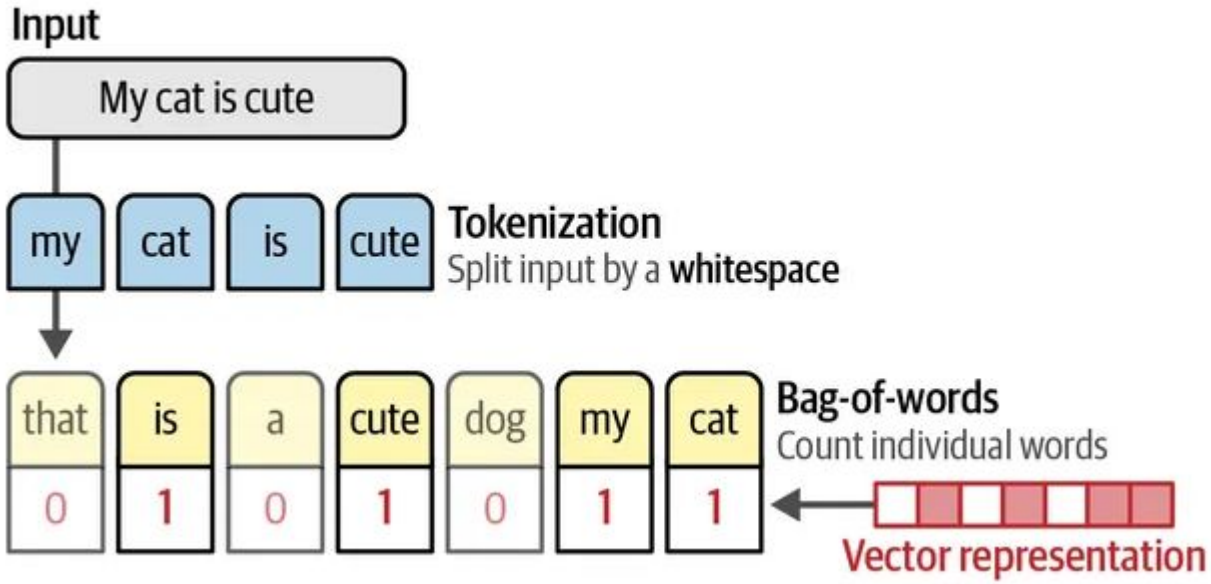
Tokenização

```
!pip install -U spacy
!python -m spacy download pt_core_news_sm

import spacy
import pandas as pd
# Carregando o modelo de idioma em português
nlp = spacy.load('pt_core_news_sm')
# Texto de exemplo
texto = "My cat is cut"
# Aplicando a tokenização
doc = nlp(texto)
# Extraindo os tokens
tokens = [token.text for token in doc]
print("Tokens:", tokens)
```

```
SAíDA:
Tokens: ['My', 'cat', 'is', 'cut']
```

Tokenização



Representação

Atribuem probabilidades a palavras ou sequências de palavras:

1. Probabilidade de uma palavra: P(w1)
2. Probabilidade de uma sequência de palavras: P(w1, w2, w3, ..., wn)
3. Probabilidade da próxima palavra: P(wn|w1, w2, ..., wn-1)

Representação

1. Probabilidade de **uma palavra** P(w1)
   Indica a probabilidade de “Modelos” aparecer isoladamente

   ```
   P(Modelos)
   ```

2. Probabilidade de uma **sequência de palavras** P(w1, w2, w3, ..., wn)
   Para a sequência completa: “Modelos de linguagem geram texto”

   ```
   w1=Modelos
   w2=de
   w3=linguagem
   w4=geram
   w5=texto
   ```

Modelos de Linguagem: Representação

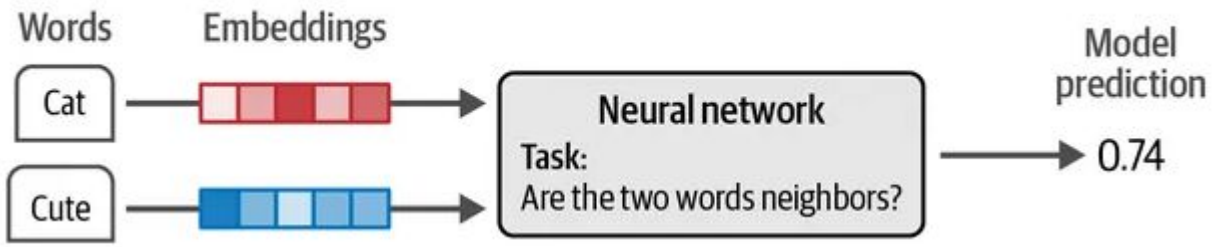
3. Probabilidade da próxima palavra P(wn|w1, w2, ..., wn-1)
   Próxima palavra após “Modelos de linguagem geram”:

P(texto | Modelos de linguagem geram)→ alta
P(imagens | Modelos de linguagem geram)→ média
P(carros | Modelos de linguagem geram)→ baixa

