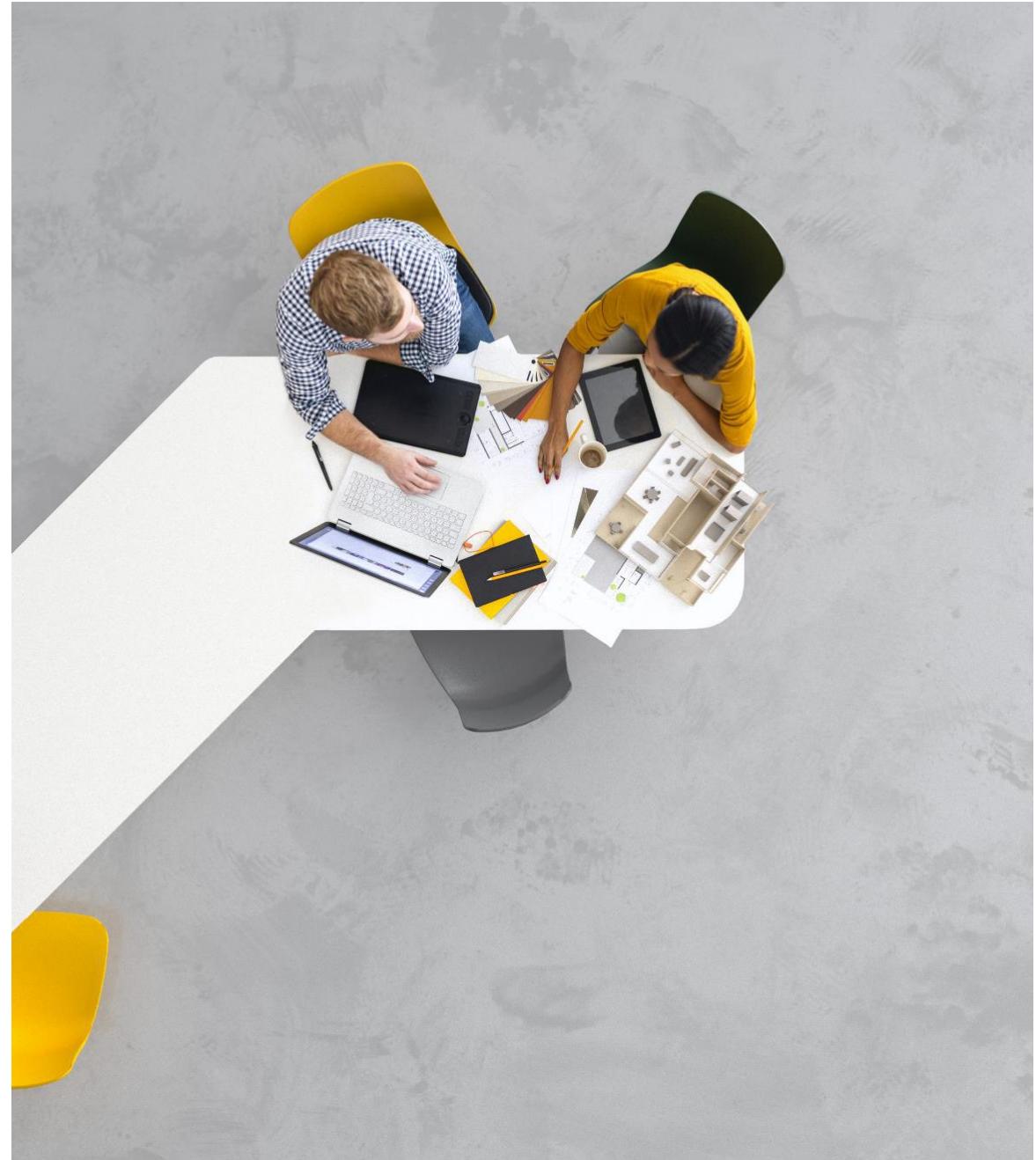


# O que é uma IA generativa ?

## Parte III

Glauco Reis



# AS IAS GENERATIVAS. QUANDO SURGIRAM ?

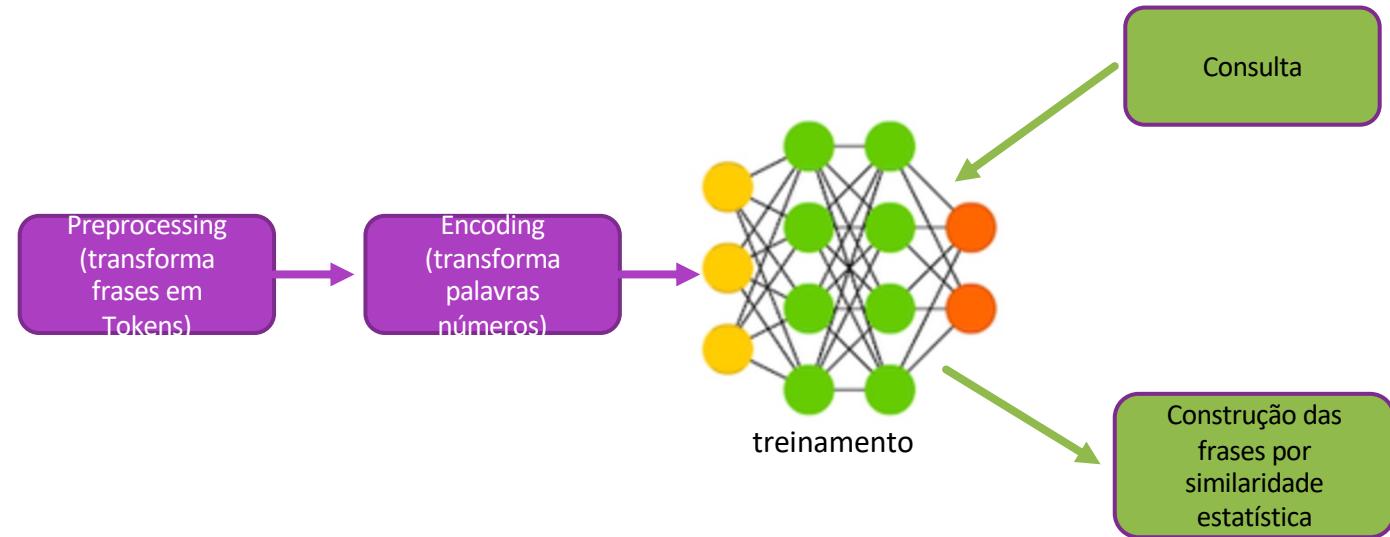
- Junho 2014 (provavelmente primeiro artigo sobre IAS generativas)
- 2015. OpenAI fundada por Sam Altman, Elon Musk, Greg Brockman, Peter Thiel
- 2022. InstructGPT introduzido. Uma máquina para seguir instruções humanas treinada com feedback humano.
- 2022. ChatGPT, um irmão do InstructGPT, foi criado. Foi treinado para conversações humanas de formato genérico, e treinado com amplo conjunto de informações da Web.

