Universidade de São Paulo
Instituto de Matemática e Estatística
Aluna: Nara Avila Moraes
Nro. USP: 5716734

2o. semestre, 2025
Prova 02 - 05/12

MAE5911/IME: Fundamentos de Estatística e Machine Learning.
Prof.: Alexandre Galvão Patriota

Questão 01: Descreva o mecanismo de atenção com múltiplas cabeças (Multi-head Attention). Apresente o desenvolvimento como feito em sala, considerando a versão "mascarada", para identificar médias ponderadas dinamicamente.

O mecanismo de atenção é o que transforma representação semântica de tokens em representação contextualizada, garantindo que os tokens sejam reinterpretados de acordo com o contexto em que é referido. O mecanismo consiste em durante o treino aprender as matrizes W_Q e W_K que representam parâmetros de *query*, o que este token procura, e *key*, o que este token oferece, tais que

$$Q_i = x_i * W_Q$$

$$K_i = x_i * W_K,$$

no qual, durante o teste, o embedding x de cada token é redefinido para um novo valor utilizando a fórmula

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + M\right) V \;=\; \sum_j A_{ij}\, V_j$$

na qual M é a máscara que apenas atribui contribuição não nula para tokens em relação a si mesmo e aos tokens predecessores, garantindo uma análise causal de contexto e V é a estatística suficiente sobre a qual tomamos a expectativa, também calculado via matriz de parâmetros W_V, tal que $V_i = x_i * W_V$. O resultado final é a matriz Attention, que tem dimensão $n \times d_{model}$, para n tokens e dimensão do embedding d_{model}, no modelo com uma cabeça.

Para o modelo com multiplas cabeças, o *Multi-head Attention*, a dimensão do embedding é dividida em h cabeças, cada uma com dimensão $d_k = d_{model}/h$. Cada cabeça aprende suas próprias matrizes de parâmetros $W_Q^{(i)}$, $W_K^{(i)}$ e $W_V^{(i)}$, para $i = 1, \ldots, h$. O mecanismo de atenção é aplicado separadamente para cada cabeça, resultando em h matrizes de atenção distintas.

Este modelo tem a vantagem de reduzir a dimensionalidade do embedding e permitir a paralelização do cálculo da matriz de atenção, além de permitir que cada cabeça foque em diferentes partes da sequência de entrada x, analisando separadamente diferentes aspectos de relacionamento entre tokens, como por exemplo, semântica, sintaxe, ou interdependências. Essas matrizes são então concatenadas e projetadas de volta para a dimensão original do embedding usando uma matriz de projeção de parâmetros adicional W_O, que combina as matrizes de atenção com pesos diferentes, adicionando uma camada adicional de refinamento do contexto, podendo adicionalmente ser utilizada para reduzir a dimensionalidade do embedding. A fórmula do modelo com multi-head attention é:

$$\text{MultiHead}(Q,K,V) = \text{Concat}\Big(\text{Attention}(QW_1^Q,\, KW_1^K,\, VW_1^V),\; \ldots,\; \text{Attention}(QW_h^Q,\, KW_h^K,\, VW_h^V)\Big)\, W^O.$$

Exemplo do cálculo de *multi-head attention*, partindo dos seguintes blocos W_Q, W_K e W_V de parâmetros de atenção treinados, e $W_O = I_4$ neste exemplo por simplicidade, para $h = 2$:

Head 1

$$W_Q^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, W_K^{(1)} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, W_V^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Head 2

$$W_Q^{(2)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, W_K^{(2)} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, W_V^{(2)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Considere um prompt com $n = 3$ tokens:

$$(\text{Life},\ \text{is},\ \text{awesome})$$

Cada token é representado por um embedding de dimensão $d_{\text{model}} = 4$. Para fins ilustrativos consideramos a seguinte matriz de embeddings:,

$$X = \begin{bmatrix} \text{Life} \\ \text{is} \\ \text{awesome} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{3\times 4}.$$

Cálculo das projecoes Q, K, V para cada token:

$$Q^{(i)} = XW_Q^{(i)}, \quad K^{(i)} = XW_K^{(i)}, \quad V^{(i)} = XW_V^{(i)},$$

Head 1

$$Q^{(1)} = \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}, K^{(1)} = \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 3 \end{bmatrix}, V^{(1)} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \\ 1 & 2 \end{bmatrix}$$

Head 2

$$Q^{(2)} = \begin{bmatrix} 0 & 2 \\ 1 & 1 \\ 2 & 1 \end{bmatrix}, K^{(2)} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 3 & 1 \end{bmatrix}, V^{(2)} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \\ 1 & 2 \end{bmatrix}$$

Calculamos agora, para $d_k = d_{\text{model}}/h = 2$:

$$S^{(i)} = \frac{Q^{(i)} {K^{(i)}}^\top}{\sqrt{d_k}}$$

Head 1

$$S^{(1)} = \frac{1}{\sqrt{2}} \begin{bmatrix} 4 & 2 & 2 \\ 3 & 2 & 4 \\ 4 & 3 & 7 \end{bmatrix}$$

Head 2

$$S^{(2)} = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & 4 & 2 \\ 2 & 3 & 4 \\ 3 & 4 & 7 \end{bmatrix}$$

