UNIVERSIDADE DE SÃO PAULO
Programa de Pos-Graduação em Estatística

Alex Monito Nhancololo

1 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.

Resolução

Os modelos de linguagem baseiam-se na interpretação probabilística de sequências. Conforme descrito por Patriota [2025], a linguagem é definida como um sistema composicional de sinais e tratada estatisticamente como um processo estocástico gerador de dados.

Seja $\{X_t\}_{t\geq 1}$ um processo estocástico discreto assumindo valores em um vocabulário finito $\mathcal{V}$ de tamanho N_{voc}. O objetivo da modelagem, conforme Patriota [2024], é estimar a distribuição de probabilidade conjunta de uma sequência de tokens observados $\boldsymbol{x} = (x_1, \ldots, x_k)$. Pela regra da cadeia, decompomos a probabilidade conjunta no produto das probabilidades condicionais:

$$P(X_1 = x_1, \ldots, X_k = x_k) = \prod_{t=1}^{k} P(X_t = x_t \mid X_1 = x_1, \ldots, X_{t-1} = x_{t-1}).$$

Assumimos que a distribuição condicional de cada token segue uma distribuição **Multinomial** (ou Categórica) condicionada ao histórico, cujos parâmetros são estimados por uma função parametrizada $\mathbf{p}_\theta^{(t)}(x_{1:t-1})$. O mecanismo de *Atenção Multi-Head Mascarada* é o componente funcional do Transformer [Vaswani et al., 2017], responsável por aproximar a dependência temporal complexa contida nesse histórico, atuando como um estimador não-paramétrico de médias ponderadas.

Espaço de entrada e parametrização

Seja a sequência de entrada representada pela matriz $\boldsymbol{X} \in \mathbb{R}^{k\times d_{model}}$:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^\top \\ \boldsymbol{x}_2^\top \\ \vdots \\ \boldsymbol{x}_k^\top \end{bmatrix},$$

onde k é o comprimento do contexto e d_{model} a dimensão do espaço de *embedding*. A i-ésima linha, $\boldsymbol{x}_i^\top \in \mathbb{R}^{1\times d_{model}}$, representa o vetor de *embedding* do token na posição i, somado à sua codificação posicional para preservar a ordem sequencial.

Para capturar múltiplos subespaços de dependência (e.g., sintática, semântica) simultaneamente, definimos H cabeças de atenção. Para uma cabeça arbitrária $h \in \{1, \ldots, H\}$, definimos as seguintes matrizes de projeção linear (parâmetros a serem estimados via Máxima Verossimilhança):

- $U_Q^{(h)} \in \mathbb{R}^{d_{model}\times d_k}$: Matriz de projeção para Queries (Consultas).
- $U_K^{(h)} \in \mathbb{R}^{d_{model}\times d_k}$: Matriz de projeção para Keys (Chaves).
- $U_V^{(h)} \in \mathbb{R}^{d_{model}\times d_v}$: Matriz de projeção para Values (Valores).

Assume-se $d_k = d_v = d_{model}/H$.

Single-head Attention

A entrada $\boldsymbol{X}$ é projetada nos três subespaços latentes para a cabeça h:

$$Q^{(h)} = \boldsymbol{X}U_Q^{(h)}, \quad K^{(h)} = \boldsymbol{X}U_K^{(h)}, \quad V^{(h)} = \boldsymbol{X}U_V^{(h)},$$

com $(k \times d_{model}) \cdot (d_{model} \times d_k) \to (k \times d_k)$.

A relevância entre o token na posição i (query) e o token na posição j (key) é mensurada pelo produto interno (covariância não centrada). A matriz de escores normalizada $S \in \mathbb{R}^{k\times k}$ é dada por:

$$S = \frac{Q^{(h)}K^{(h)\top}}{\sqrt{d_k}} = \frac{(\boldsymbol{X}U_Q^{(h)})(\boldsymbol{X}U_K^{(h)})^\top}{\sqrt{d_k}} = \frac{1}{\sqrt{d_k}}\begin{bmatrix} \boldsymbol{q}_1\boldsymbol{k}_1^\top & \boldsymbol{q}_1\boldsymbol{k}_2^\top & \dots & \boldsymbol{q}_1\boldsymbol{k}_k^\top \\ \boldsymbol{q}_2\boldsymbol{k}_1^\top & \boldsymbol{q}_2\boldsymbol{k}_2^\top & \dots & \boldsymbol{q}_2\boldsymbol{k}_k^\top \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{q}_k\boldsymbol{k}_1^\top & \boldsymbol{q}_k\boldsymbol{k}_2^\top & \dots & \boldsymbol{q}_k\boldsymbol{k}_k^\top \end{bmatrix},$$

onde o elemento escalar (i, j) é dado por $S_{ij} = \frac{\boldsymbol{q}_i\boldsymbol{k}_j^\top}{\sqrt{d_k}}$, sendo $\boldsymbol{q}_i$ a i-ésima linha de Q e $\boldsymbol{k}_j$ a j-ésima linha de K.
O termo $\frac{1}{\sqrt{d_k}}$ é introduzido para estabilizar a variância Vaswani et al. [2017]. Assumindo que os componentes dos vetores projetados são variáveis aleatórias i.i.d. com média 0 e variância 1, o produto interno teria variância d_k. A divisão normaliza a variância para 1, evitando a saturação da função Softmax e facilitando a otimização Vaswani et al. [2017].