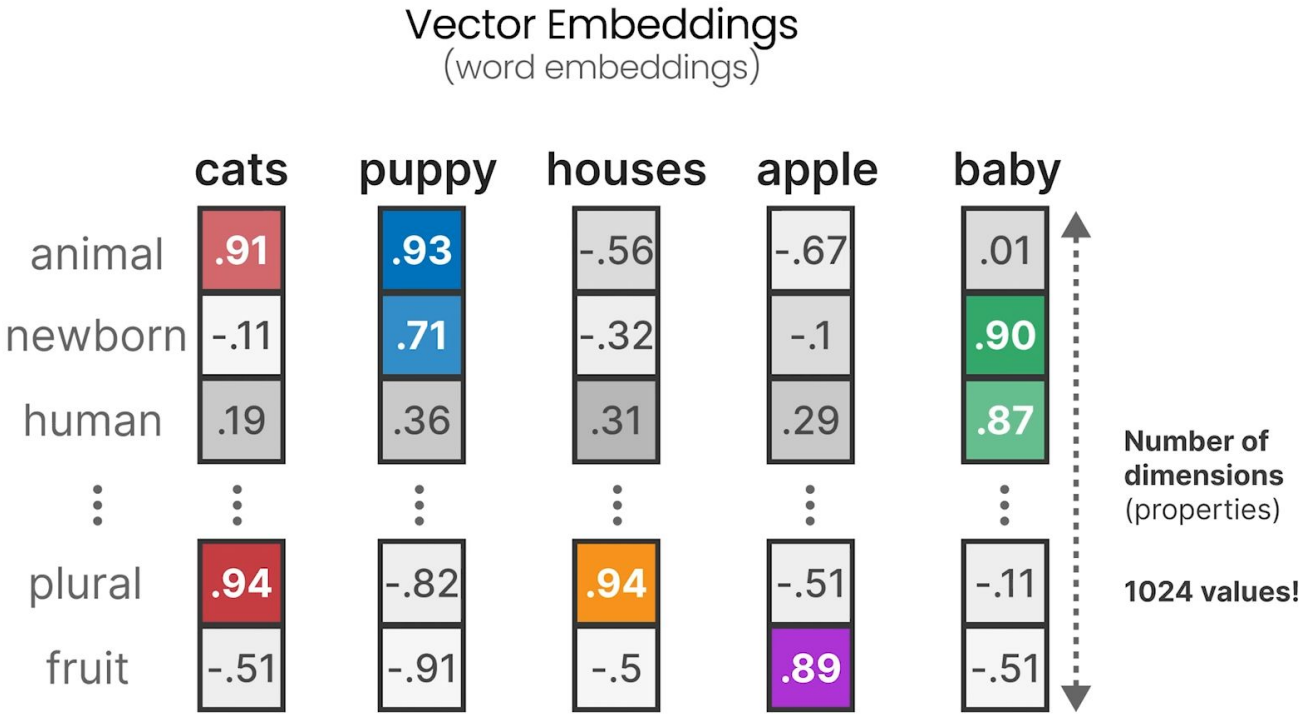
Embeddings

Um embedding é uma representação vetorial de um dado, geralmente usado para palavras ou tokens, pois:

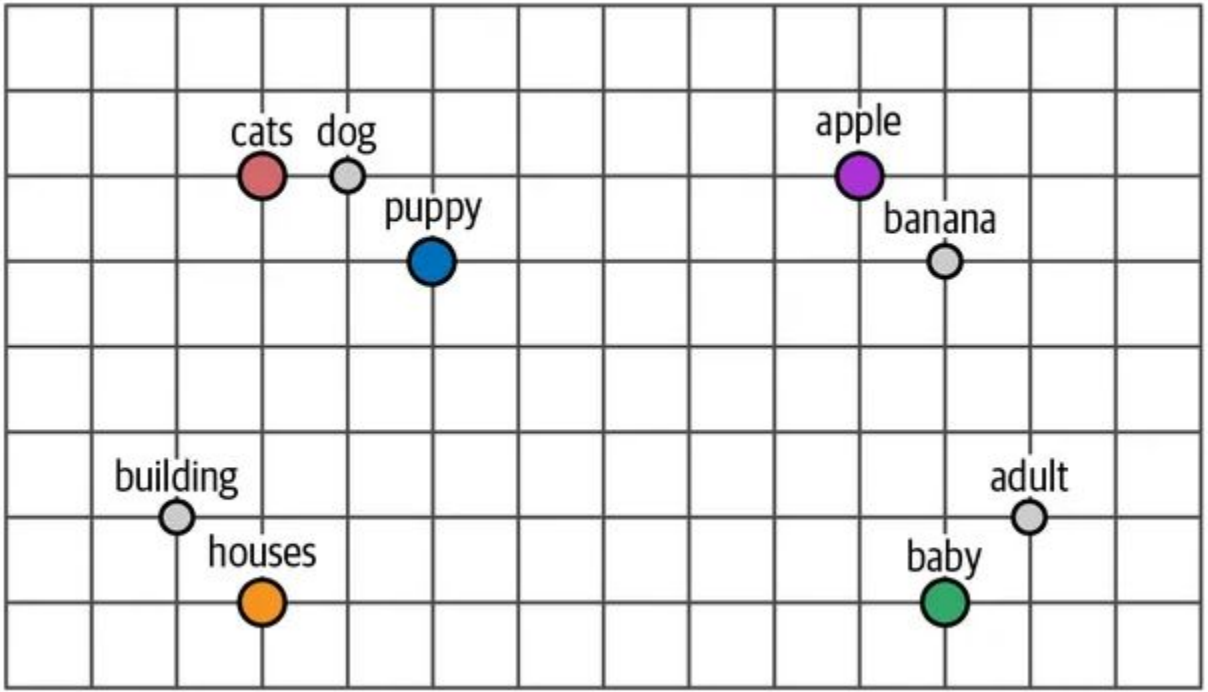
- Palavras ou tokens são categorias discretas
- Computadores não conseguem usar essas categorias diretamente em redes neurais, então transformamos cada palavra em um vetor de números reais.
- Esse vetor é chamado de embedding, que normalmente usa o modelo Word2Vec



Embeddings


Vector Embeddings
(word embeddings)
cats
puppy
houses
apple
baby
animal
newborn
human
plural
fruit
.91
-.11
.19
.94
-.51
.93
.71
.36
-.82
-.91
-.56
-.32
.31
.94
-.5
-.67
-.1
.29
-.51
.89
.01
.90
.87
-.11
-.51
Number of dimensions
(properties)
1024 values!