Aplicamos em seguida a máscara causal, o que impede que o token t seja co-relacionado com tokens em posições subsequêntes (futuras) $j > t$, exatamente igual ocorre no treinamento. Os scores mascarados são obtidos por:

$$\widetilde{S}^{(i)} = S^{(i)} + M, \qquad M = \begin{bmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{bmatrix},$$

Note que a máscara zera as probabilidades das posições não causais:

$$\widetilde{S}^{(1)}_{1,:} = \big(s_{11}, -\infty, -\infty\big) \quad \Longrightarrow \quad \text{softmax}\big(\widetilde{S}^{(1)}_{1,:}\big) = \frac{\big(e^{s_{11}}, e^{-\infty}, e^{-\infty}\big)}{e^{s_{11}} + e^{-\infty} + e^{-\infty}} = (1, 0, 0),$$

Aplicando a função softmax linha a linha aos scores mascarados, obtemos as matrizes $A^{(1)}$ e $A^{(2)}$.

Head 1

$$A^{(1)} = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ 0.670 & 0.330 & 0.000 \\ 0.102 & 0.050 & 0.848 \end{bmatrix}$$

Head 2

$$A^{(2)} = \begin{bmatrix} 1.000 & 0.000 & 0.000 \\ 0.330 & 0.670 & 0.000 \\ 0.050 & 0.102 & 0.848 \end{bmatrix}$$

A saída da cabeça i é

$$\text{head}_i = A^{(i)} V^{(i)} \in \mathbb{R}^{3\times 2}.$$

Head 1

$$A^{(1)}V^{(1)} =\simeq \begin{bmatrix} 2.000 & 1.000 \\ 1.670 & 1.330 \\ 1.102 & 1.898 \end{bmatrix}$$

Head 2

$$A^{(2)}V^{(2)} =\simeq \begin{bmatrix} 1.000 & 1.000 \\ 1.670 & 0.330 \\ 1.102 & 1.747 \end{bmatrix}$$

Considerando, por simplicidade, a projeção de saída como a identidade,

$$W^O = I_4,$$

a saída final do bloco de *multi-head attention* é

$$\text{MultiHead}(Q, K, V) = \text{Concat}(H^{(1)}|H^{(2)})W_O \in \mathbb{R}^{3\times 4}.$$

Multi-Head Attention

$$MultiHead(Q, K, V) = \begin{bmatrix} 2.000 & 1.000 & 1.000 & 1.000 \\ 1.670 & 1.330 & 1.670 & 0.330 \\ 1.102 & 1.898 & 1.102 & 1.747 \end{bmatrix} \begin{bmatrix} 1.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 1.000 & 0.000 & 0.000 \\ 0.000 & 0.000 & 1.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 1.000 \end{bmatrix}$$

Resultado final:

Multi-Head Attention (novos *embeddings* contextualizados)

$$Attention(Q, K, V) = \begin{bmatrix} \text{Life} \\ \text{is} \\ \text{awesome} \end{bmatrix} = \begin{bmatrix} 2.000 & 1.000 & 1.000 & 1.000 \\ 1.670 & 1.330 & 1.670 & 0.330 \\ 1.102 & 1.898 & 1.102 & 1.747 \end{bmatrix}$$

```
# ------------------------------------------------------------------------------------------------------------
# Cada camada N da rede neural receberá um módulo de atenção multi-cabeças definido da seguinte forma:
# ------------------------------------------------------------------------------------------------------------
self$MM <- torch::nn_module_list(lapply(1:N_Layers,
  function(x) torch::nn_multihead_attention(
    n_embd,               # d_model: dimensão do vetores de embedding
    N_Head,               # h: número de cabeças
    dropout = p0,         # tecnica para reduzir o overfitting
    batch_first = TRUE))) # informa a disposição dos dados à biblioteca torch
```

Listing 1: **Definição** dos blocos de multi-head attention treináveis

```
# ------------------------------------------------------------------------------------------------------------
# Cria uma matriz TxT, sendo T = número de tokens na sequência de entrada
# Garante que será tratado no mesmo device que X (CPU ou GPU) para que eles possam ser operados juntos
# Aplica 1 à diagonal superior, transforma 1 em booleano TRUE para que a biblioteca torch saiba quais
# posições precisam ser mascaradas
# ------------------------------------------------------------------------------------------------------------
wei <- torch::torch_triu(
  torch::torch_ones(T, T, device = x$device),
  diagonal = 1)$to(dtype = torch::torch_bool())
```

Listing 2: **Definição** da Máscara causal

```
for (j in 1:self$N) {

  # ------------------------------------------------------------------------------------------------------------
  # 1) Pré-normalização (LayerNorm)
  #    Normalização dos embeddings, já que todos embeddings interagirão com todos os outros.
  #    A normalização não altera a informação e estabiliza o treinamento.
  # ------------------------------------------------------------------------------------------------------------
```

```
  QKV <- self$scale1[[j]](output)

  # ------------------------------------------------------------------------------------------------------------
  # 2) Chamada da função Multi-Head Self-Attention mascarada do torch que calcula Q, K e V internamente a
    partir de QKV.
  # ------------------------------------------------------------------------------------------------------------
  attn_out <- self$MM[[j]](
    query      = QKV,     # Input para o cálculo de Q
    key        = QKV,     # Input para o cálculo de K
    value      = QKV,     # Input para o cálculo de V
    attn_mask  = wei,     # Máscara causal
    need_weights = FALSE  # Dispensa o retorno dos pesos utilizados no cáculo da média ponderada
  )[[1]]                  # Guarda apenas o resultado da média ponderada, mas não os pesos

  # ------------------------------------------------------------------------------------------------------------
  # 3) Conexão residual após atenção
  # Mantém a informação original dos embeddings e evita que a transformação da atenção distorça
    excessivamente a representação.
  # Estabiliza o fluxo do gradiente.
  # ------------------------------------------------------------------------------------------------------------
  output <- output + attn_out

  # ------------------------------------------------------------------------------------------------------------
  # 4) Feed-Forward Network + residual
  #  Aplicação de uma MLP ponto-a-ponto em cada token individualmente, capturando relações não lineares
    permitindo que a rede aprenda padrões complexos.
  #  Mantendo o resíduo e aplicando a normalização para estabilizar a operação.
  # ------------------------------------------------------------------------------------------------------------
  output <- output + self$FFN[[j]]( self$scale2[[j]](output) )
}
```