Mascaramento

Para que o modelo seja um estimador válido da probabilidade condicional $P(X_t|X_{<t})$, a representação do token i não pode conter informação dos tokens futuros $j > i$. Define-se a matriz de máscara $\boldsymbol{M} \in \mathbb{R}^{k\times k}$ como:

$$M_{ij} = \begin{cases} 0 & \text{se } j \le i \quad \text{(passado e presente)} \\ -\infty & \text{se } j > i \quad \text{(futuro proibido)} \end{cases} \implies \boldsymbol{M} = \begin{bmatrix} 0 & -\infty & \dots & -\infty \\ 0 & 0 & \dots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix}.$$

A matriz de escores mascarada é dada pela operação aditiva $E = S + \boldsymbol{M}$.

Normalização e matriz de pesos

Aplicamos a função Softmax linha a linha sobre E para obter a matriz de pesos estocásticos $\boldsymbol{W}^{(h)} \in \mathbb{R}^{k\times k}$. A soma $S + \boldsymbol{M}$ garante que, ao aplicarmos a exponencial, os pesos para o futuro sejam anulados ($\lim_{z\to-\infty} e^z = 0$). Assim, temos:

$$\boldsymbol{W}^{(h)} = \text{softmax}\left(\frac{\boldsymbol{X}U_Q^{(h)}U_K^{(h)\top}\boldsymbol{X}^\top}{\sqrt{d_k}} + \boldsymbol{M}\right) = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ w_{21}^{(h)} & w_{22}^{(h)} & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ w_{k1}^{(h)} & w_{k2}^{(h)} & \cdots & w_{kk}^{(h)} \end{pmatrix},$$

onde o elemento $w_{ij}^{(h)}(\boldsymbol{X})$ representa o peso que a posição i atribui à posição j na cabeça h:

$$w_{ij}^{(h)}(\boldsymbol{X}) = \frac{\exp\left(\frac{\boldsymbol{q}_i\boldsymbol{k}_j^\top}{\sqrt{d_k}} + M_{ij}\right)}{\sum_{l=1}^{k}\exp\left(\frac{\boldsymbol{q}_i\boldsymbol{k}_l^\top}{\sqrt{d_k}} + M_{il}\right)} = \begin{cases} \dfrac{\exp\left(\frac{\boldsymbol{x}_i^\top U_Q^{(h)}U_K^{(h)\top}\boldsymbol{x}_j}{\sqrt{d_k}}\right)}{\sum_{l=1}^{i}\exp\left(\frac{\boldsymbol{x}_i^\top U_Q^{(h)}U_K^{(h)\top}\boldsymbol{x}_l}{\sqrt{d_k}}\right)} & \text{para } j \le i, \\ 0 & \text{caso contrário.} \end{cases}$$

Note que $\sum_{j=1}^{i} w_{ij}^{(h)} = 1$, definindo uma distribuição de probabilidade sobre o histórico.

Identificação de médias ponderadas dinamicamente

A saída da cabeça h, denotada por $\boldsymbol{Z}^{(h)}$, é a ponderação dos valores V pelos pesos $\boldsymbol{W}$.

$$\boldsymbol{Z}^{(h)} = \boldsymbol{W}^{(h)}V^{(h)} = \begin{bmatrix} \sum_{j=1}^{1} w_{1j}^{(h)}\boldsymbol{v}_j \\ \vdots \\ \sum_{j=1}^{k} w_{kj}^{(h)}\boldsymbol{v}_j \end{bmatrix} \in \mathbb{R}^{k\times d_v}.$$

Para a i-ésima posição (linha), temos a dedução explícita baseada em Patriota [2025]:

$$\text{Attention}^{(h)}(\boldsymbol{X})_i = \sum_{j=1}^{k} w_{ij}^{(h)}\boldsymbol{v}_j = \sum_{j=1}^{i} w_{ij}^{(h)}(\boldsymbol{X})(\boldsymbol{x}_j^\top U_V^{(h)}).$$