Embeddings



Embeddings

```
!pip install gensim #biblioteca especializada em embeddings de palavras

import gensim.downloader as api
from scipy.spatial.distance import cosine
# Carregar o modelo pré-treinado Word2Vec
model = api.load("word2vec-google-news-300")
words = ["cats", "dog", "puppy", "banana", "apple", "adult", "baby"]
embeddings = {word: model[word] for word in words}
print("Embeddings:")
for word, embedding in embeddings.items():
    print(f"{word}: {embedding[:1]}...")
```

saída

```
Embeddings:
cats: [-0.0703125]...
dog: [0.05126953]...
puppy: [0.07177734]...
banana: [-0.08544922]...
apple: [-0.06445312]...
adult: [-0.04589844]...
baby: [0.03393555]...
```

Embeddings

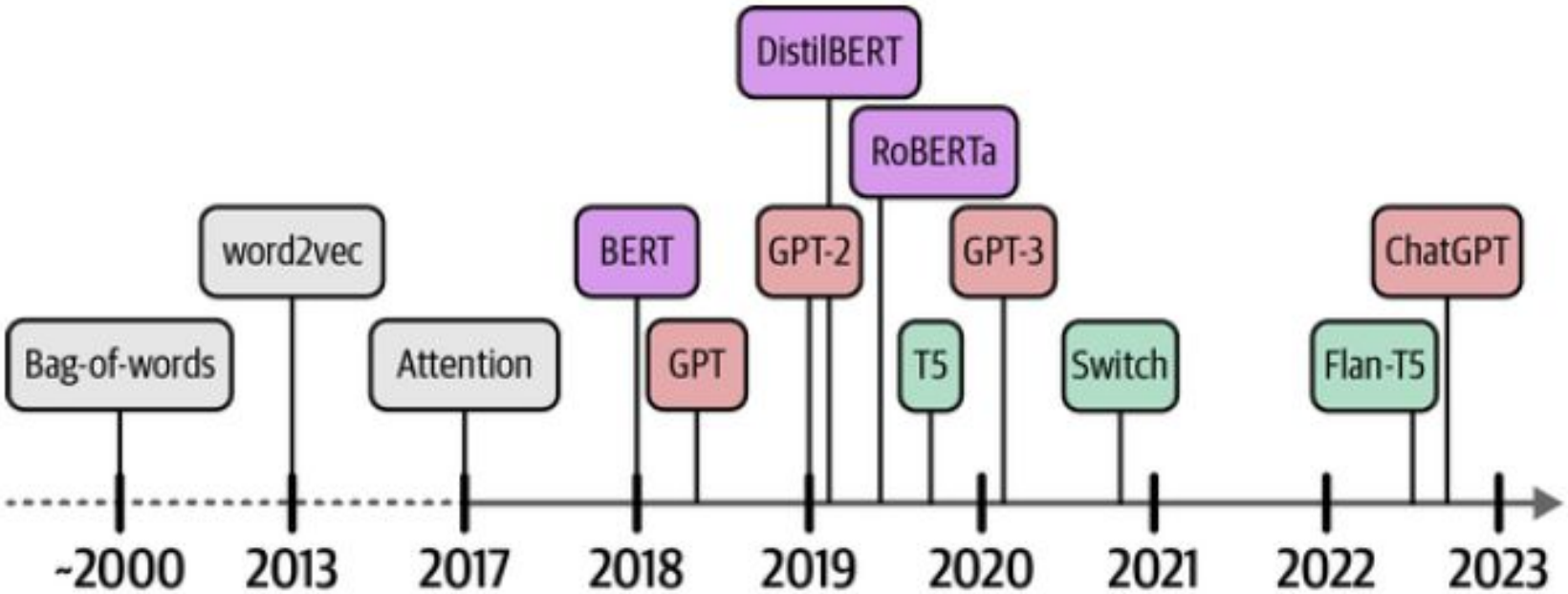
```
!pip install gensim #biblioteca especializada em embeddings de palavras


# Calcular e exibir similaridades automaticamente
print("\nSimilaridades semânticas entre palavras:")
for i, word1 in enumerate(words):
    for j, word2 in enumerate(words):
        if i < j:  # evita repetir pares
            similarity = 1 - cosine(embeddings[word1], embeddings[word2])
            print(f"{word1} ↔ {word2}: {similarity:.2f}")
```

saída

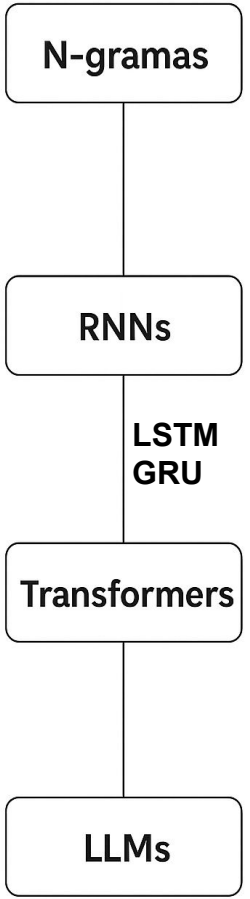
```
Similaridades semânticas entre palavras:
cats ↔ dog: 0.67
cats ↔ puppy: 0.62
cats ↔ banana: 0.09
cats ↔ apple: 0.12
cats ↔ adult: 0.25
cats ↔ baby: 0.35
dog ↔ puppy: 0.81
dog ↔ banana: 0.19
dog ↔ apple: 0.22
dog ↔ adult: 0.24
dog ↔ baby: 0.36
```

História dos Modelos de Linguagem



Evolução da arquitetura: de RNNs a Transformers

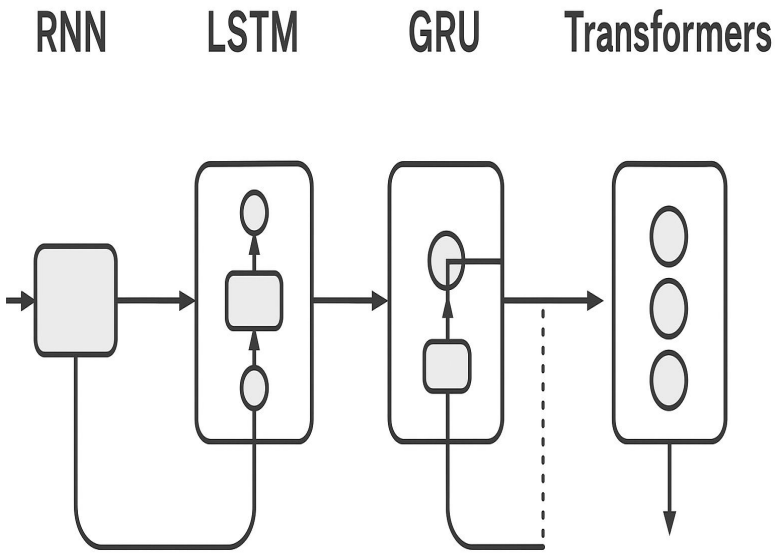
- **N-gramas**: Abordagem estatística simples.
- **RNNs** (Recurrent Neural Network): modelo de aprendizado profundo treinado para processar e converter uma entrada de dados sequencial (palavras, frases ou dados de séries temporais) em uma saída de dados sequencial específica.
- **LSTM / GRU**: versões aprimoradas das RNNs.
- **Transformers:** Arquitetura de rede neural projetada para processar dados sequenciais sem depender de conexões recorrentes ou convolucionais.
- **LLMs:** Um modelo de linguagem treinado com grandes volumes de dados textuais, possuindo grande quantidade de parâmetros, e implementado quase sempre com arquiteturas baseadas em Transformers.