Listing 3: **Cálculo** da atenção multi-cabeças mascarada em cada camada

Questão 02: Treine um modelo de linguagem com dimensão embedding de 128, duas camadas e duas cabeças para os dados de Shakespeare.

Descrição do experimento e configuração do modelo

Corpus utilizado

O modelo foi treinado sobre um corpus textual extraído da edição completa das obras de William Shakespeare disponibilizada pelo *Project Gutenberg*, um repositório digital que distribui gratuitamente obras que já se encontram em domínio público, contendo todo o conteúdo produzido pelo autor: peças teatrais, poemas narrativos, sonetos, trechos introdutórios e notas editoriais presentes na edição digital, com aproximadamente 5,4 milhões de caracteres.

```
DUKE.
So that, from point to point, now have you heard
The fundamental reasons of this war,
Whose great decision hath much blood let forth,
And more thirsts after.

FIRST LORD.
Holy seems the quarrel
Upon your Grace's part; black and fearful
On the opposer.

DUKE.
Therefore we marvel much our cousin France
Would, in so just a business, shut his bosom
Against our borrowing prayers.
```

```
MENECRATES.
We, ignorant of ourselves,
Beg often our own harms, which the wise powers
Deny us for our good; so find we profit
By losing of our prayers.

POMPEY.
I shall do well.
The people love me, and the sea is mine;
My powers are crescent, and my auguring hope
Says it will come to th' full. Mark Antony
In Egypt sits at dinner, and will make
No wars without doors. Caesar gets money where
He loses hearts. Lepidus flatters both,
Of both is flattered; but he neither loves
Nor either cares for him.
```

Figura 1: Trechos do corpus utilizado no experimento, provenientes da obra completa de Shakespeare disponibilizado pelo Projeto Gutenberg.

Vocabulário extraído do corpus

O modelo foi treinado no regime *character-level*, o primeiro passo consistiu em extrair todos os caracteres distintos presentes no arquivo `Shakespeare.txt`. Inclui letras maiúsculas e minúsculas, dígitos, pontuação, símbolos especiais, acentos, quebras de linha e espaços.

O vocabulário final contém **109 tokens**, sendo o primeiro reservado para o símbolo especial `<PAD>` utilizado em operações internas. Segue o conjunto de caracteres identificados:

```
  [1] "<PAD>" ""      "\t"    "\n"    "\r"    " "     "_"     "-"     "—"     ","     ";"     ":"
 [13] "!"     "?"     "."     "…"     "'"     "‘"     "’"     "“"     "”"     "("     ")"     "["
 [25] "]"     "*"     "/"     "&"     "#"     "%"     "•"     "$"     "0"     "1"     "2"     "3"
 [37] "4"     "5"     "6"     "7"     "8"     "9"     "a"     "A"     "à"     "À"     "â"     "æ"
 [49] "Æ"     "b"     "B"     "c"     "C"     "ç"     "Ç"     "d"     "D"     "e"     "E"     "é"
 [61] "É"     "è"     "ê"     "ë"     "f"     "F"     "g"     "G"     "h"     "H"     "i"     "I"
 [73] "î"     "j"     "J"     "k"     "K"     "l"     "L"     "m"     "M"     "n"     "N"     "o"
 [85] "O"     "œ"     "p"     "P"     "q"     "Q"     "r"     "R"     "s"     "S"     "t"     "T"
 [97] "™"     "u"     "U"     "v"     "V"     "w"     "W"     "x"     "X"     "y"     "Y"     "z"
[109] "Z"
```

Figura 2: Vocabulário extraido no corpus utilizado (obra completa de Shakespeare).

Hiperparâmetros do modelo

Os hiperparâmetros utilizados no modelo são:

Tabela 1: Hiperparâmetros utilizados no modelo GPT treinado.

Hiperparâmetro	Valor	Descrição
`block_size`	50	contexto máximo (janela do attention).
`n_embd`	128	dimensão dos embeddings.
`N_Layers`	2	número de blocos Transformer.
`N_Head`	2	número de cabeças de atenção.
`dropout`	0.2	regularização.
`learning_rate`	0.003	taxa de aprendizado.
`batch_size`	64	tamanho do mini-batch.
`epochs`	1200	número total de épocas.