Multi-head Attention

A *Multi-head Attention (MHA)* é obtida concatenando as saídas $\boldsymbol{Z}^{(h)}$ horizontalmente:

$$\boldsymbol{Z}_{\text{concat}} = \text{Concat}(\boldsymbol{Z}^{(1)}, \ldots, \boldsymbol{Z}^{(H)}) \in \mathbb{R}^{k \times (H \cdot d_v)} = \left[\boldsymbol{Z}^{(1)} \middle| \boldsymbol{Z}^{(2)} \middle| \ldots \middle| \boldsymbol{Z}^{(H)}\right] \in \mathbb{R}^{k \times (H \cdot d_v)},$$

e aplica-se uma transformação linear final parametrizada pela matriz $U_O \in \mathbb{R}^{(H \cdot d_v) \times d_{model}}$ para misturar as informações dos subespaços e projetar a representação de volta à dimensão do modelo:

$$\text{MHA}(\boldsymbol{X}) = \boldsymbol{Z}_{\text{concat}} U_O$$

$$\text{MHA}(\boldsymbol{X}) = \text{Concat}\left(\underbrace{\text{softmax}\left(\frac{(\boldsymbol{X}U_Q^{(h)})(\boldsymbol{X}U_K^{(h)})^\top}{\sqrt{d_k}} + \boldsymbol{M}\right)}_{\text{Pesos } \boldsymbol{W}^{(h)}} (\boldsymbol{X}U_V^{(h)})\right)_{h=1}^{H} U_O,$$

Decompondo U_O em blocos $U_O^{(h)}$ correspondentes a cada cabeça, podemos expandir a expressão para o vetor de saída na posição i:

$$\begin{aligned}\text{MHA}(\boldsymbol{X})_i &= \left[\sum_{j=1}^{i} w_{ij}^{(1)} \boldsymbol{x}_j^\top U_V^{(1)}, \ldots, \sum_{j=1}^{i} w_{ij}^{(H)} \boldsymbol{x}_j^\top U_V^{(H)}\right] U_O \\ &= \sum_{h=1}^{H} \left(\sum_{j=1}^{i} \underbrace{w_{ij}^{(h)}(\boldsymbol{X})}_{\text{Peso Dinâmico}} \times \underbrace{(\boldsymbol{x}_j^\top U_V^{(h)})}_{\text{Valor Transformado}}\right) U_O^{(h)}\end{aligned}$$

A equação acima demonstra explicitamente que a nova representação do token i é uma agregação de médias ponderadas dos vetores passados em diferentes subespaços. Diferente de modelos de séries temporais clássicos com coeficientes fixos, aqui os pesos $w_{ij}^{(h)}(\boldsymbol{X})$ são dinâmicos: são funções dos próprios dados observados, variando conforme a similaridade semântica (kernel) entre o token atual e o contexto passado. Isso configura a atenção como um estimador de núcleo (*kernel smoother*), conforme discutido por Tsai et al. [2019].

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

Resolução

A resolução baseou-se nos cálculos feitos no exercício 1), uso de Transformer, com dimensão interna fixada em 128 e taxa de dropout de 0,2 para regularização. A entrada é mapeada para um espaço latente através da soma de embeddings de tokens e posicionais, dada por $\boldsymbol{h}_0 = \boldsymbol{X}W_e + \boldsymbol{P}$. A estrutura principal compõe-se de duas camadas idênticas, cada uma contendo normalização, uma rede feed-forward com ativação GeLU e o mecanismo de Atenção Multi-Cabeça Mascarada (duas cabeças), cuja dinâmica de pesos é dada pela equação $\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + \boldsymbol{M}\right)$, onde a máscara $\boldsymbol{M}$ (com $-\infty$ no triângulo superior). Na etapa de processamento, o corpus de Shakespeare foi tokenizado em nível de caractere e, para otimizar a execução mantendo a generalização, utilizou-se um subconjunto de 30% dos dados segmentados em janelas de contexto de tamanho 64. Minimizou-se a Entropia Cruzada $\mathcal{L}(\theta) = -\sum \log P(x_t|x_{<t})$ sobre lotes estocásticos, utilizando o otimizador AdamW e gradient clipping ao longo de três épocas. Por fim, implementou-se um decodificador auto-regressivo onde, a cada passo t, os logits de saída $\boldsymbol{z}_t$ são escalonados por uma temperatura $\tau = 0.8$, definindo a distribuição condicional $P(x_{next}|x_{<t}) = \text{softmax}(\frac{\boldsymbol{z}_t}{\tau})$. O próximo token foi obtido via amostragem Multinomial direta sobre essa distribuição ajustada, sem a aplicação de filtros de truncamento como Top-k ou Top-p (Nucleus), garantindo que a diversidade sintática fosse controlada exclusivamente pelo parâmetro térmico (vide o texto gerado e a Figura 1).

$\Longrightarrow$ **Texto gerado**

```
> Generate_Text_Base(fitted_model, start_str = "MARCIUS: ", max_new_tokens = 2000)

--- Gerando texto inspirado em Shakespeare (Start: 'MARCIUS: ') ---
MARCIUS: Why, with unswer'd his shall be royal?

CORIOLANUS:
My wife
To sland more the lord.
```

BUCKINGHAM:
Which prevail'd to thee consul.
What I pains, the peace, not liver your to his sanctuary.