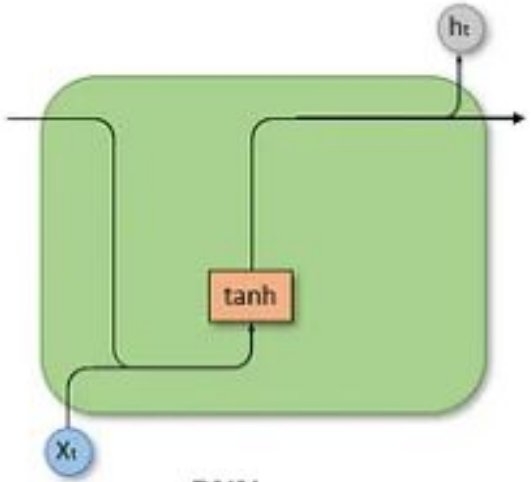
Evolução da arquitetura: de RNNs a Transformers

- **LSTM** (Long Short-Term Memory): evolução da RNN que consegue memorizar informações por mais tempo. Dividem o problema de gerenciamento de contexto em dois subproblemas: remover informações que não são mais necessárias do contexto e adicionar informações que provavelmente serão necessárias para a tomada de decisões posteriores.

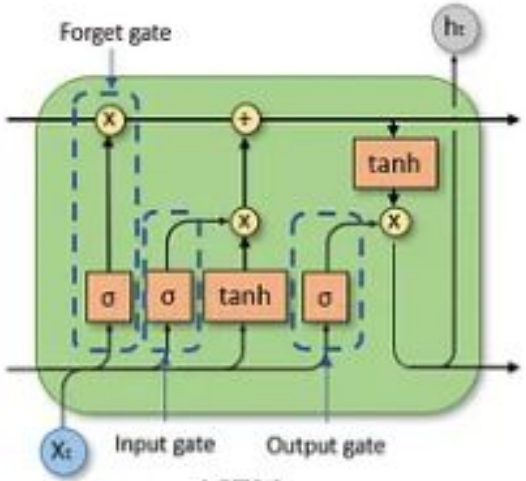
- **GRU** (Gated Recurrent Unit): Variante simplificada do LSTM também projetada para lidar com dependências de longo prazo em sequências, sendo mais leve, simples e rápido



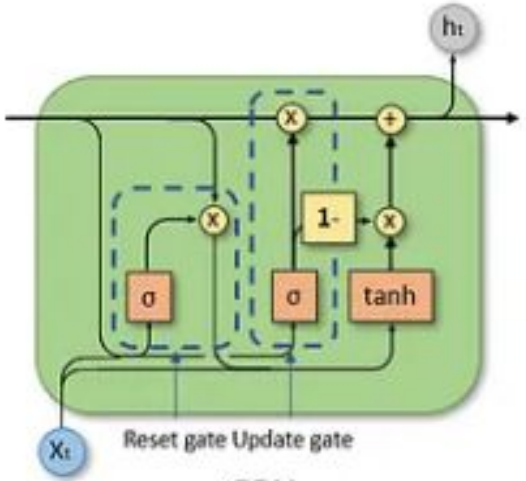
Evolução da arquitetura: de RNNs a Transformers



RNN



LSTM



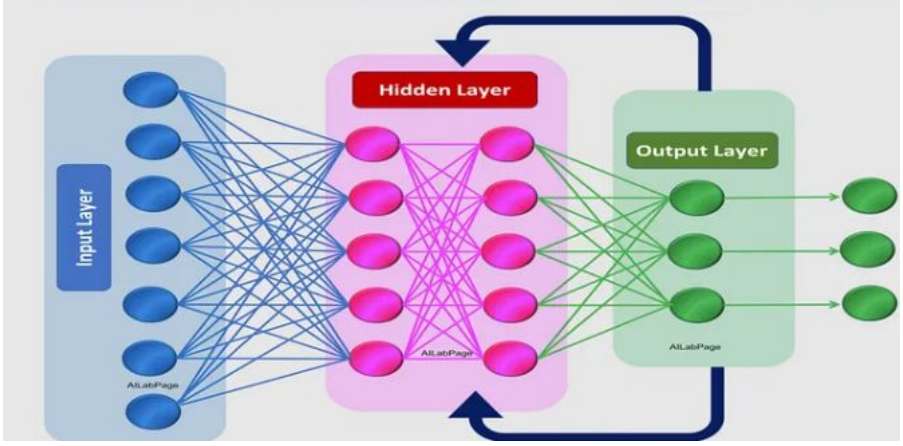
GRU

RNN (Recurrent Neural Network)

São um tipo de rede neural p**rojetada para lidar com dados sequenciais**.