O tamanho de contexto (`block_size = 50`) limita o alcance da atenção foi ajustado empiricamente para conseguir captar o estilo do texto e ao mesmo tempo ser viável para ser treinado em hardware pessoal. A dimensão dos vetores de embedding (`n_embd = 128`) é significativo, mas combinada ao número reduzido de camadas e a utilização de multiplas cabeças de atenção (`N_Layers = 2`, `N_Head = 2`), mantém o modelo ágil e compacto, suficiente para capturar regularidades sintáticas e estilísticas do texto. O dropout (`0.2`) fornece regularização. A taxa de aprendizado escolhida (`0.003`) foi ajustada empiricamente.

Arquitetura e fluxo do modelo

A arquitetura utilizada neste trabalho segue o padrão dos modelos GPT do tipo *decoder-only*, cujo fluxo computacional está ilustrado na Fig. 3. O processamento inicia-se pela soma entre o embedding dos tokens e o embedding posicional aprendido, resultando em uma representação contínua que preserva a ordem sequencial do texto. Em seguida, cada camada Transformer aplica uma normalização prévia (*LayerNorm*) seguida de um bloco de autoatenção multi-cabeças mascarada, garantindo o comportamento autoregressivo: o modelo só pode utilizar informações de tokens passados para prever o próximo token.

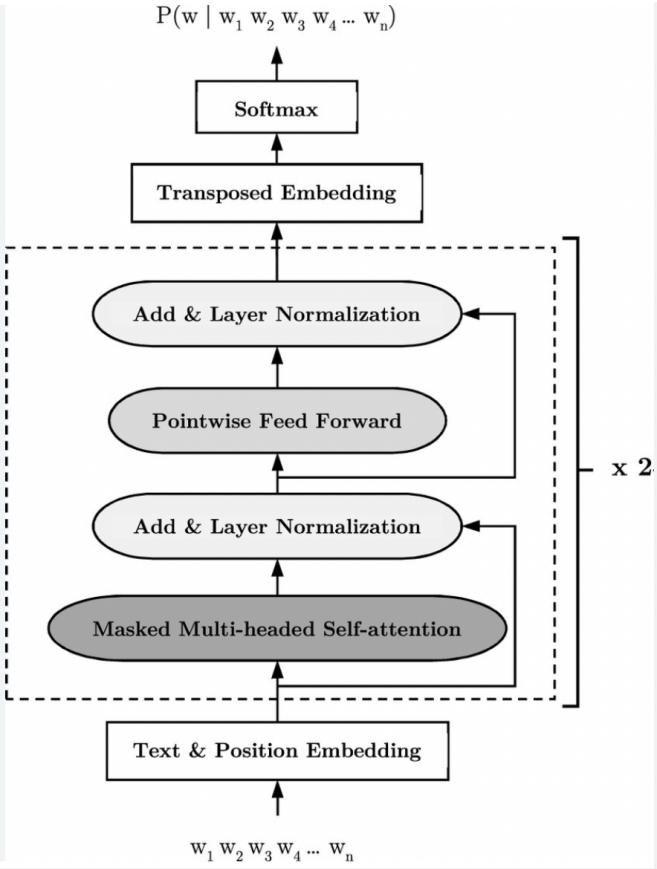

Figura 3: Arquitetura geral do modelo GPT utilizado no experimento.

O resultado da atenção é então somado à entrada original por meio de uma conexão residual, estabilizando o fluxo de gradientes. Após isso, uma nova normalização é aplicada antes do bloco *feed-forward* ponto-a-ponto, responsável por expandir e recomprimir a dimensionalidade interna, permitindo ao modelo capturar não-linearidades locais. Esse bloco também é seguido por uma conexão residual, completando a estrutura *pré-norm* típica de arquiteturas modernas de Transformers. Ao final das camadas empilhadas (2) uma projeção linear — equivalente ao uso do embedding transposto — produz os logits sobre o vocabulário, posteriormente convertidos em probabilidades via *softmax*, que indica o próximo token mais provável.

Tamanho do modelo

O modelo GPT implementado apresenta um total de **418 866 parâmetros** treináveis, distribuídos entre embeddings, projeções lineares das camadas de atenção, normalizações e redes feed-forward. A Tabela 2 apresenta o detalhamento dos tensores de parâmetros que compõem o modelo.

Parâmetro	Dimensão	Nº de parâmetros
wpe.weight	50 × 128	6 400
wte.weight	109 × 128	13 952
MM.0.in_proj_weight	384 × 128	49 152
MM.0.in_proj_bias	384	384
MM.0.out_proj_weight	128 × 128	16 384
MM.0.out_proj_bias	128	128
MM.1.in_proj_weight	384 × 128	49 152
MM.1.in_proj_bias	384	384
MM.1.out_proj_weight	128 × 128	16 384
MM.1.out_proj_bias	128	128
scale1.{0,1}.weight	128	128 + 128
scale1.{0,1}.bias	128	128 + 128
scale2.{0,1}.weight	128	128 + 128
scale2.{0,1}.bias	128	128 + 128
scale3.weight	128	128
scale3.bias	128	128
FFN.0.{0,2}.weight	512 × 128	65 536 + 65 536
FFN.0.{0,2}.bias	512	512 + 128
FFN.1.{0,2}.weight	512 × 128	65 536 + 65 536
FFN.1.{0,2}.bias	512	512 + 128
ln_f.weight	109 × 128	13 952
Total	—	**418 866**

Tabela 2: Parâmetros treináveis do modelo GPT utilizado no experimento.