QUEEN ELIZABETH:
What shall they were senator: for my son your good mother discaster.

BUCKINGHAM:
And you talking of the consul, you that see what aery,
That not this as for this such and mine ears,
Which like the that do dead.

VOLUMNIA:
On brother, for let us has care in grace of my better the wive would them,
Madam, be never is shall be better in the fair marry; which had thou had be common we'll private and foul
Sicinius spoke of the break the soitors of his bold.

First Senator:
What you have my dead be a cosqued of the king mine that the be prides are in,
When news are so down, and I have look of mine
Your enemy, as the remain my life, no partly, and sweet be mine,
And for all thee time writes that grave enemy?
When I would the people.

GLOUCESTER:
What is say, what in the been thou will request?

CORIOLANUS:
Farewell'd with his good your curse a king!

CORIOLANUS:
You have been doth the bloody have noise,
And I think and many he
creeting find
the treasoness angry to my friends for this the duke out of mine and pity well inform'd,
When then, to be a was it she curse
Which, enter thate a blood lord; and the mighty fight,
That much of what set us taunt the world as against the great of forth
Your death?

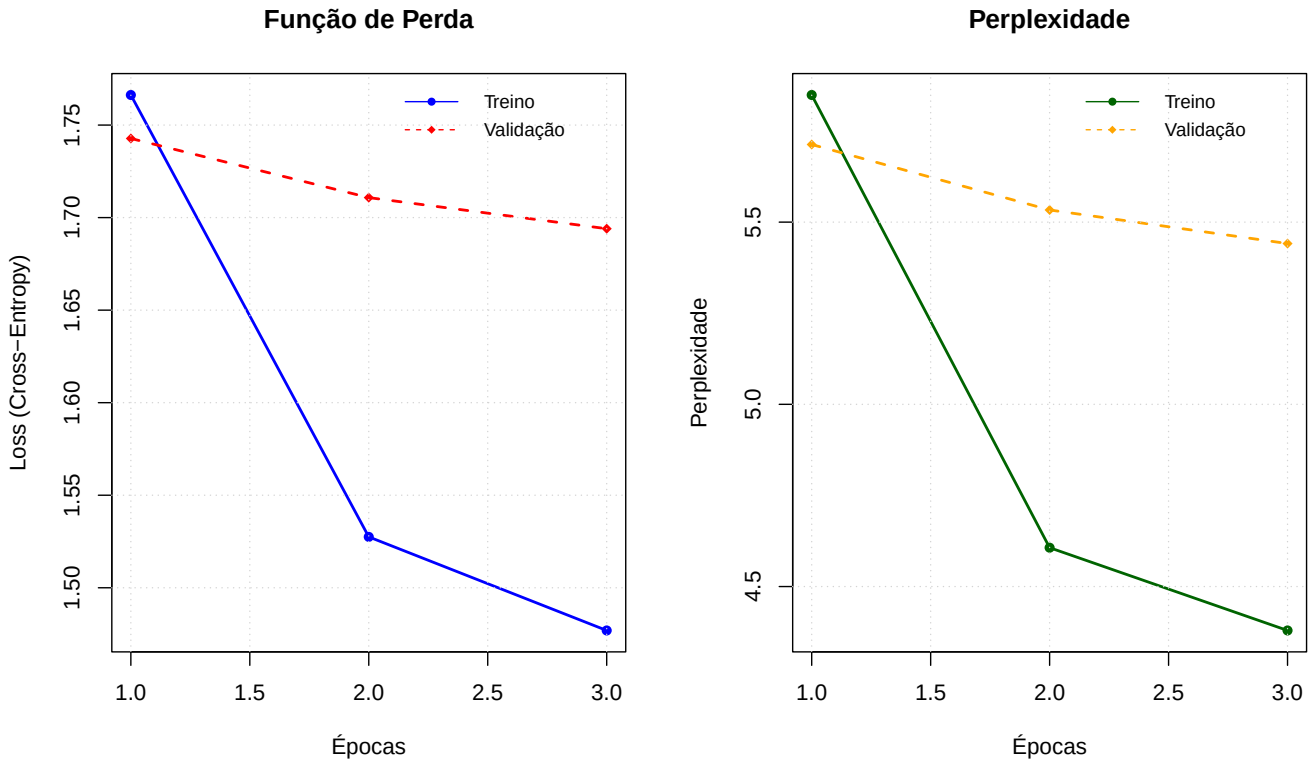
CORIOLANUS:
Why throther's should not the citizens: and debt the mayor my mother,
To for a worthy wife men of a judgment and more to give his people in the has in a man a gainst thy state
And, that is on the did vergise honours; and thee, She had before,
And what I would say your good not us they have your france and your prison wife!