## Generative Adversarial Nets

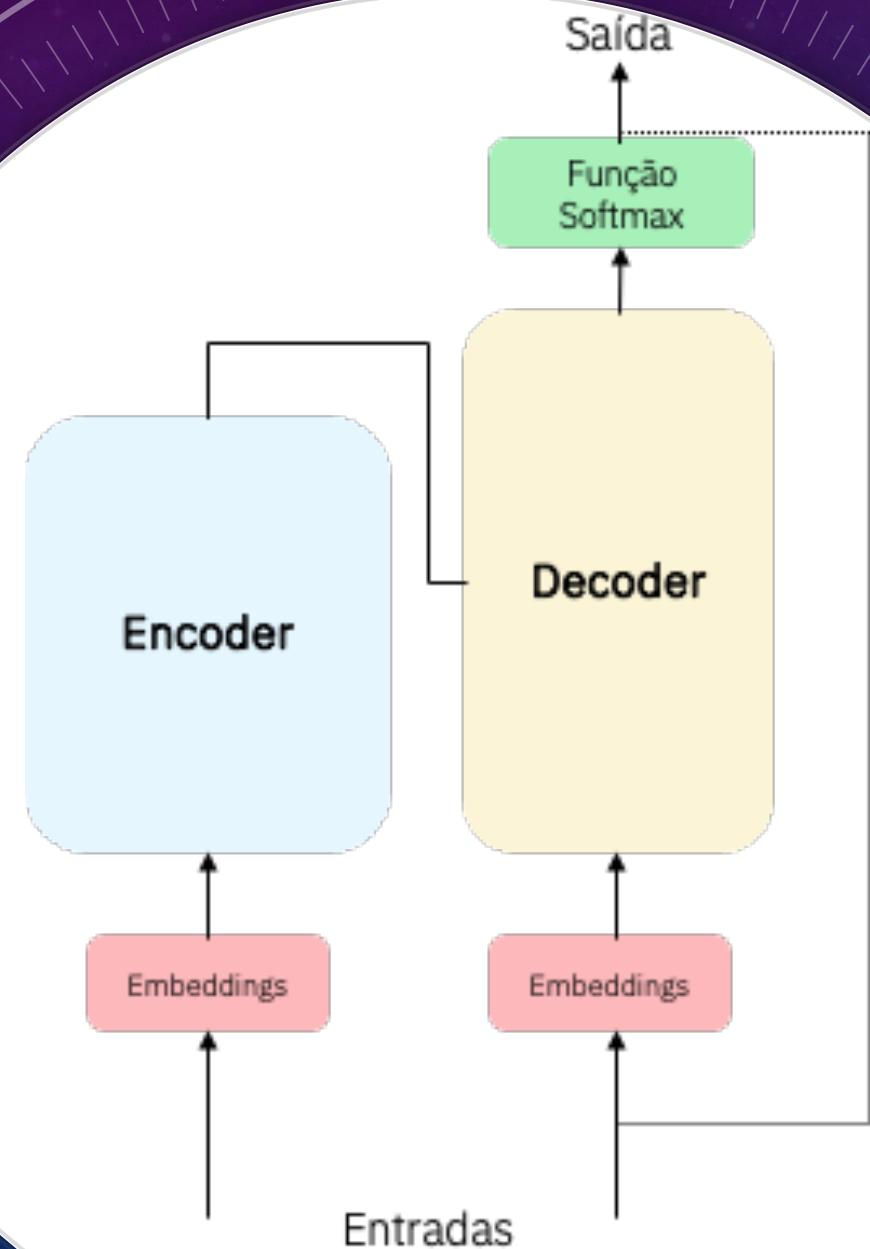
Ian J. Goodfellow, Jean Pouget-Abadie\*, Mehdi Mirza, Bing Xu, David Warde-Farley,  
Sherjil Ozair†, Aaron Courville, Yoshua Bengio‡

Département d'informatique et de recherche opérationnelle  
Université de Montréal  
Montréal, QC H3C 3J7

# COMO O CHATGPT FUNCIONA ?



# UMA ARQUITETURA FORMAL DOS TRANSFORMERS



# PARTE DO TREINAMENTO

- Transformers – Transformam frases subjetivas em frases objetivas. Exemplo

Glauco gosta de laranjas. Ele as come todos os dias



- Durante o treinamento, palavras encontradas na massa de treinamento recebem um peso estatístico, indicando a probabilidade de aparecimento naquele ponto. Ex.



Glauco gosta de comer X

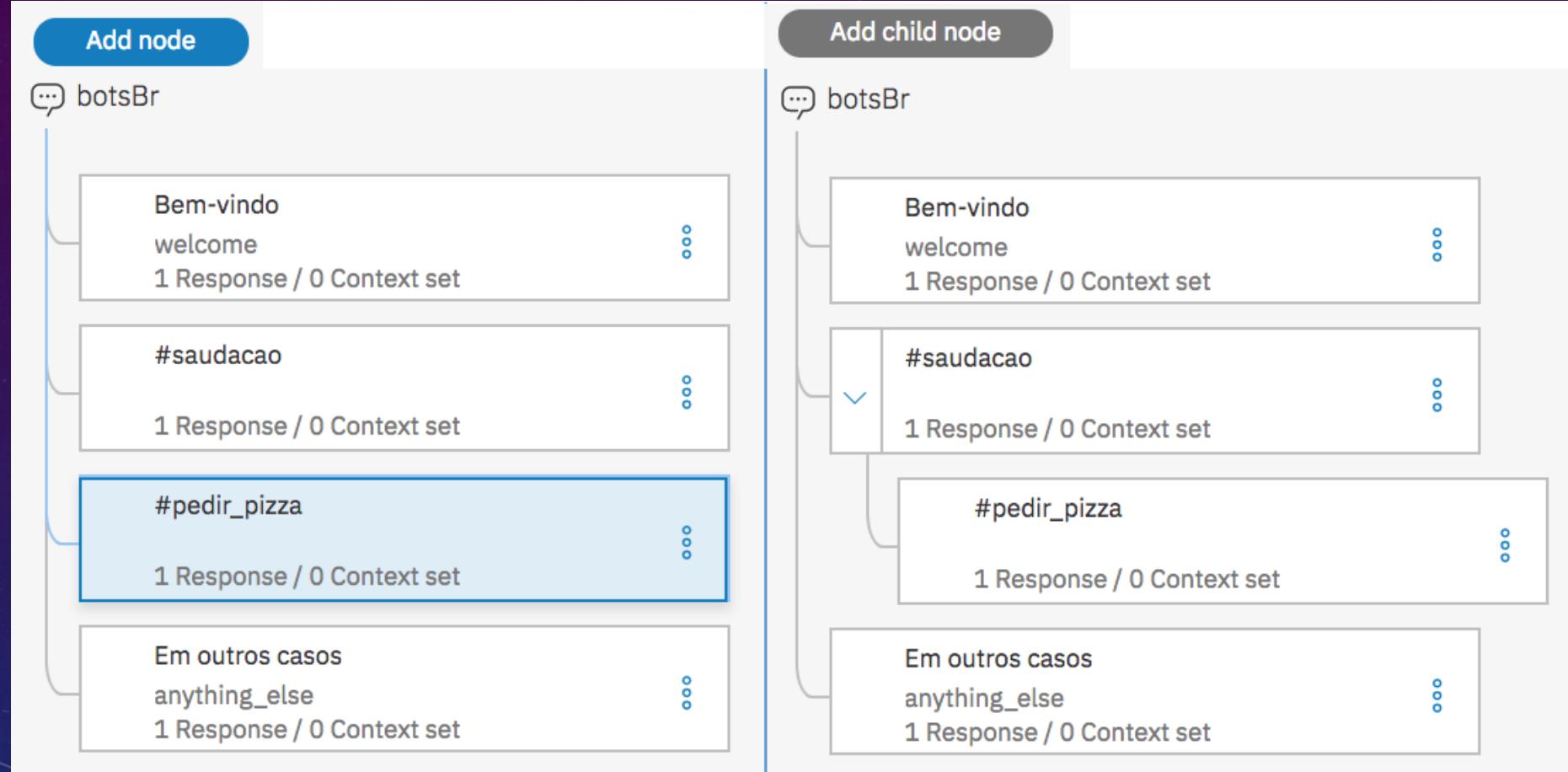


X = peras (80%)  
X = melancia (85%)  
X = pregos (3 %)  
X = cimento (2 %)

# E DAI ? O QUE ISTO SIGNIFICA?

- Como as respostas são ESTATÍSTICAS, não é possível estabelecer um controle rígido nelas
- É possível, mesmo que treinado com um controle absolutamente rígido de respostas corretas, termos respostas sem precisão (Lembre-se, é estatística)
- É da natureza das IAs generativas. Elas geram respostas NOVAS, com base em treinamentos de entradas ANTIGAS
- Por isto IAs generativas são mais bem aplicadas em atividades onde se exige variedade e inovação na resposta, mas não precisão.