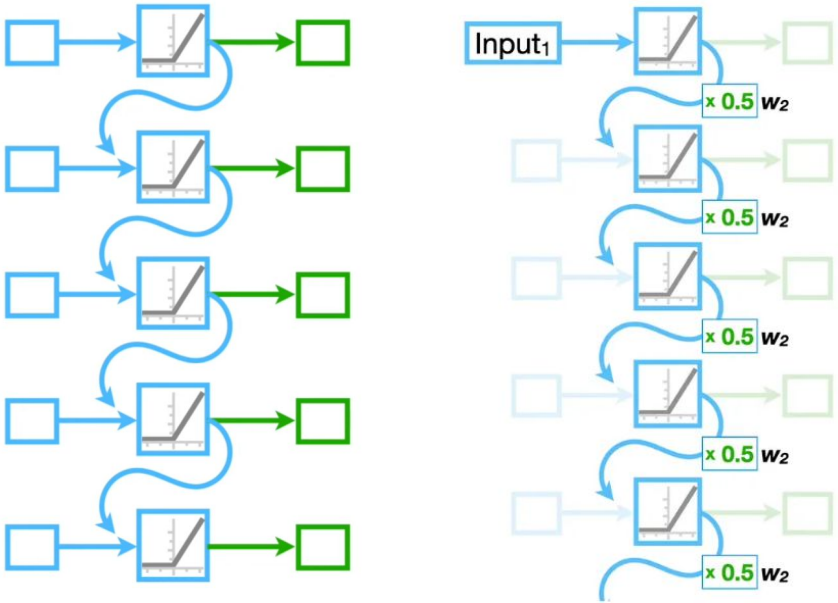
Diferente das redes neurais tradicionais que processam entradas de forma independente, as RNNs possuem “memória”

Elas conseguem levar em conta informações de etapas anteriores para influenciar o processamento da etapa atual.



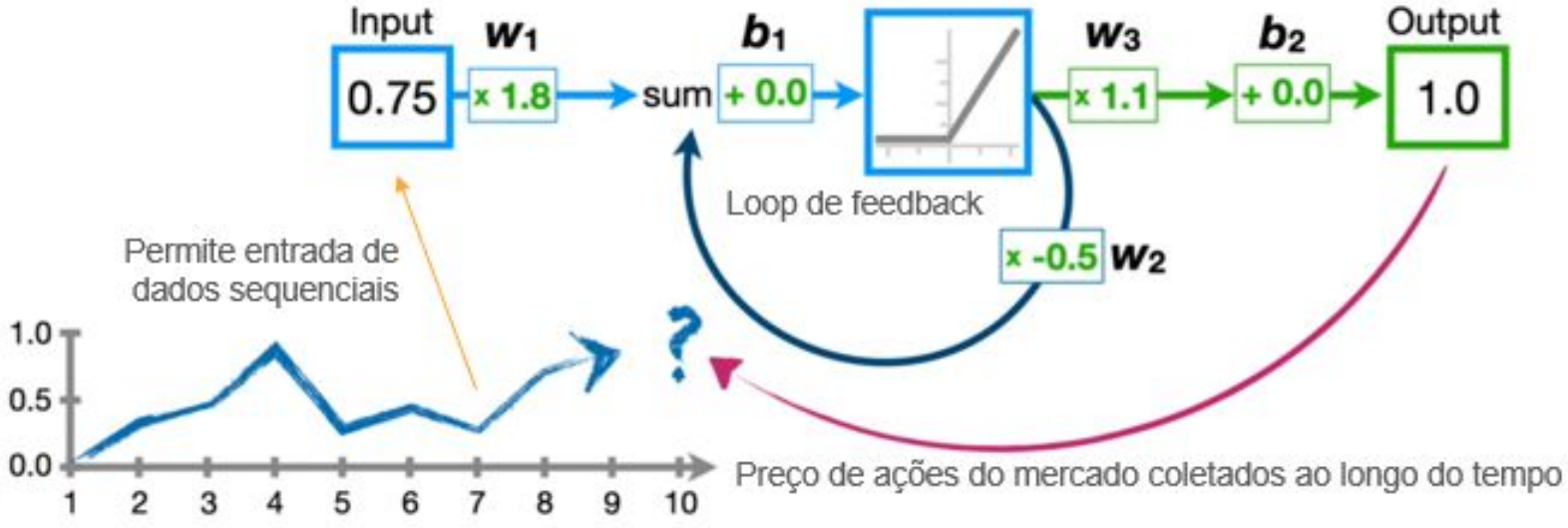
RNN (Recurrent Neural Network)

A RNN usa recorrência para que **entradas anteriores influenciam o processamento das próximas,** funcionando como um acumulador sequencial.

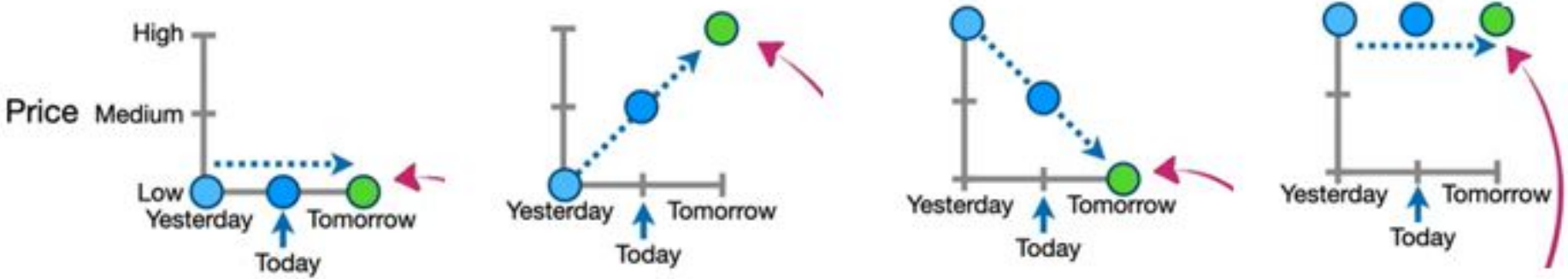


RNN (Recurrent Neural Network)

- Peso (w): parâmetro que multiplica a entrada ou o estado anterior da rede.
- Bias (b) : Viés de valor adicionado ao somatório antes da ativação
- Funções de ativação.: função não linear aplicada ao somatório ponderado



RNN (Recurrent Neural Network)