KING RICHARD III:
Pray, Grance is heard you then the Tower;
The comes of poor thou have greating from me,
The gods, and if their of God my friends,
Where them all is would be a shall your mother,
Since we have the more whe

Table 1: Desempenho do Modelo: Loss e Perplexidade por Época

Época	Loss (Cross-Entropy)		Perplexidade (PPL)	
	Treino	Validação	Treino	Validação
1	1.7662	1.7427	5.85	5.71
2	1.5275	1.7108	4.61	5.53
3	1.4769	1.6940	4.38	5.44

Figure 1: Desempenho do GPT implementado



Os dados demonstram que o modelo é capaz de aprender, visto que as métricas de erro diminuem progressivamente em ambas as etapas (Tabela 1 e Figura 1). Contudo, há um claro indicativo de sobreajuste (overfitting) a partir da segunda época, caracterizado pelo descolamento entre as curvas de desempenho: enquanto o modelo evolui rapidamente na memorização dos dados de treino (reduzindo a perda de 1.76 para 1.47), a capacidade de generalização no conjunto de validação melhora de forma muito mais lenta e marginal (de 1.74 para 1.69), sugerindo que a rede está começando a decorar o texto específico em vez de aprender as regras gerais da linguagem.

3. Repita o item dois sem o mecanismo da atenção (com múltiplas cabeças) e descreva o que ocorre no treinamento.

Resolução

Para a resolução, manteve-se a mesma dimensão de embedding em 128 e as configurações de regularização, mas removendo o mecanismo de Atenção Multi-Cabeça. A entrada preservou o mapeamento para o espaço latente via $\boldsymbol{h}_0 = \boldsymbol{X}W_e + \boldsymbol{P}$, contudo, o processamento nas duas camadas ocultas restringiu-se exclusivamente a redes feed-forward (FFN) com ativação GeLU e conexões residuais. Isso transformou a atualização do estado oculto na posição t e camada l em uma operação estritamente pontual dada por $\boldsymbol{h}_t^{(l)} = \boldsymbol{h}_t^{(l-1)} + \text{FFN}(\text{LN}(\boldsymbol{h}_t^{(l-1)}))$, eliminando a matriz de correlação $QK^\top$ e, consequentemente, impedindo o fluxo de informação dos tokens passados $x_{<t}$ para o estado atual. O treinamento replicou as condições anteriores, utilizando o otimizador AdamW para minimizar a Entropia Cruzada $\mathcal{L}(\theta) = -\sum \log P(x_t|x_{<t})$ sobre o mesmo subconjunto de 30% do corpus de Shakespeare. O resultado observado foi um subajuste (underfitting) severo (Figura 2), onde o modelo degenerou para um estimador estatístico condicionado apenas à posição absoluta e não ao contexto. Para a geração de texto, manteve-se o decodificador auto-regressivo com temperatura $\tau = 0.8$ e amostragem Multinomial direta, o que resultou na produção de sequências de caracteres incoerentes, comprovando que, sem a atenção, o modelo é incapaz de modelar dependências de longo prazo ou apreender a estrutura sintática da linguagem.