# E COMO O WATSON ASSISTANT FAZ ISTO ?



As respostas são estatísticas, mas de um conjunto de respostas previsíveis. É impossível apresentar uma resposta não desejada, por uma escolha estatística de palavras em uma rede neural.

# WATSON ASSISTANT X CHATGPT

Generativas

- Poemas
- Estilizaçā
- Criação de
- Criação de
- (reunião)
- Em resum
- (normalme
- “criativid

Não existe u



cisão na  
íquinas  
culos  
om legado e  
samento  
e exigem  
s, sem a  
os”

# PARTE DO TREINAMENTO

- Transformam frases subjetivas em frases objetivas. Exemplo

Glauco gosta de laranjas. Ele as come todos os dias



- Durante o treinamento, palavras encontradas na massa de treinamento recebem um peso estatístico, indicando a probabilidade de aparecimento naquele ponto. Ex.



Glauco gosta de comer X



X = peras (80%)  
X = melancia (85%)  
X = pregos (3 %)  
X = cimento (2 %)



# APRESENTANDO O PAPERGPT

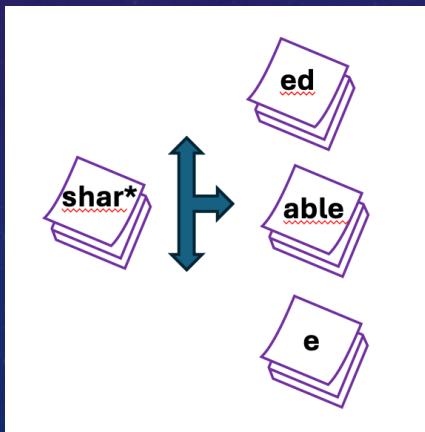
# COMO UMA IA GENERATIVA FUNCIONA ?

We are accounted poor citizens , the patricians good .  
authority surfeits on would relieve us : if they would yield  
us but the superficiality , while it were wholesome , we  
might guess they relieved us humanely but they think we are

0 3 4 5 13 15 18 24 27 32 30  
1 6 7 10 14 16 19 25 28 10 31  
16 8 18 11 15 17 20 26 29 15 0  
2 9 28 12 16 22 21 28 23 0 3

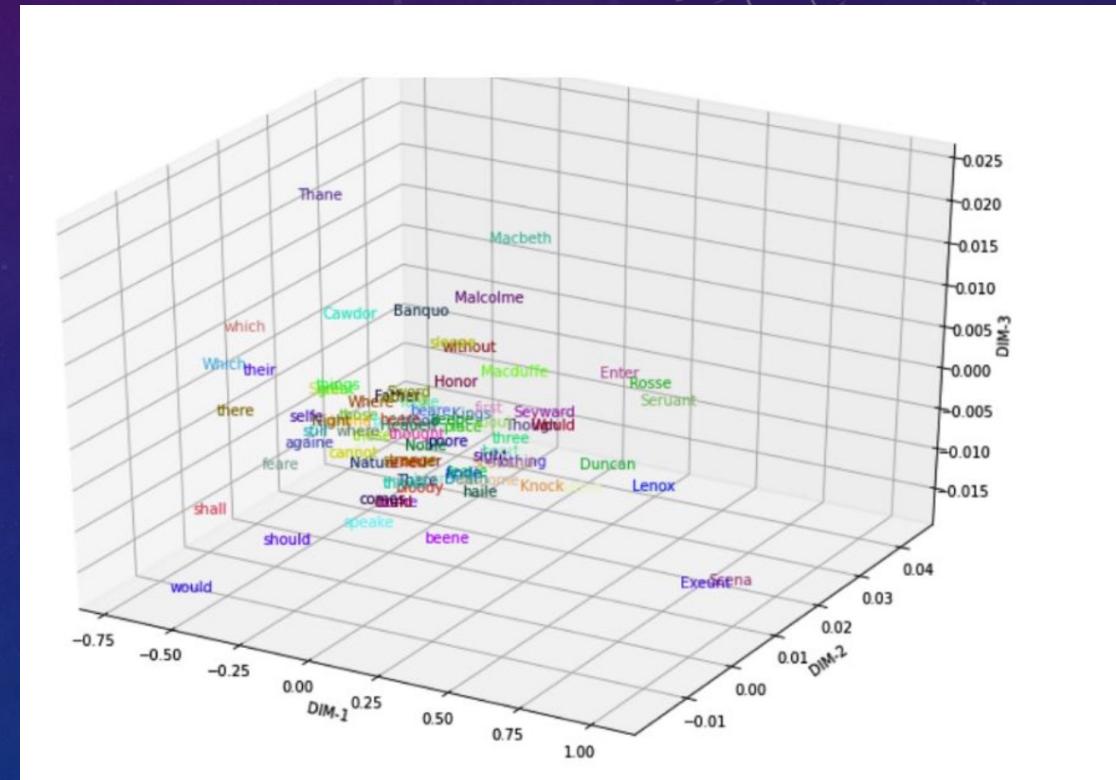
```
In [36]: # separando as palavras
chars = sorted(list(set(text.split())))
vocab_size = len(chars)
print(chars[:2000])
print(vocab_size)
```

```
['+', '+1', '+4=0', '-', '-1', '-1+1=0', '-1^1/2', '-1^2/2', '-2', '-4', '->', '/d', '0', '0+1', '0+1=1', '00', '00
0', '001', '01', '0V', '0]', '0s', '1', '1+0=1', '1+1', '1+1=0', '1+2i', '1-1=0', '1-2', '1/10', '10', '10+0i', '10
0', '100%', '1000', '10000', '101', '11', '12', '1200', '12ns', '13', '14', '15', '1593', '16', '175', '18', '180',
'1800', '1801', '18445744073709550000', '1895', '1921', '1922', '1935', '1960', '1964', '1965', '1966', '1980', '19
82', '1984', '1986', '1997', '1]', '1s', '1x1', '2', '2+', '2-1', '2-1=1', '2/2=1', '20', '20%', '200', '2000', '20
10', '2011', '2014', '2016', '2017', '2019', '2021', '2022', '2023', '2042', '21', '22', '23', '2345J', '24', '2
5%', '250', '255', '256', '256Mb', '2D', '2GHz', '2X-4', '2X2', '2]', '2i', '2x0-4', '2x1=2', '2x2', '2√-1', '3',
'3%', '3/2', '30', '300', '300km/h', '31', '32', '320Gb/s', '32Mb', '33', '34562', '34562+0', '360', '37', '3GHz',
'3i', '3x2x1=6', '4', '40', '4000', '40Km/h', '42', '433', '45', '46', '479', '480', '4X1', '4x3x2x1=24', '5', '5
0', '50%', '512', '550km/h', '5GHz', '5V', '5]', '6', '60', '60%', '600', '6000', '64', '64BITS', '65535', '6GHz',
'7', '70', '7000', '750', '762234', '762234+0', '8', '80', '80%', '800', '8712312J]', '8:00', '8Gb', '8bits', '9',
'90', '98%', ':', ';', '=', '=2x1-4', 'A', 'A->B->C->A', 'A->C->B->A', 'ABCDA', 'ABDCA', 'ACBDA', 'ACDBA',
'ADBCA', 'ADCBA', 'AGORA', 'AI', 'AIOps', 'AND', 'ANTES', 'AO', 'API', 'AQUELE', 'AQUI', 'ATAs', 'AUMENTANDO', 'Abaixo',
'Ab
solutamente', 'Acabamos', 'Achamos', 'Achei', 'Acho', 'Acontece', 'Aconteceu', 'Acredite', 'Acreditem', 'Acredito',
'Adão', 'Afinal', 'Agamoto', 'Age', 'Agora', 'Aguarde', 'Ahh', 'Ahhh', 'Ahhhhh', 'Ai', 'Ainda', 'Airton',
'Ajustando', 'Alan', 'Albert', 'Alfred', 'Algo', 'Algum', 'Algumas', 'Alguns', 'Alguém', 'Alias', 'Alice', 'Aliás',
'Alpha', 'AlphaGO', 'AlphaGo', 'Alphago', 'Alunos', 'Além', 'Ambas', 'Ambassador', 'Ambos', 'Analizar', 'Andrade',
'Antenas', 'Antes', 'Anticítera', 'Ao', 'Apaixonado', 'Apenas', 'Apesar', 'Aplicar', 'Aplicaram', 'Aposto', 'Appl
o', 'Aproveite', 'Aquele', 'Aqueles', 'Aquele', 'Aqueles', 'Aqui', 'Aquile', 'Arantes', 'Aritméticos', 'Arnaldo'
```