Exemplo de preços das ações do mercado

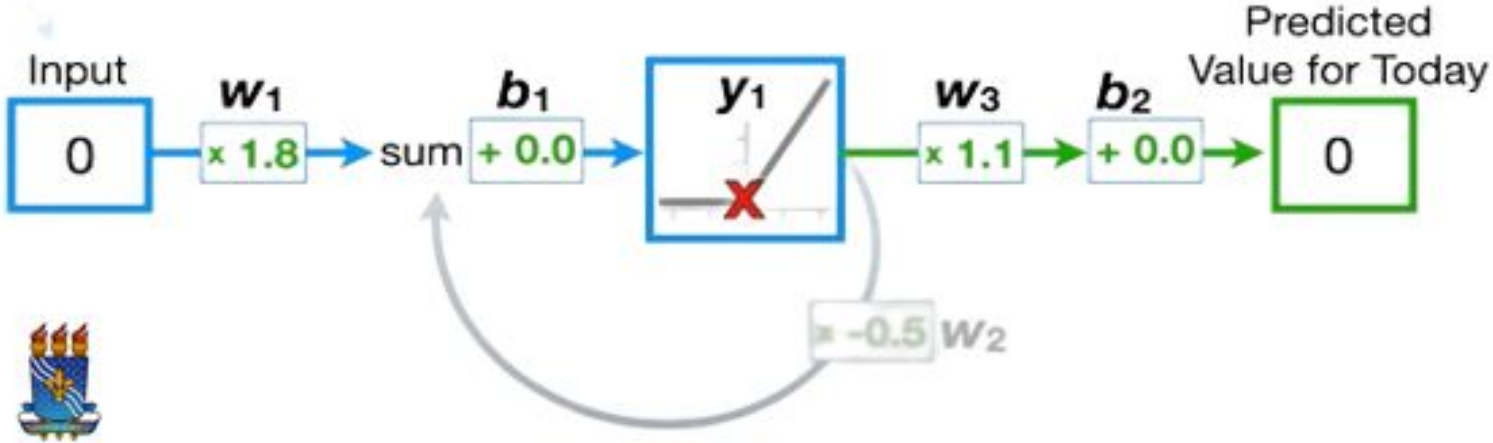
Yesterday × w_1 + b_1 = x-axis coordinate

0 × 1.8 + 0.0 = 0

$f(x) = \max(0, x)$ = y-axis coordinate

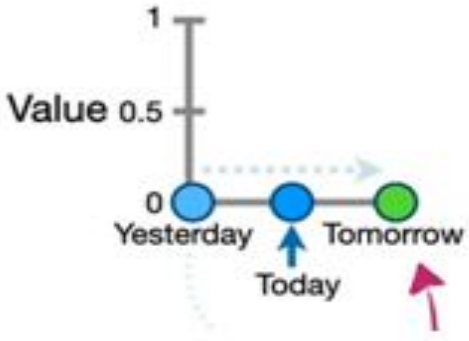
$f(0) = \max(0, 0) = 0 = y_1$

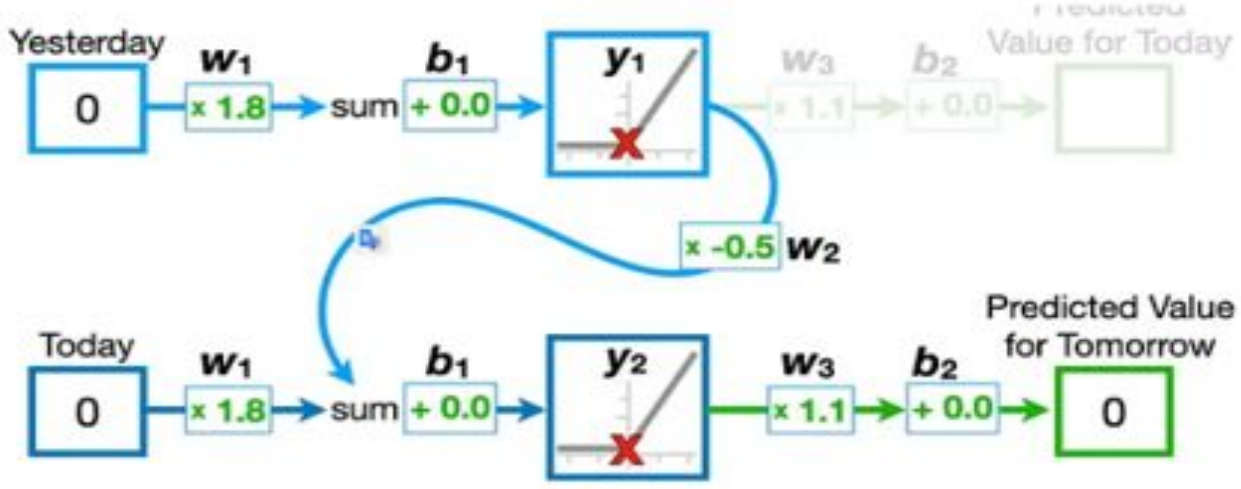
$y_1 \times w_3 + b_2 =$ **0** × 1.1 + 0.0 = 0