⟹ Texto gerado

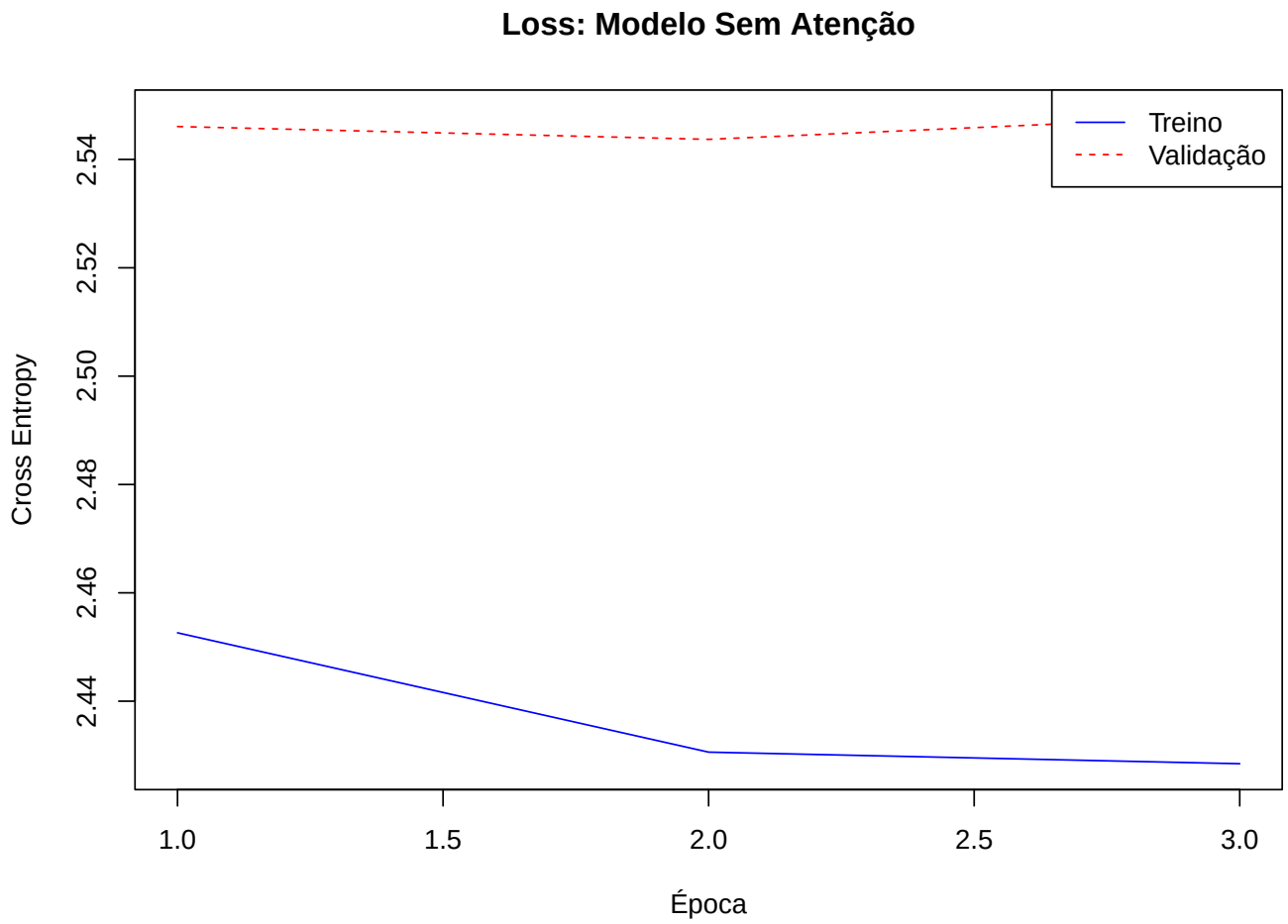
```
--- Gerando texto (SEM ATENÇÃO) - Seed: 'MARCIUS: ' ---
MARCIUS: t womendst t wesunge wo mol at h ly he povery thounghe ind h by?
Yowel l iun than:
TENGI ny heimer s IOUCOLAnde, thy CORDYo lath s thite s watsthin.

Toue s thean chasowisseld atou weanolldie.
We thiragreare tere INothad myonnotithed s ke oungllou t t thanyow hobe

They me tofo win d thand toreromelome, te wnein t the theit
or lly arathad f ns be, anor ive parelly s,

CHas

Lofimy len t hapay y prers har o ss thy ur an our this bl.

QUS:

ARI my

Anthat pous wau u bevestherord toswethy sandodit t a githon ave
pou ou asast ad shewe, y wir I hise ard henterd sord
he belld t hes the hist t Themasthuowethe m:
The s s thounthes tou hesedithe me tl ghot s.
INN atch hey prt the oue me wll-se tthaiathot wigle
he, w me ht therd tond y fer we!
VA:

BENICOr an tale!
Tothinghicofe'th w!
Thourofin f weand t alle kear ETher, nd athorerd d hind ICOnd ourst
thire her s isthesslly ieithellledink te thareatsein mus;

Wheswour o my? hasus atha RIUSThowhe; t er aled mer ous the hat tinthath

Courat II he ws, g he

ICENoug.
Mor urabe wit is gat INENUMatilld cen sint wor, ayougenthendimimat
we here thand an. l
be ane ak y ourowouse be t
The ms!
t it prd,
The whaldo tousthe beces fanthar I'rooou then enckelan t ou t gr
ndereand yory, s wowond t.
Th t 'scaise Mame
SHABure ce y en t avees ply as mavouly pe f ge
BUSott the d,

HESIINI'd Incth hofe he ay he llin thimur f thaven titoren:
Tolselly ghe
To lonounateloullit walasl s banly, s y? the my hithons pend
wat oul sther'sthir, d lo cire y ande me h hely ounth ING if,
METhat ond walinendines s and Thay; igr swelind y! thand
DWhabily on lve the thie bu t metheeag be fee hede tharo l,
BUSen t t alling' ulors fopoube INENELARod shy pe ager note he
mesthatoure mure t y at mincoun th d le cor theandinstord
sthisou t ce they s I at y heshanspous.
Shawouctho t ce plisthie nd hat he beag ofave t, heweind
torit he he IICERI k whayoof be.
Mament s beceat ne wh be Ifomato th'te s.

My lithie,
```

```
AS:
IVEThanghen thale, lind wounouse, m RI hathit oulothi
```

Table 2: Desempenho Comparativo: Modelo Sem Atenção

Época	**Loss (Cross-Entropy)**		**Perplexidade (PPL)**	
	Treino	**Validação**	**Treino**	**Validação**
1	2.4520	2.5450	11.61	12.74
2	2.4310	2.5430	11.37	12.72
3	2.4290	2.5480	11.35	12.78