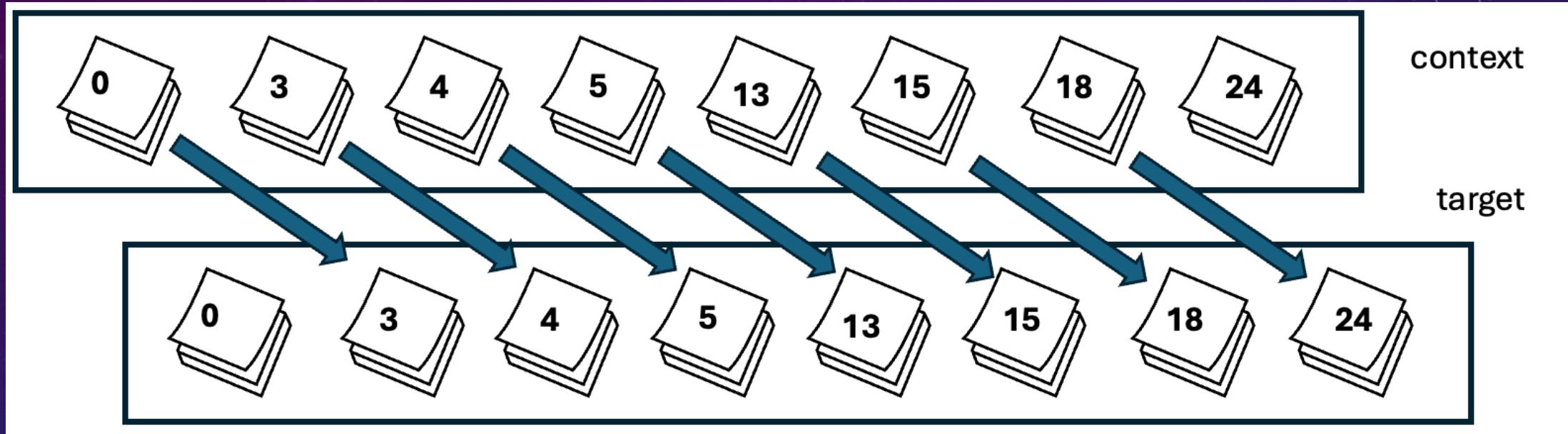
É um pouco mais complicado que isto

# DISTRIBUIÇÃO DAS PALAVRAS



# QUAL A PRÓXIMA PALAVRA ?

Glauco gosta de comer



Glauco gosta de comer X



X = peras (80%)

X = melancia (85%) - 13

X = pregos (2%)

X = cimento (1 %)

# COMO FAZER MASSIVAMENTE ?



inputs:

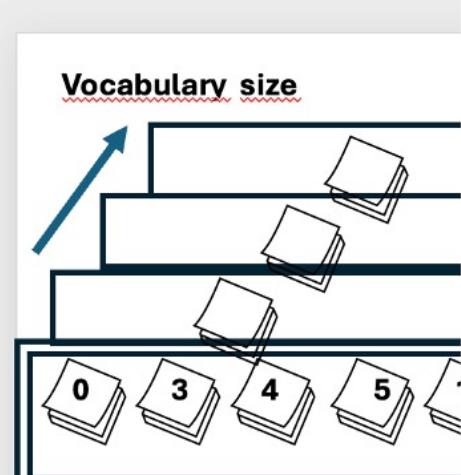
```
torch.Size([4, 8])  
tensor([[1181, 3679, 6110, 4585, 8301, 6069, 750, 1300],  
       [6229, 532, 7871, 3863, 7024, 7136, 3481, 3178],  
       [2639, 3311, 1382, 3112, 3629, 2293, 2639, 3080],  
       [1382, 5713, 3315, 6250, 5326, 7024, 4270, 6256]])
```

targets:

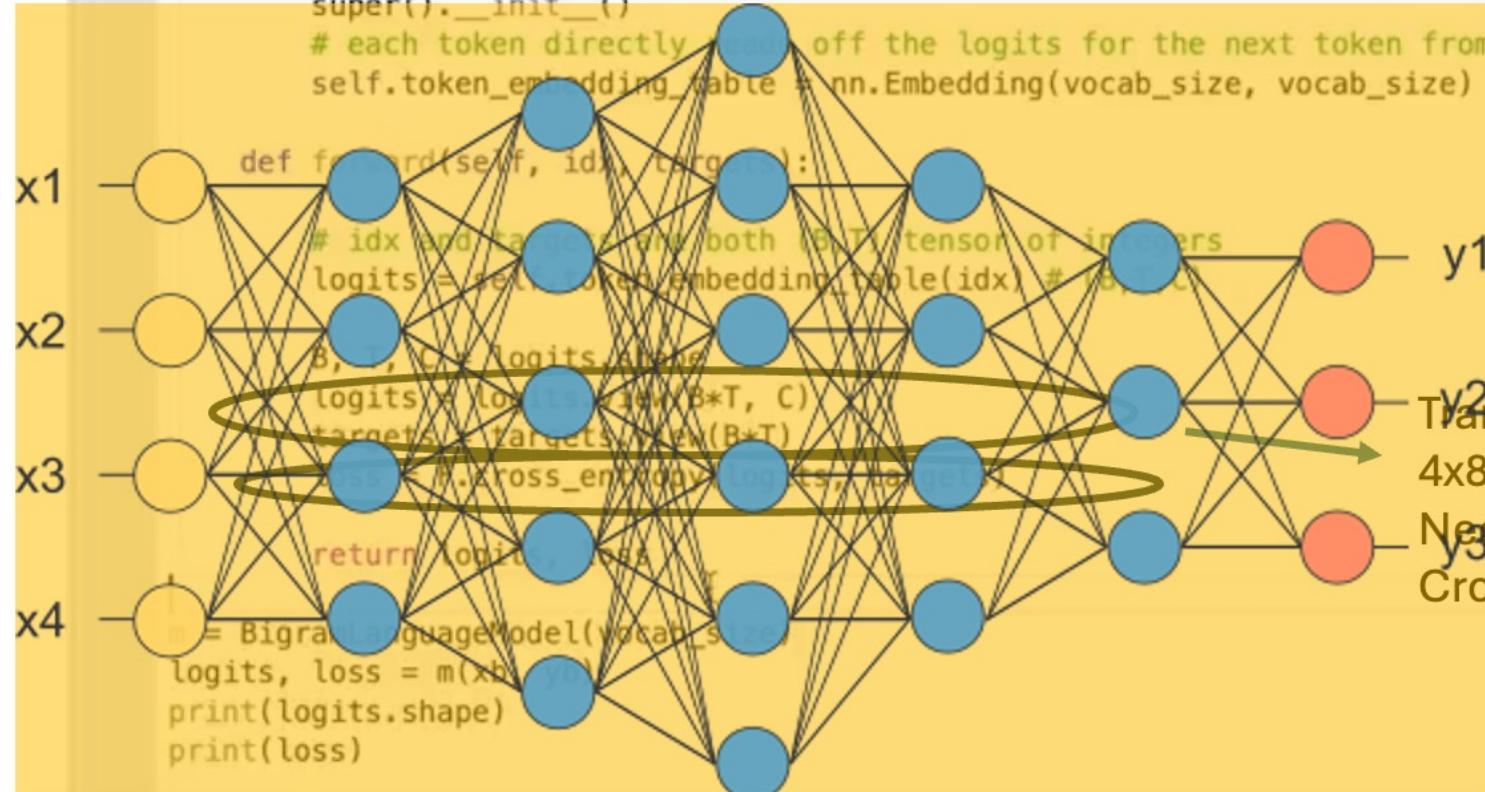
```
torch.Size([4, 8])  
tensor([[3679, 6110, 4585, 8301, 6069, 750, 1300, 2799],  
       [ 532, 7871, 3863, 7024, 7136, 3481, 3178, 6664],  
       [3311, 1382, 3112, 3629, 2293, 2639, 3080, 6101],  
       [5713, 3315, 6250, 5326, 7024, 4270, 6256, 6121]])
```

# FINALMENTE A IA !!!!

**Vocabulary size**



the actual distribution, or true



```
def __init__(self, vocab_size):
    super().__init__()
    # each token directly maps off the logits for the next token from a
    self.token_embedding_table = nn.Embedding(vocab_size, vocab_size)

    def forward(self, idx, targets=None):
        # idx and targets are both (B,T) tensor of integers
        logits = self.token_embedding_table(idx) # B*T, C
        logits = logits.view(-1, logits.shape[-1])
        targets = targets.view(-1)
        loss = F.cross_entropy(logits, targets, reduction='sum')
        return logits, loss

    = BiLSTMLanguageModel(vocab_size)
logits, loss = m(xb)
print(logits.shape)
print(loss)
```

L: torch.Size([32, 65])  
tensor(4.8786, grad\_fn=<NllLossBackward0>)

```
class BigramLanguageModel(nn.Module):

    def __init__(self, vocab_size):
        super().__init__()
        # each token directly reads off the logits for the next token from a
        self.token_embedding_table = nn.Embedding(vocab_size, vocab_size)
        print(self.token_embedding_table.weight.shape)
        print(self.token_embedding_table.weight[:5,:5])
        print(self.token_embedding_table(torch.tensor([0])))
        print(self.token_embedding_table(torch.tensor([0])).shape)

    def forward(self, idx, targets=None):
        # idx and targets are both (B,T) tensor of integers
        logits = self.token_embedding_table(idx) # (B,T,C)
        print('forward')
        print(logits.shape)
        if targets is None:
            loss = None
        else:
            B, T, C = logits.shape # B=4 (batch) T=8 (Input Tokens) C=8k (size)
            logits = logits.view(B*T, C) # cross entropy function needs a b:
            targets = targets.view(B*T) # we generate bidimensional for input
            loss = F.cross_entropy(logits, targets)
        return logits , loss

    def generate(self, idx, max_new_tokens):
        # idx is (B, T) array of indices in the current context
        for _ in range(max_new_tokens): # get the predictions
            logits, loss = self(idx) # focus only on the last time step
            #print(logits.shape)
            logits = logits[:, -1, :] # becomes (B, C) - apply softmax to get
            #print(logits.shape)
            probs = F.softmax(logits, dim=-1) # (B, C) # sample from the dist
            idx_next = torch.multinomial(probs, num_samples=1) # (B, 1) - ap
            idx = torch.cat((idx, idx_next), dim=1) # (B, T+1)
        return idx

m = BigramLanguageModel(vocab_size)
out, loss = m(xb, xb)
```

COMO ISTO  
É EM  
PYTHON ?

# E QUAL O TAMANHO DESTE MAPA ?

40.000

'CONSCIENTEMENTE', 'CONTRA', 'CPD', 'CPH4', 'CPU', 'CPUs', 'CUDA', 'CX', 'Cabe', 'Cada', 'Caixeiro', 'Call', 'Calm', 'Caos', 'Caos:', 'Caprio', 'Cara', 'Carl', 'Carolus', 'Carry', 'Carry+0+1', 'Casas', 'Cego', 'Centers', 'Certe', 'Buscamos', 'Chamamos', 'Charles', 'Consegue', 'Einstein', 'ChatBot', 'ChatBots', 'ChatGPT', 'ChatGPT2', 'ChatGPT3', 'ChatGPI4', 'ChatGPT:', 'Chatbots', 'Chatoff', 'Chega', 'Chegaremos', 'Chicó', 'China', 'Chopp', 'Chrome', 'Claro', 'Cloro', 'Cloud', 'Clusius', 'Cobol', 'Coisas', 'Colaborar', 'Colocar', 'Coloque', 'Coloquei', 'Com', 'Começam', 'Buscamos', 'Começaram', 'Começaremos', 'Como', 'Comparando', 'Comparativamente', 'Compare', 'Complex', 'Comprei', 'Computador', 'Computadores', 'Computar', 'Computação', 'Computer', 'Computing', 'Conciliar', 'Concordo', 'Concret', 'Confesso', 'Conforme', 'Consciência', 'Consegue', 'Conseguimos', 'Conservação', 'Constantinopla', 'Consumidor', 'Contador', 'Contas', 'Contato', 'Continua', 'Control', 'Controlled', 'Controlled', 'Convencionou-se', 'Coproc', 'sadores', 'Core', 'Cores', 'Coroa', 'Corresponde', 'Cossenos', 'Cria', 'Criamos', 'Criar', 'Criação', 'Cuidado', 'Chamamos', 'Cálculos', 'D', 'D1', 'D2', 'DA', 'DADOS', 'DE', 'DIA', 'DIMINUINDO', 'DNA', 'DSL', 'DUAS', 'Da', 'Dali', 'Danilo', 'Darwin', 'Darwinismo', 'Dawkins', 'Daí', 'De', 'Deep', 'DeepMind', 'Delphi', 'Dentro', 'Depende', 'Depe', 'dendo', 'Depois', 'Depp', 'Desabafo', 'Desaparecem', 'Descartes', 'Desculpe', 'Desde', 'Desejamos', 'Desmontou', 'esta', 'Deu', 'Deus', 'Dev', 'Deve', 'Devemos', 'Deveria', 'Deverão', 'Devo', 'Difícil', 'Digo', 'Dino', 'Diríamo', 's', 'Discussões', 'Disto', 'Diversas', 'Dividimos', 'Dizem', 'Dizemos', 'Do', 'Document', 'Does', 'Doing', 'Dois', 'Charles', 'in', 'Domingo', 'Dos', 'Doutor', 'Doutorado', 'Down', 'Dr', 'Dragster', 'Dream', 'Duas', 'Durante', 'Duvido', 'Dá', 'Dê', 'E', 'E1', 'E2', 'ELMo', 'EM', 'ENTRADA', 'EQUAÇÃO', 'ERRADO', 'ESTATISTICAMENTE', 'ESTÁ', 'EUA', 'EXT', 'FMAMENTE', 'Edson', 'Edward', 'Einstein', 'Finstein', 'Fis', 'Fla', 'Flas', 'Fle', 'Fles', 'Fletrônica', 'Elevando'