Treinamento e métricas

No treinamento, foi utilizada a função de perda *cross-entropy*, computada entre os logits produzidos pelo modelo e os caracteres-alvo da sequência. A cada época, foi registrado tanto a perda de treino quanto a de teste, permitindo acompanhar a evolução do aprendizado e o possível surgimento de sobreajuste.

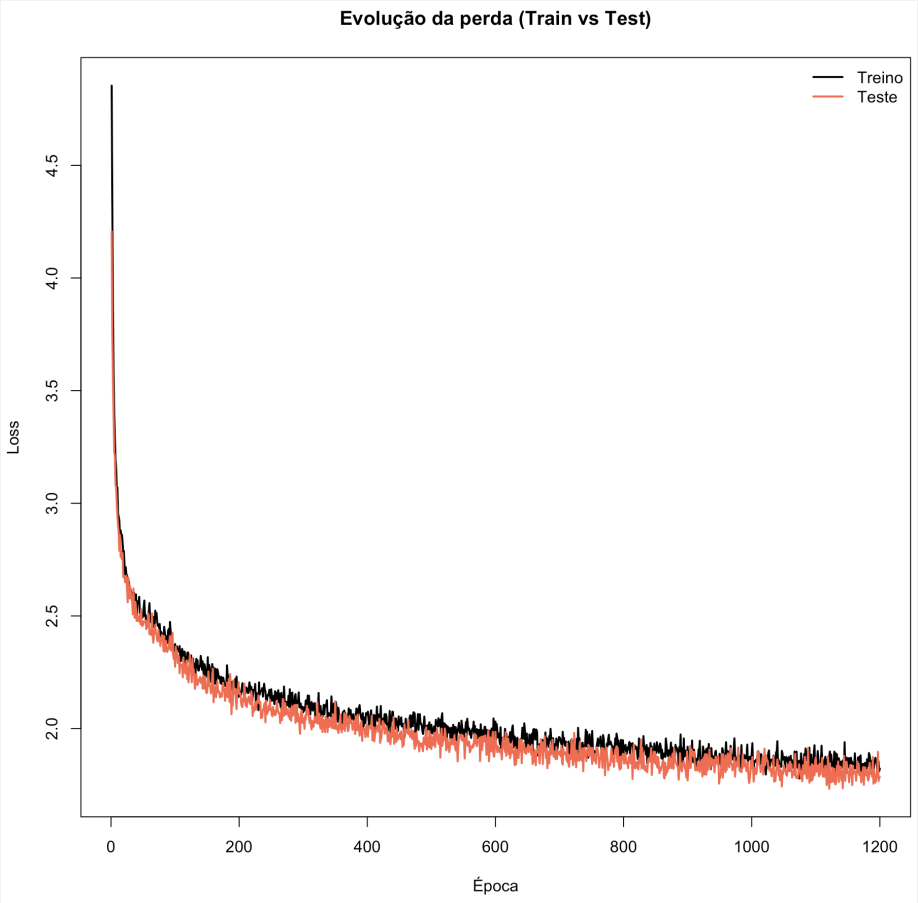

Figura 4: Evolução da função de perda durante o treinamento: comparação entre as curvas de treino e de teste ao longo das 1200 épocas.

As curvas de perda apresentaram comportamento regular ao longo das épocas: as perdas de treino e de teste permaneceram próximas, decrescendo de forma suave sem indícios de divergência. Essa proximidade entre as curvas indica ausência de sobreajuste, sugerindo que o modelo conseguiu aprender padrões relevantes do corpus sem memorizar o conjunto de treinamento. Além disso, a estabilidade observada nas últimas épocas reflete um processo de otimização bem-sucedido.

O valor final da perda obtido após 1200 épocas foi de aproximadamente `1.82` para o conjunto de treino e `1.78` para o conjunto de teste.

Resultado

Os resultados obtidos mostram que o modelo, apesar de compacto, aprendeu *padrões estatísticos* presentes no inglês shakespeariano, como a distribuição típica de comprimentos de palavras, certa cadência rítmica, alternância entre diálogos e rubricas, além da reprodução aproximada de construções sintáticas características do período.

Entretanto, o texto gerado permanece semanticamente incoerente, apresentando repetições excessivas e sentenças circulares. Esse comportamento é esperado para um modelo de pequena escala: ele captura a superfície do estilo, mas não reproduz a coerência e os encadeamentos semânticos ao longo do texto. Uma sugestão para testes futuros é que um processo de tokenização mais robusto poderá contribuir para melhorar a coerencia do texto.