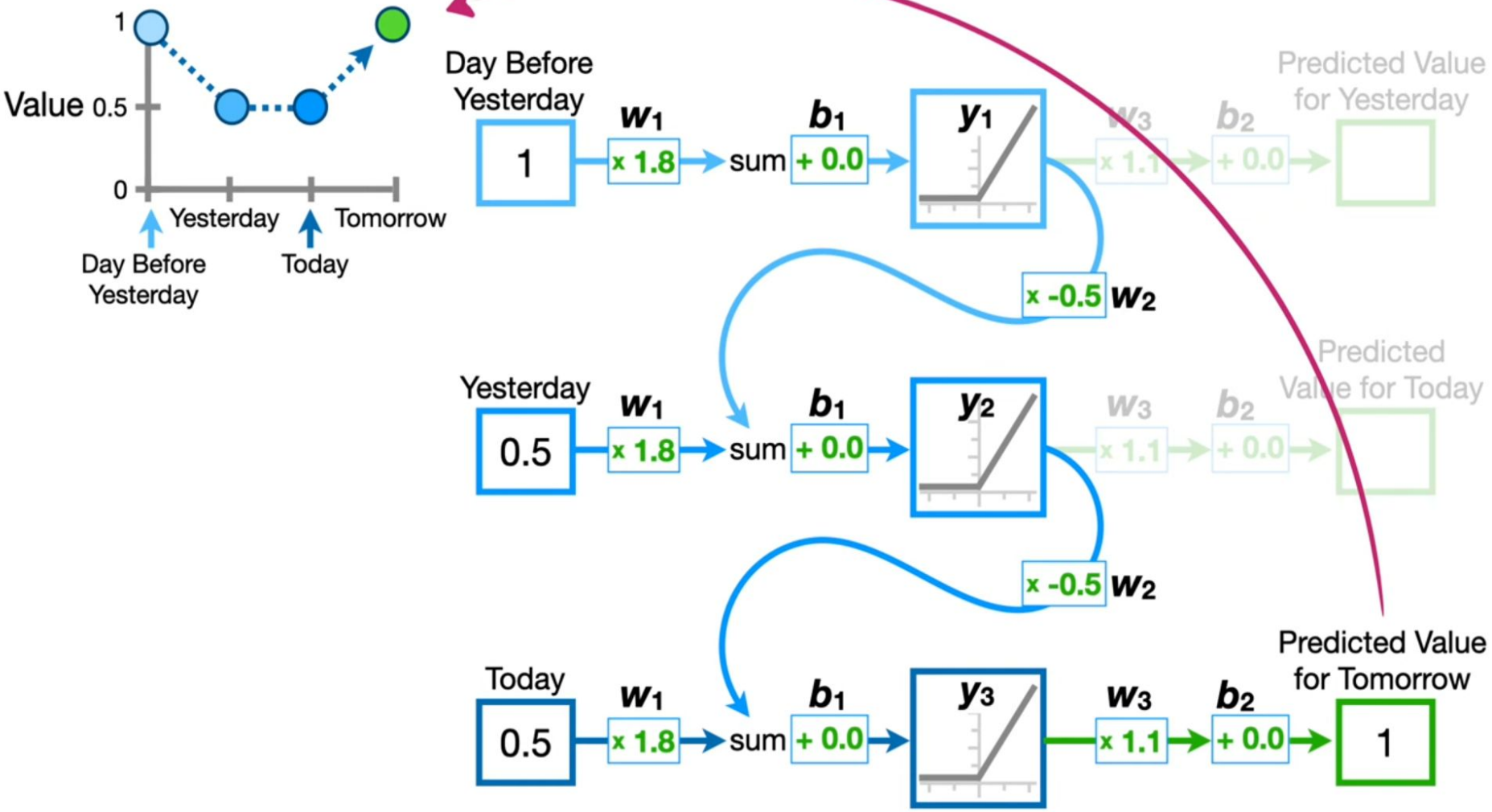


LASER
Laboratório de Engenharia de Sistemas e Robótica

UFPB

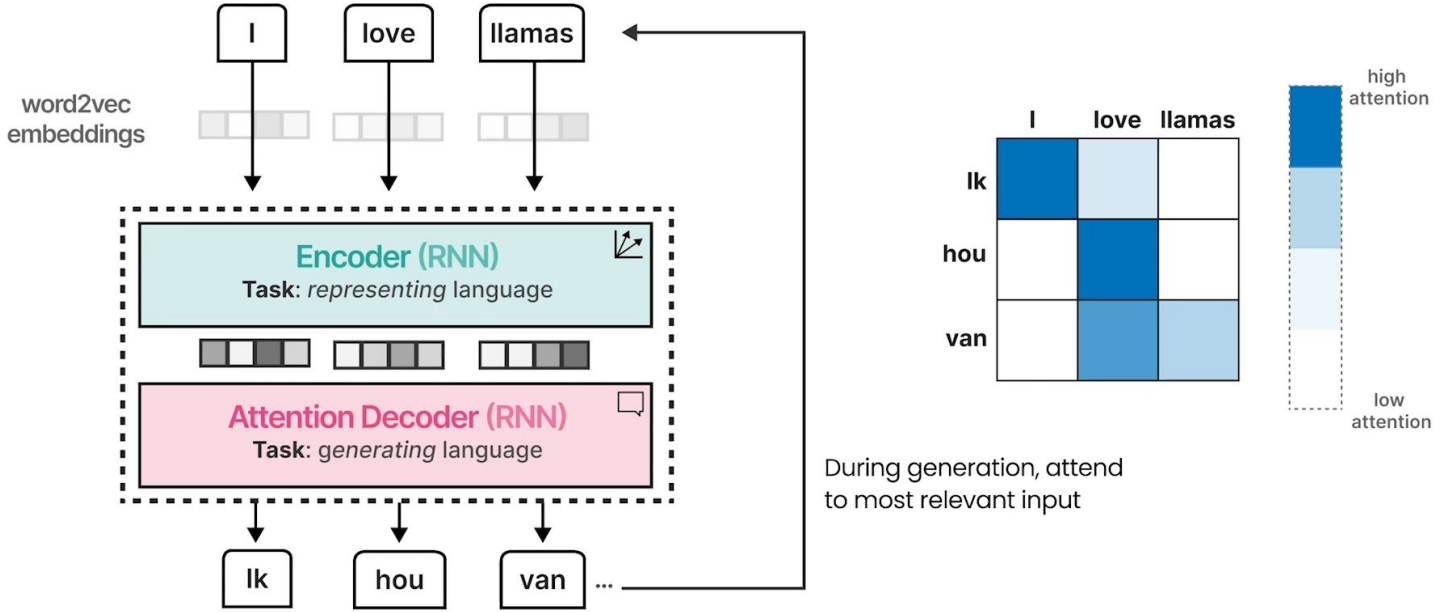

1
Value 0.5
0
Yesterday
Today
Tomorrow


Yesterday
0
w_1
x 1.8
sum
b_1
+ 0.0
y_1
w_3
x 1.1
b_2
+ 0.0
Value for Today
x -0.5
w_2
Today
0
w_1
x 1.8
sum
b_1
+ 0.0
y_2
w_3
x 1.1
b_2
+ 0.0
Predicted Value for Tomorrow
0


1
Value 0.5
0
Yesterday
Tomorrow
Day Before Yesterday
Today
Day Before Yesterday
1
w1
x 1.8
sum
b1
+ 0.0
y1
w3
x 1.1
b2
+ 0.0
Predicted Value for Yesterday
x -0.5 w2
Yesterday
0.5
w1
x 1.8
sum
b1
+ 0.0
y2
w3
x 1.1
b2
+ 0.0
Predicted Value for Today
x -0.5 w2
Today
0.5
w1
x 1.8
sum
b1
+ 0.0
y3
w3
x 1.1
b2
+ 0.0
Predicted Value for Tomorrow
1

Evolução do Modelo RNN

- Precursor da Attention com arquitetura RNN EncDec (2014)
- "NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE [https://arxiv.org/pdf/1409.0473]"



Problema

- **Processamento Sequencia**l: RNNs e LSTMs processam dados sequencialmente, o que é computacionalmente caro, especialmente para sequências longas. Isso os torna inadequados para processar textos grandes ou lidar com aplicativos em tempo real.