Figure 2: Desempenho do modelo sem atenção



A remoção do mecanismo de atenção prejudicou a capacidade de aprendizado da rede, evidenciando um cenário claro de subajuste. Ao contrário do modelo completo, que atingiu uma perplexidade de 5.44 (Tabela 1), esta versão sem atenção estagnou em uma perplexidade de 12.78 (Tabela 2), indicando que o modelo é incapaz de capturar as dependências contextuais e relacionamentos de longo prazo entre os caracteres. A curva de validação praticamente plana (Figura 2) sugere que, sem a capacidade de ponderar a relevância entre diferentes tokens da sequência, a rede se limita a aprender apenas estatísticas posicionais básicas e frequências de caracteres.

⟹ Código R Exercício 2

```
> library(torch)
> library(luz)
> library(magrittr)
> GPT <- torch::nn_module(
+   name = "GPT",
+   initialize = function(block_size, n_embd, N_Layers, nvoc, Head, p0 = 0.1) {
+     self$N <- N_Layers
+     self$block_size <- block_size
+     self$head_dim <- n_embd / Head
+     R^{d_{model}}
+     self$wpe <- torch::nn_embedding(block_size, n_embd)
+     self$wte <- torch::nn_embedding(nvoc, n_embd, padding_idx = 1)
+     # MHA(X) = Concat(head_1, ..., head_H) U_O
+     # head_h = Softmax( (X U_Q)(X U_K)^T / sqrt(d_k) + M ) (X U_V)
+     self$MH <- torch::nn_module_list(lapply(
+       1:N_Layers,
+       function(x) torch::nn_multihead_attention(embed_dim = n_embd, num_heads = Head, dropout = p0)
+     ))
+     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) {
+     # Entrada x: (Batch, Seq_Len)
+     B <- x$size(1)
+     k <- x$size(2)
+     x1 <- torch::torch_arange(1, k, dtype = torch::torch_int(), device = x$device)$unsqueeze(1)
+     #\boldsymbol{h}_0 = \text{Embed}(x) + \text{Pos}(x)
+     output <- self$wte(x) + self$wpe(x1)
+     output <- self$drop0(output)
+     # M_{ij} = -\infty \text{ se } j > i \text{ (futuro proibido)}
+     wei <- torch::torch_zeros(k, k, dtype = torch::torch_bool(), device = x$device)
+     wei[upper.tri(wei)] <- 1
+     for (j in 1:self$N) {
+       output_norm <- self$scale1[[j]](output)
+       Q <- torch::torch_transpose(output_norm, 1, 2)
+       mha_out <- self$MH[[j]](query = Q, key = Q, value = Q,
+                               attn_mask = wei,
+                               need_weights = FALSE)[[1]]
+       mha_out <- torch::torch_transpose(mha_out, 1, 2)
+    #\boldsymbol{h}' = \boldsymbol{h} + \text{MHA}(\text{LN}(\boldsymbol{h}))
+       output <- output + mha_out
+   #\boldsymbol{h}_{next} = \boldsymbol{h}' + \text{FFN}(\text{LN}(\boldsymbol{h}'))
+       output <- output + self$FFN[[j]](self$scale2[[j]](output))
+     }
+     #P(X_t|X_{<t}) = \text{softmax}(\text{Logits})
+     output <- self$ln_f(self$scale3(output))
+     return(output)
+   }
+ )
```

```
> config <- list(
+   file_name = "Shakespeare.txt",
+   block_size = 64,
+   n_embd = 128,
+   N_Layers = 2,
+   Head = 2,
+   lr = 0.001,
+   batch_size = 64,
+   epochs = 3,
+   p0 = 0.2
+ )
> Encoder <- function(File_chars, Vocabulary) {
+   FileX <- numeric(length(File_chars))
+   lookup <- setNames(seq_along(Vocabulary), Vocabulary)
+   FileX <- lookup[File_chars]
+   return(unname(FileX))
+ }
> Decoder <- function(File_indices, Vocabulary) {
+   return(Vocabulary[File_indices])
+ }
> if (!file.exists(config$file_name)) {
+   download.file("https://github.com/AGPatriota/GPT4R/blob/main/Shakespeare.txt",
+                 config$file_name)
+ }
> raw_text <- readChar(config$file_name, file.info(config$file_name)$size)
> vocab <- sort(unique(unlist(strsplit(raw_text, ""))))
> vocab <- c("<PAD>", vocab)
> nvoc <- length(vocab)
> full_tokens <- Encoder(unlist(strsplit(raw_text, "")), vocab)
> set.seed(42)
> limit_data <- floor(0.3  length(full_tokens))
> subset_tokens <- full_tokens[1:limit_data]
> ShakespeareDataset <- torch::dataset(