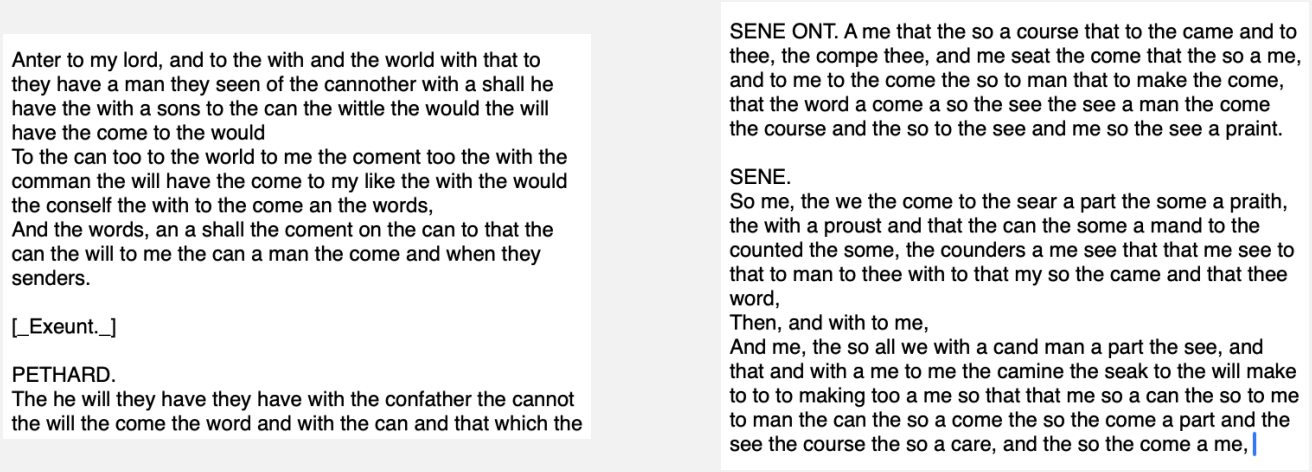

Figura 5: Trechos do texto gerado pelo modelo treinado.

O experimento valida a implementação completa de um Transformer do tipo *decoder-only*, confirmando que mesmo arquiteturas reduzidas conseguem capturar características linguísticas relevantes quando treinadas sobre um corpus particular. O modelo foi capaz de replicar com consistência o *ritmo*, o *fluxo* e a *forma* geral do estilo shakespeariano.

Questão 03: Repita o item anterior sem o mecanismo de atenção (com múltiplas cabeças) e descreva o que ocorre no treinamento.

Removendo o mecanismo de atenção, cada camada passa a consistir em uma rede *feed-forward* aplicada de forma autoregressiva sobre uma janela fixa de contexto atuando independentemente a cada token, utilizando o embedding original, isto é, sem identificação de relevância e correlação de contexto entre tokens.

A Fig. 6 mostra a evolução da perda durante o treinamento. Embora a perda apresente uma redução inicial significativa, o seu valor estabiliza em um platô mais alto do que no modelo com o Attention. Não há overfitting, porém as curvas de treino estabilizadas com uma loss alta demonstram a incapacidade do modelo de progredir para regiões mais ricas da função de custo. A loss alta é evidência da perda da capacidade de capturar padrões complexos no texto.

O valor final da perda obtido após 1200 épocas foi de aproximadamente `2.41` para o conjunto de treino e `2.44` para o conjunto de teste.

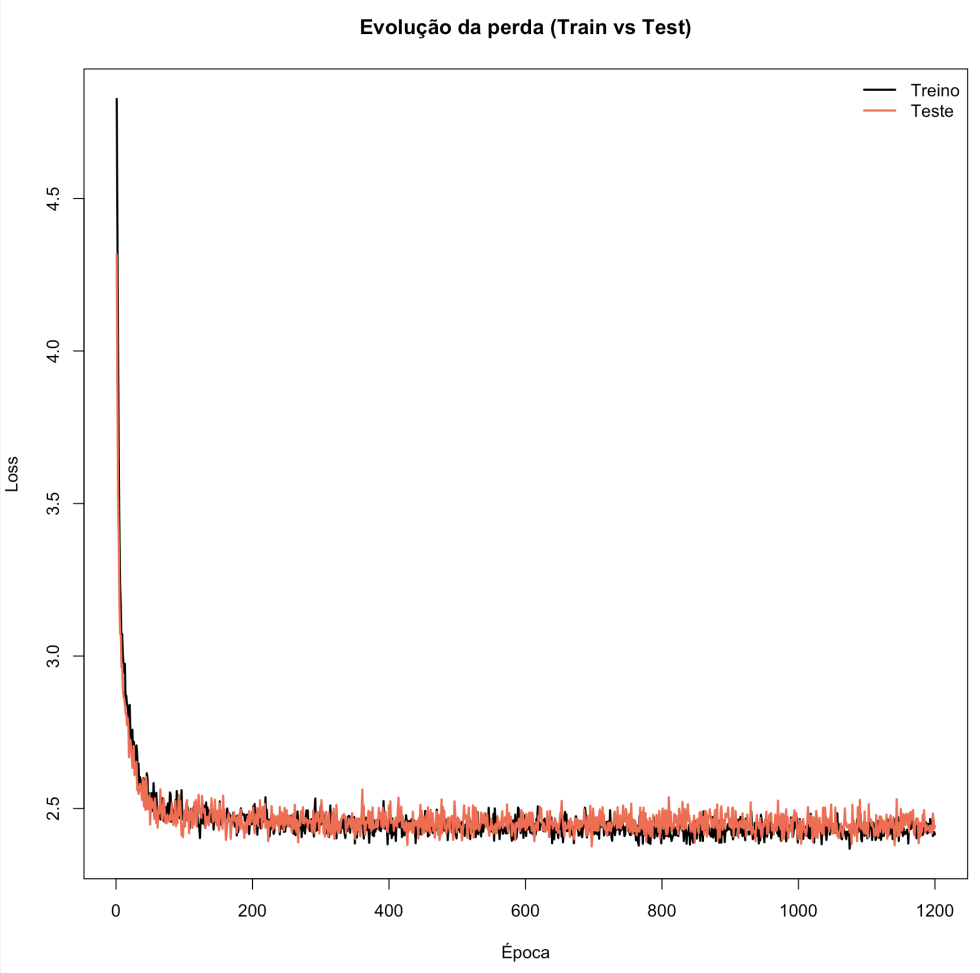

Figura 6: Curva de perda do modelo sem o mecanismo de atenção.