- Problema de **Dissipação do Gradiente**: Embora os LSTMs mitiguem o problema de dissipação do gradiente (um problema matemático) até certo ponto, eles não o superam completamente. Esse problema dificulta a capacidade do modelo de aprender dependências de longo prazo.

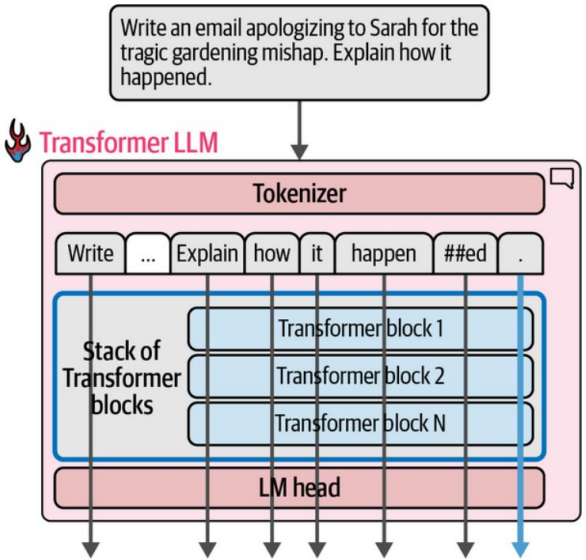
- **Dificuldade em Paralelizar**: Devido à sua natureza sequencial inerente, esses modelos não podem ser facilmente paralelizados, limitando sua eficiência de treinamento em hardware moderno.

Transformer

Arquitetura de rede neural projetada para processar sequências de dados, especialmente texto, de forma mais eficiente que RNNs e LSTMs.

Principais características:

1. **Mecanismo de Atenção (Attention)** Permite que o modelo foque nas partes mais relevantes da entrada ao gerar a saída e resolve o problema de dependências de longo prazo, comum em RNNs.
2. **Paralelização:** Processa todos os elementos da sequência ao mesmo tempo, acelerando o treinamento.
3. **Escalabilidade:** Funciona bem com sequências longas e grandes volumes de dados.



RNNs vs. Transformer

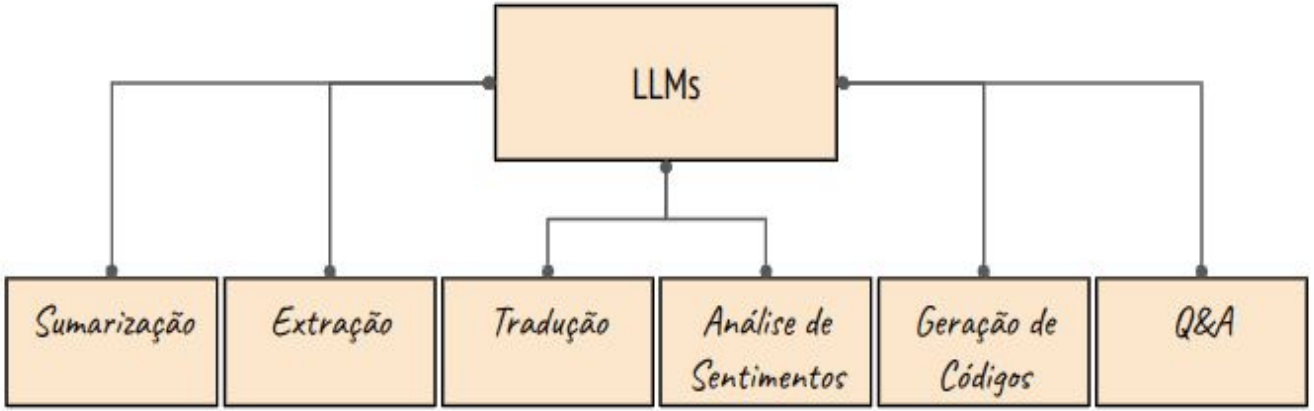
Característica	RNN	Transformer
Dependência sequencial	Sim, passo a passo	Não, atenção permite acesso global
Paralelismo	Limitado, só por batch	Total, cada token processado simultaneamente
Velocidade em sequências longas	Lenta	Muito rápida
Capacidade de contexto	Curto a médio	Longo, pois atenção considera toda a sequência
Aplicações típicas	Texto curto, séries temporais	Tradução, geração de texto, LLMs, áudio

LLM (Large Language Models)

Modelos de Linguagem de Grande Escala são treinados em enormes conjuntos de dados de texto, a fim de aprender padrões e estruturas na linguagem.

Uma vez treinados, eles podem gerar novos textos que são coerentes e seguindo o contexto dado.

Os modelos LLM são baseados na arquitetura Transformer de aprendizado profundo



LLM (Large Language Models)

Modelo de Linguagem via API da Groq

```python
from groq import Groq
client = Groq(api_key="CHAVE")

def gerar_texto(prompt):
    resposta = client.chat.completions.create(
        messages=[{"role": "user", "content": prompt}],
        model="llama3-8b-8192"   # modelo de LLM disponível na plataforma Groq
    )
    return resposta.choices[0].message.content
prompts = ["Explique o que é inteligência artificial de forma simples." ]
for prompt in prompts:
    resposta = gerar_texto(prompt)
    print(f"Prompt: {prompt}\nResposta: {resposta}\n{'-'*50}")
```

Bibliografia

- Bommasani et al., "On the Opportunities and Risks of Foundation Models" (2021).
- Papers recentes de arXiv sobre LLMs e Transformers.
- Vaswani et al., "Attention is All You Need" (2017).
- Criação de GPT: https://help.openai.com/en/articles/8554397-creating-a-gpt