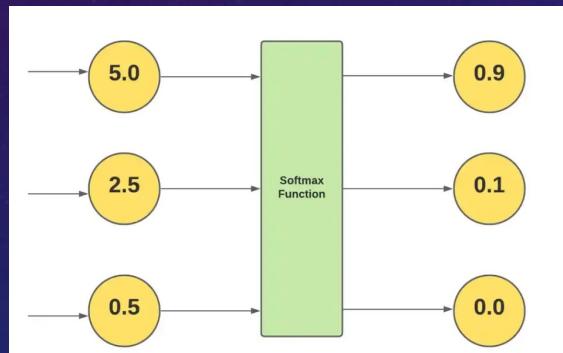
40 TB de dados e 8 milhões de artigos da Wikipédia

## Softmax e multinomial

```
def generate(self, idx, max_new_tokens):
    # idx is (B, T) array of indices in the current context
    for _ in range(max_new_tokens):    # get the predictions
        logits, loss = self(idx)        # focus only on the last time step - SELF CALLS forward
        logits = logits[:, -1, :] # becomes (B, C) - apply softmax to get probabilities
        probs = F.softmax(logits, dim=-1) # (B, C) # sample from the distribution
        idx_next = torch.multinomial(probs, num_samples=1) # (B, 1) - append sampled index to the running sequence
        idx = torch.cat((idx, idx_next), dim=1) # (B, T+1)
    return idx

m = BigramLanguageModel(vocab_size)
print(decode(m.generate(torch.zeros((1, 1), dtype=torch.long) , max_new_tokens=50)[0].tolist()))
```

Softmax – A função softmax é uma função que converte um vetor de valores reais K em um vetor de valores reais K que somam 1.



Multinomial – retorna elementos de um conjunto de probabilidades ordenados pelas maiores probabilidades

```
In [199]: print(torch.multinomial(torch.tensor([0,0,5,2], dtype=torch.float64), num_samples=1))
```

tensor(2)

Quantos elementos desejo

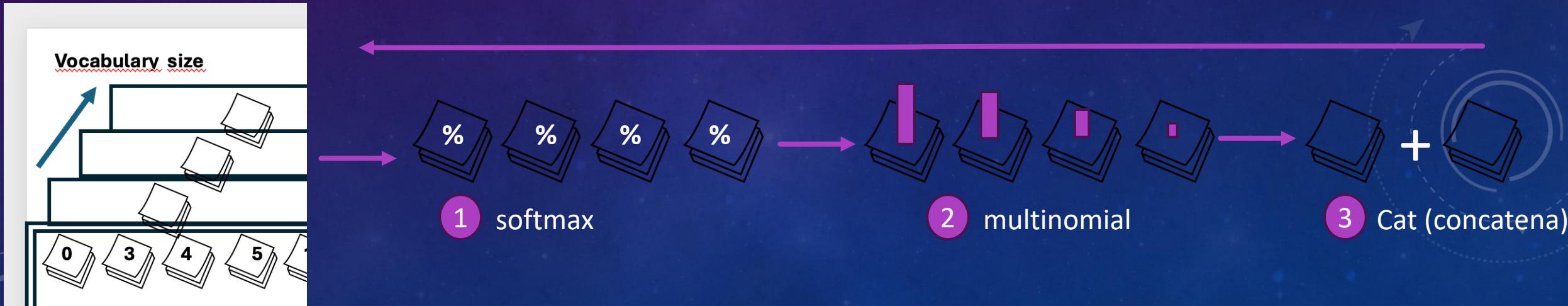
## Softmax e multinomial

Self actiona forward

```
def generate(self, idx, max_new_tokens):
    # idx is (B, T) array of indices in the current context
    for _ in range(max_new_tokens):      # get the predictions
        logits, loss = self(idx)          # focus only on the last time step - SELF CALLS forward
        logits = logits[:, -1, :] # becomes (B, C) - apply softmax to get probabilities
        probs = F.softmax(logits, dim=-1) # (B, C) # sample from the distribution
        idx_next = torch.multinomial(probs, num_samples=1) # (B, 1) - append sampled index to the running sequence
        idx = torch.cat((idx, idx_next), dim=1) # (B, T+1)
    return idx

m = BigramLanguageModel(vocab_size)
print(decode(m.generate(torch.zeros(1, 1), dtype=torch.long), max_new_tokens=50)[0].tolist())
```

SKIcLT;AcELMoTbvZv C?nq-QE33:CJqkOKH-q;:la!oiywkHj

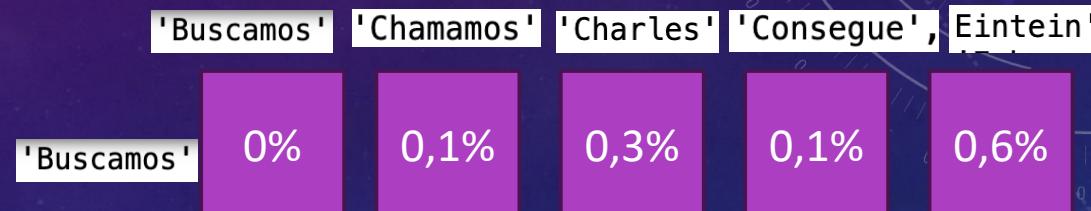


Transforma um array de próximas palavras em probabilidades, retorna o elemento de maior probabilidade e concatena com o resultado anterior

# O QUE É A TEMPERATURA ?



A temperatura é o quanto estamos dispostos a “tolerar” em termos de probabilidade para escolha da próxima palavra



Depois de ordenar (softmax + ordenação)