Os efeitos desta remoção é evidente no texto gerado como resultado, o qual manteve o seu aspecto de linguagem, porém sem reproduzir o estilo ou a variedade de palavras do corpus utilizado.

```
And than t t mer t the the the st the man thand mand than man t mer thand t that t than t the the that
hande mand mand t me the t me thand the t the mere therere than t therer mathathat there the t t t tha
nd mathan me ther merer the t mathathe me me there the mer the me mer me than the mand t mand me man m
ere t the me me thathe t t the mere t the me thathat me thand t thande thather t the thande mand t man
d t me ther me there ther me mere ther thand mere thathat t me than the me the mat t t the the t merer
me t the the that me ther mat t mather the than the me me t the t the than me t there ther mer t the m
e man thand me me there me t t me t me ther mer mererer than than me ther the than mat me t mand t the
me me me mere t t mathe mat the ther t than t thande the t the me the me mer man therer t than that me
t thathathat t the me t the t thand the mathe t man man t t the the me me thathe t t than the me than
mand the t the than merere mere the mer there man the me thatherer the t t mand t me man t the me than
d t t t t the the t the t me me mer t t the than the me ther mand the t the t thand t mer mat than tha
t me t mathe mathe t man mand t mande t mande t t t mer thande me t the t thand mere t mer t the thand
mere mathe that the t the the me the t me me me t thand the man the the the t there mer t t mathand th
e t me ther thand mand t the than thather me t ther mat the therer mat the t t me t mand than the me t
he the therer me the the t ther t mathe than mat there the me man the
```

Figura 7: Amostra de texto gerado pelo modelo treinado sem o bloco de atenção.

Anexo

```r
config <- list(
  file_name   = "Shakespeare.txt",
  train       = !TRUE,
  run         = TRUE,
  read_weights = !TRUE,
  p_train     = 0.8,
```

```
  k_top       = 2,

  block_size = 50,    # Maximum context
  n_embd     = 128,   # Embedding dimension
  N_Layers   = 2,     # Number of layers
  N_Head     = 2,     # Number of heads

  lr          = 0.003, # Learning rate
  batch_size  = 64,    # Batch size
  p0          = 0.2,   # Dropout proportion
  epochs      = 1200,  # Number of epochs
  num_workers = 6,     # Number of CPU workers

  max_new_tokens = 1500
)
```

Listing 4: Configurações do experimento.

```
library(torch)

GPT <- torch::nn_module(
  initialize = function(block_size, n_embd, N_Layers, nvoc, N_Head, p0 = 0.1) {

    self$N   <- N_Layers
    self$wpe <- torch::nn_embedding(block_size, n_embd)
    self$wte <- torch::nn_embedding(nvoc, n_embd, padding_idx = 1)

    self$MM  <- torch::nn_module_list(lapply(1:N_Layers,
      function(x) torch::nn_multihead_attention(
        n_embd, N_Head, dropout = p0, batch_first = TRUE)))

    self$scale1 <- torch::nn_module_list(lapply(1:N_Layers,
      function(x) torch::nn_layer_norm(n_embd)))

    self$scale2 <- torch::nn_module_list(lapply(1:N_Layers,
      function(x) torch::nn_layer_norm(n_embd)))

    self$scale3 <- torch::nn_layer_norm(n_embd, elementwise_affine = TRUE)

    self$FFN <- torch::nn_module_list(lapply(1:N_Layers,
      function(x) {
        torch::nn_sequential(
          torch::nn_linear(n_embd, 4 * n_embd),
          torch::nn_gelu(),
          torch::nn_linear(4 * n_embd, n_embd),
          torch::nn_dropout(p0))}))

    self$ln_f  <- torch::nn_linear(n_embd, nvoc, bias = FALSE)
    self$drop0 <- torch::nn_dropout(p = p0)
  },

  forward = function(x, return_intermediates = FALSE) {
    # x: [B, T]
    B <- x$size(1)
    T <- x$size(2)

    x1 <- torch::torch_arange(
      1, T, dtype = torch::torch_long(), device = x$device
    )

    wei <- torch::torch_triu(
      torch::torch_ones(T, T, device = x$device),
      diagonal = 1)$to(dtype = torch::torch_bool())

    output <- self$wte(x) + self$wpe(x1)$unsqueeze(1)  # [B, T, E]
    output <- self$drop0(output)

    for (j in 1:self$N) {
      QKV <- self$scale1[[j]](output)
```

```
      attn_out <- self$MM[[j]](
        query = QKV, key = QKV, value = QKV,
        attn_mask = wei, need_weights = FALSE
      )[[1]]
      output <- output + attn_out
      output <- output + self$FFN[[j]](self$scale2[[j]](output))
    }

    output <- self$scale3(output)
    logits <- self$ln_f(output)

    if (return_intermediates) {
      return(list(
        x1     = x1$cpu(),
        wei    = wei$to(dtype = torch_int())$cpu(),
        out    = output$cpu(),
        logits = logits$cpu()))}
    logits
  }
)
```

Listing 5: Implementação do modelo.

```
source("Config.R")
source("GPT.R")
source("Generator.R")

library(torch)
library(cli)

file0 <- base::readChar(
  config$file_name,
  file.info(config$file_name)$size
)

voc     <- c("<PAD>", sort(unique(unlist(strsplit(file0, "")))))
Encoded <- Encoder(file0, voc)
nvoc    <- length(voc)

n       <- length(Encoded)
p_train <- config$p_train
n_train <- round(p_train * n)

BD.train <- torch_tensor(Encoded[1:n_train], dtype = torch_int())
BD.test  <- torch_tensor(Encoded[(n_train + 1):n], dtype = torch_int())

n_test_total <- BD.test$size()[1]

Model <- GPT(
  block_size = config$block_size,
  n_embd     = config$n_embd,
  N_Layers   = config$N_Layers,
  nvoc       = nvoc,
  N_Head     = config$N_Head,
  p0         = config$p0
)

optimizer <- torch::optim_adamw(Model$parameters, lr = config$lr)
loss_fn   <- torch::nn_cross_entropy_loss()

loss_store      <- numeric(config$epochs)
loss_store_test <- numeric(config$epochs)

batch_train <- config$batch_size
batch_test  <- config$batch_size

for (ep in 1:config$epochs) {

  idx <- sample(
```

```
    1:(n_train - config$block_size - 1),
    batch_train
  )

  idx2 <- as.integer(c(
    t(outer(as.integer(idx), 0:config$block_size, `+`))
  ))

  Z <- BD.train[idx2, drop = FALSE]$view(
    c(length(idx), config$block_size + 1)
  )
  X <- Z[, 1:config$block_size]
  Y <- Z[, 2:(config$block_size + 1)]

  FIT  <- Model$train()(X)
  loss <- loss_fn(FIT$flatten(end_dim = 2), Y$flatten())

  optimizer$zero_grad()
  loss$backward()
  optimizer$step()

  loss_store[ep] <- loss$item()

  with_no_grad({
    idx_t <- sample(
      1:(n_test_total - config$block_size - 1),
      batch_test
    )

    idx2_t <- as.integer(c(
      t(outer(as.integer(idx_t), 0:config$block_size, `+`))
    ))

    Zt <- BD.test[idx2_t, drop = FALSE]$view(
      c(length(idx_t), config$block_size + 1)
    )

    X_test <- Zt[, 1:config$block_size]
    Y_test <- Zt[, 2:(config$block_size + 1)]

    logits_test <- Model$eval()(X_test)
    Lte <- loss_fn(
      logits_test$flatten(end_dim = 2),
      Y_test$flatten()
    )
  })

  loss_store_test[ep] <- Lte$item()

  cli::cli_progress_message(
    paste0(
      "Época: ", ep,
      " | Train loss: ", round(loss_store[ep], 4),
      " | Test loss: ",  round(loss_store_test[ep], 4)
    )
  )

  if (ep %% 10 == 0) {

    ylim_range <- range(
      c(loss_store[1:ep], loss_store_test[1:ep]),
      na.rm = TRUE
    )

    plot(
      1:ep, loss_store[1:ep],
      type = "l",
      lwd  = 2,
```

```
      col  = "black",
      xlab = "Época",
      ylab = "Loss",
      main = "Evolução da perda (Train vs Test)",
      ylim = ylim_range
    )

    lines(
      1:ep, loss_store_test[1:ep],
      col  = "tomato",
      lwd  = 2
    )

    legend(
      "topright",
      legend = c("Treino", "Teste"),
      col = c("black", "tomato"),
      lwd = 2,
      bty = "n"
    )
  }
}
```

Listing 6: Script de treinamento do modelo.

```
Encoder <- function(file, vocabulary) {
  file  <- unlist(strsplit(file, ""))
  filex <- numeric(length(file))
  for (i in seq_along(vocabulary)) {
    filex[file == vocabulary[i]] <- i
  }
  filex
}

Decoder <- function(file, vocabulary) {
  filex <- file
  for (i in seq_along(vocabulary)) {
    filex[file == i] <- vocabulary[i]
  }
  filex
}

generate_topk <- function(prompt, k_top = config$k_top, max_new = config$max_new_tokens) {
  Model$eval()
  with_no_grad({
    x <- torch_tensor(Encoder(prompt, voc), dtype = torch_int())$unsqueeze(1)

    for (i in 1:max_new) {
      if (x$size(2) <= config$block_size) {
        logits <- Model$eval()(x)[, -1, ]
      } else {
        xx     <- x[, (x$size(2) - config$block_size + 1):x$size(2)]
        logits <- Model$eval()(xx)[, -1, ]
      }

      top   <- logits$topk(k_top)
      vals  <- top[[1]]$to(dtype = torch_float())
      probs <- torch::nnf_softmax(vals, dim = -1)
      selected   <- torch_multinomial(probs, num_samples = 1)
      next_token <- top[[2]][, selected$item()]$unsqueeze(1)
      x <- torch_cat(list(x, next_token), dim = 2)
    }

    generated_idx <- as.integer(as_array(x$squeeze(1)))
    paste(voc[generated_idx], collapse = "")
  })
}
```

Listing 7: Funções de codificação e geração de texto com top-k sampling.

```r
source("Config.R")
source("GPT.R")
source("Generator.R")

prompt <- "A"
cat(generate_topk(prompt), "\n")
```

Listing 8: Script de geração de texto a partir de um prompt inicial.