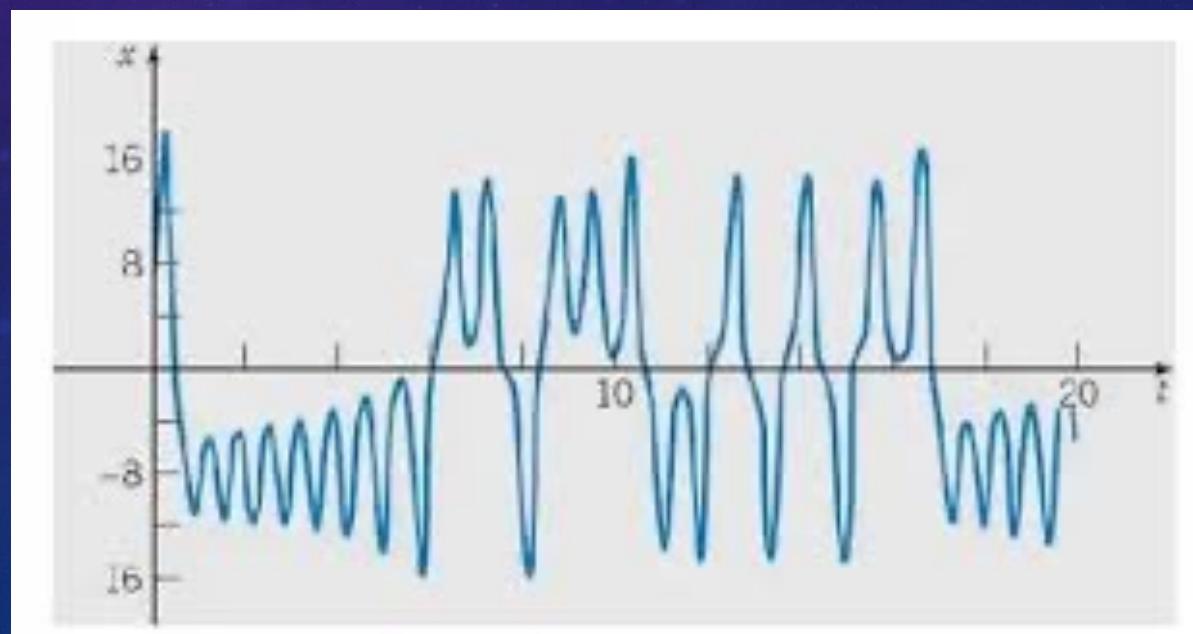
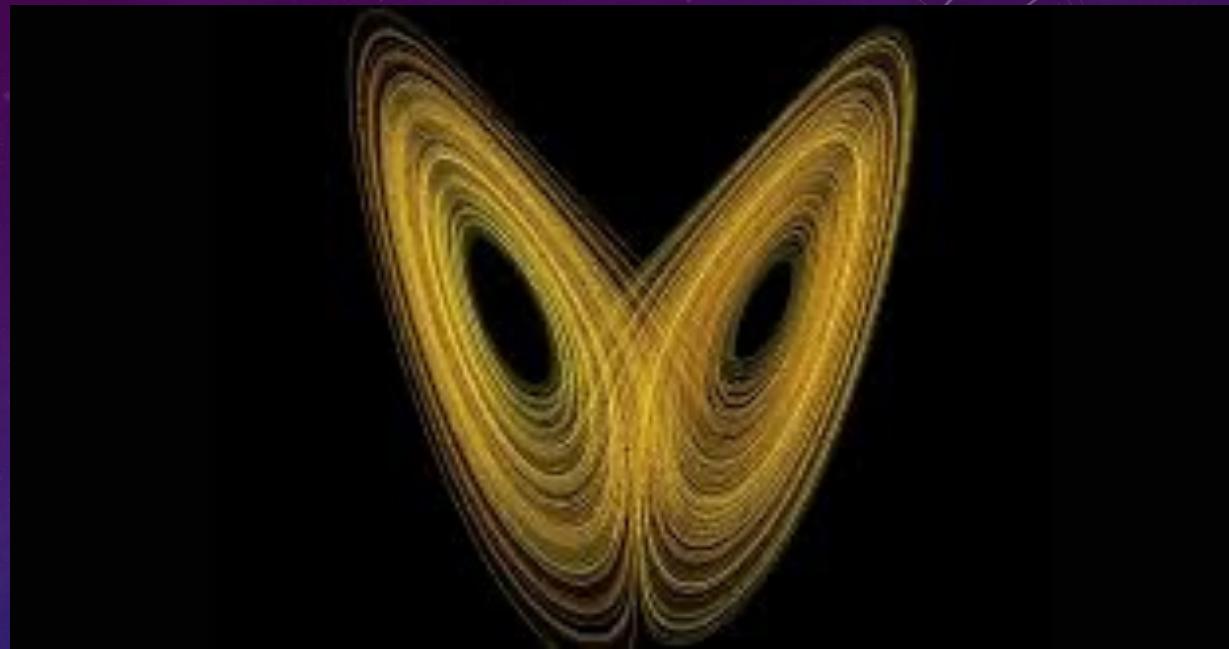
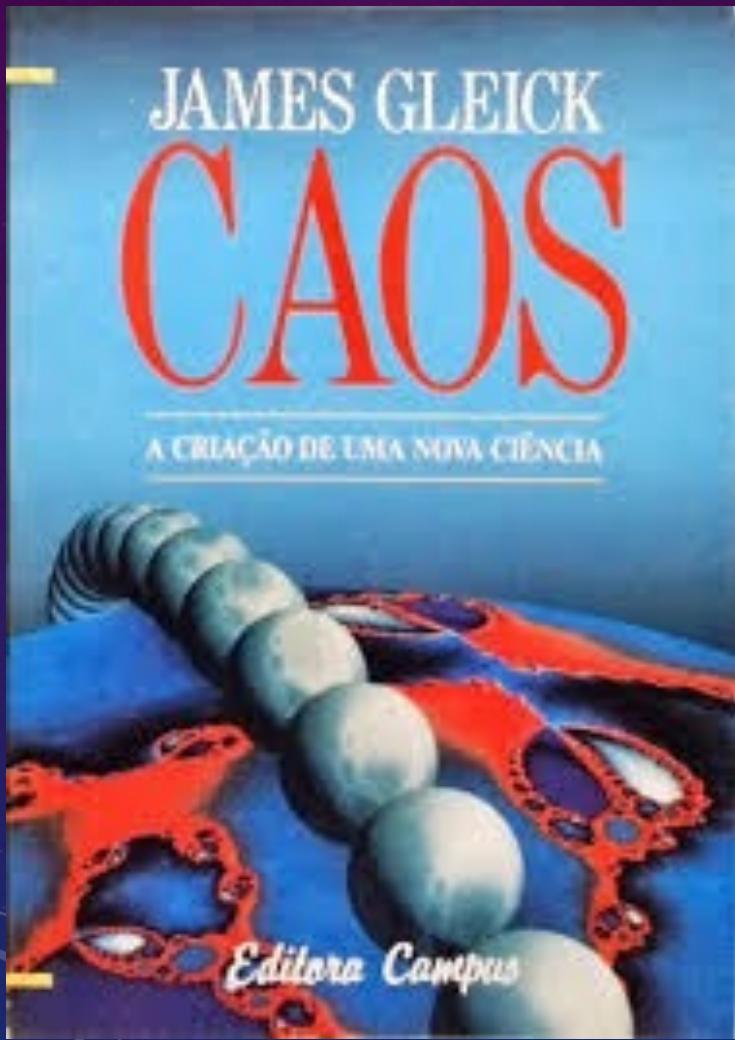


↓ alucinação  
↓ criatividade

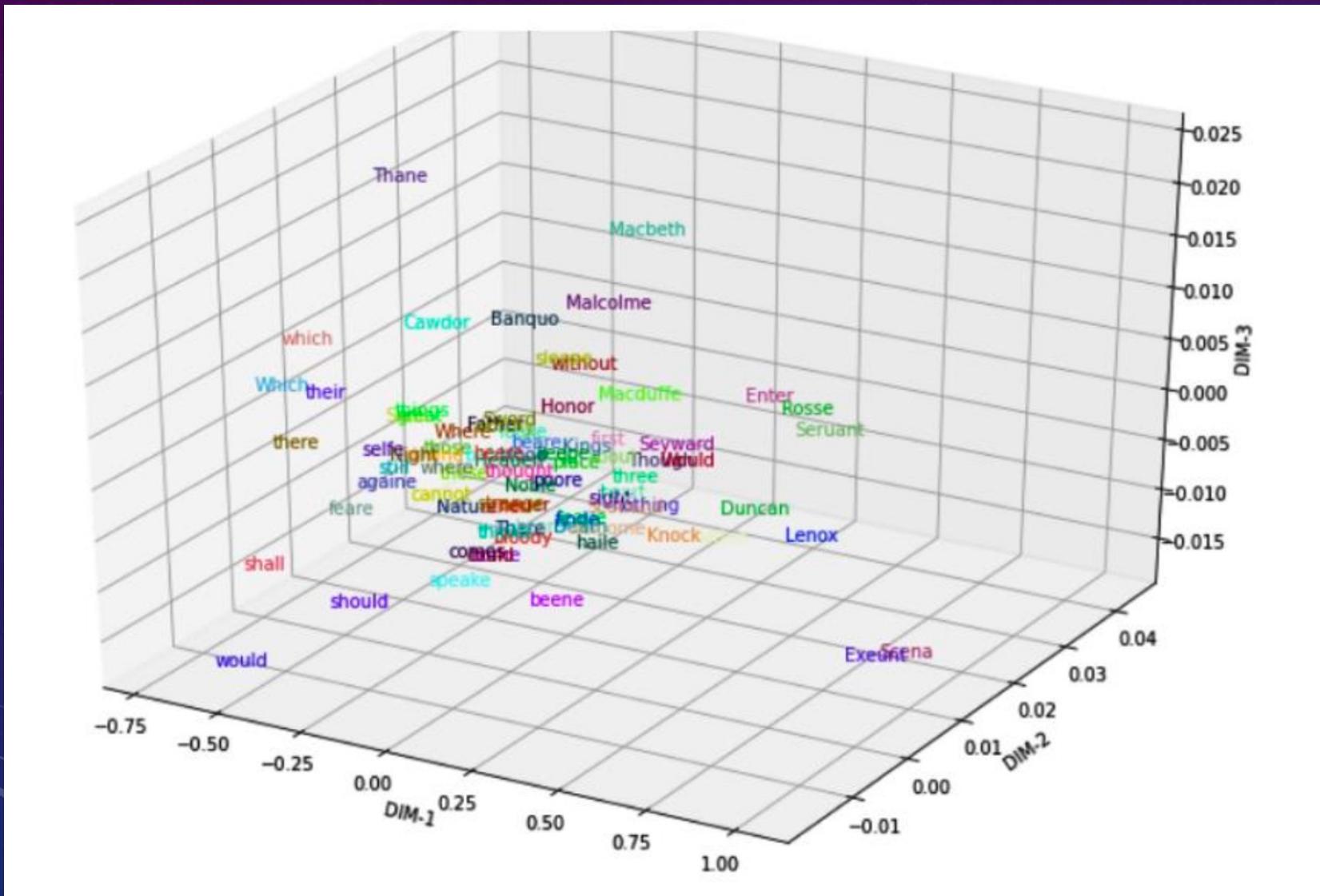
↑ alucinação  
↑ criatividade

temperatura

DÁ PARA NÃO ALUCINAR ?

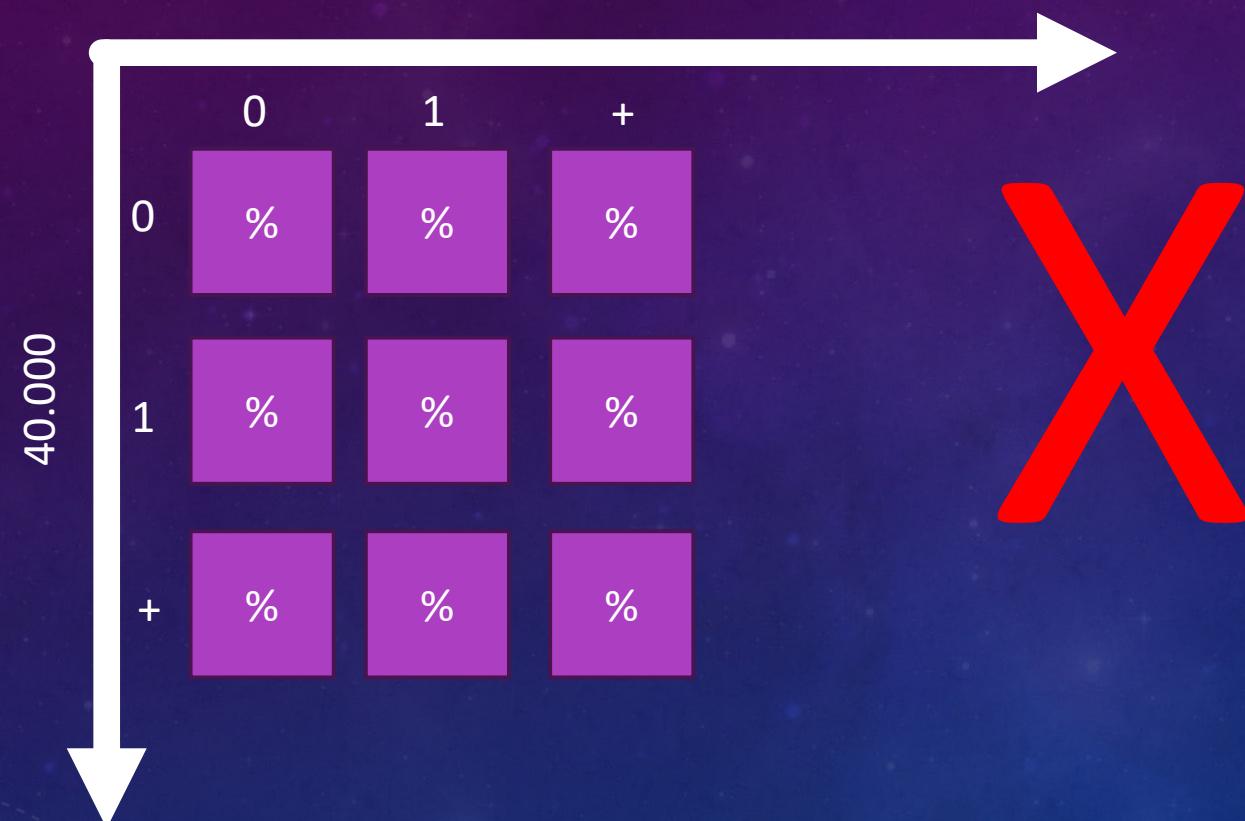


# COMO SERIA O GRÁFICO REPRESENTANDO AS PALAVRAS NO ESPAÇO ?



- Um vocabulário típico pode ter de 30.000 a 40.000 palavras

# PARA QUE SERVE ?



TEMOS 100% DE PROBABILIDADE DE QUE  $0+1$  SEJA IGUAL A 1

NÃO É ADEQUADA PARA TAREFAS PARA AS QUAIS TEMOS UMA HEURÍSTICA CONSOLIDADA

Imagine que você deseja criar uma calculadora  
Para somar números binários

A probabilidade de saírem as combinações

$0+1$

E você deseja saber o próximo número.  
Com sorte, este valor deve ser depois  
do treinamento 90%

# O INVERSO É VERDADEIRO

- Novas heurísticas podem ser criadas a partir de aprendizados obtidos com redes neurais



# VAMOS BRINCAR ?

Apresentando o GlaucoGPT – Uma  
implementação de GPT treinada para  
“falar como o Glauco”

# Obrigado !

## Perguntas