+   name = "ShakespeareDataset",
+   initialize = function(tokens, block_size) {
+     self$tokens <- tokens
+     self$block_size <- block_size
+     self$n_samples <- length(tokens) - block_size
+   },
+   .getitem = function(i) {
+     idx_start <- i
+     idx_end <- i + self$block_size
+     chunk <- self$tokens[idx_start:idx_end]
+     x <- torch::torch_tensor(chunk[1:self$block_size], dtype = torch::torch_int())
+     y <- torch::torch_tensor(chunk[2:(self$block_size + 1)], dtype = torch::torch_long())
+     list(x = x, y = y)
+   },
+   .length = function() { return(self$n_samples) }
+ )
> #Treino/Validação (90% / 10%)
> train_len <- floor(0.9  length(subset_tokens))
> train_tokens <- subset_tokens[1:train_len]
> test_tokens <- subset_tokens[(train_len + 1):length(subset_tokens)]
> train_ds <- ShakespeareDataset(train_tokens, config$block_size)
> test_ds <- ShakespeareDataset(test_tokens, config$block_size)
> train_dl <- torch::dataloader(train_ds, batch_size = config$batch_size,
shuffle = TRUE, num_workers = 0)
> test_dl <- torch::dataloader(test_ds,batch_size = config$batch_size,
num_workers = 0)
> nn_cross_entropy_wrapper <- torch::nn_module(
+   initialize = function() {
+     self$loss <- torch::nn_cross_entropy_loss()
+   },
+   forward = function(input, target) {
```

```
+     self$loss(input$reshape(c(-1, input$size(3))), target$reshape(c(-1)))
+   }
+ )
> model_setup <- GPT %>%
+   luz::setup(
+     loss = nn_cross_entropy_wrapper(),
+     optimizer = torch::optim_adamw
+   ) %>%
+   luz::set_hparams(
+     block_size = config$block_size,
+     n_embd = config$n_embd,
+     N_Layers = config$N_Layers,
+     nvoc = nvoc,
+     Head = config$Head,
+     p0 = config$p0
+   ) %>%
+   luz::set_opt_hparams(lr = config$lr)
> fitted_model <- luz::fit(
+   model_setup,
+   train_dl,
+   epochs = config$epochs,
+   valid_data = test_dl,
+   accelerator = luz::accelerator(),
+   callbacks = list(luz::luz_callback_gradient_clip(max_norm = 1.0)),
+   verbose = TRUE
+ )
> analise_resultados_base_r <- function(fitted_obj) {
+   history <- fitted_obj$records$metrics
+   epochs <- 1:length(history$train)
+   loss_train <- unlist(lapply(history$train, function(x) x$loss))
+   loss_valid <- unlist(lapply(history$valid, function(x) x$loss))
+   # Perplexidade: PP(x) = exp(Entropy(x))
+   ppl_train <- exp(loss_train)
+   ppl_valid <- exp(loss_valid)
+   df_metrics <- data.frame(
+     Epoca = epochs,
+     Loss_Treino = round(loss_train, 4),
+     Loss_Valid = round(loss_valid, 4),
+     PPL_Treino = round(ppl_train, 2),
+     PPL_Valid = round(ppl_valid, 2)
+   )
+   print(df_metrics, row.names = FALSE)
+   par(mfrow = c(1, 2), mar = c(5, 4, 4, 2) + 0.1)
+   y_range_loss <- range(c(loss_train, loss_valid))
+   plot(epochs, loss_train, type = "o", pch = 16, col = "blue", lwd = 2,
+        ylim = y_range_loss, xlab = "Épocas", ylab = "Loss (Cross-Entropy)",
+        main = "Função de Perda")
+   lines(epochs, loss_valid, type = "o", pch = 18, col = "red", lwd = 2, lty = 2)
+   grid()
+   legend("topright", legend = c("Treino", "Validação"),
+          col = c("blue", "red"), pch = c(16, 18), lty = c(1, 2), bty = "n", cex=0.8)
+   y_range_ppl <- range(c(ppl_train, ppl_valid))
+   plot(epochs, ppl_train, type = "o", pch = 16, col = "darkgreen", lwd = 2,
+        ylim = y_range_ppl, xlab = "Épocas", ylab = "Perplexidade",
+        main = "Perplexidade")
+   lines(epochs, ppl_valid, type = "o", pch = 18, col = "orange", lwd = 2, lty = 2)
+   grid()
+   legend("topright", legend = c("Treino", "Validação"),
+          col = c("darkgreen", "orange"), pch = c(16, 18), lty = c(1, 2), bty = "n", cex=0.8)
+   par(mfrow = c(1, 1))
+ }
> analise_resultados_base_r <- function(fitted_obj) {
+   history <- fitted_obj$records$metrics
+   epochs <- 1:length(history$train)
```

```
+   loss_train <- unlist(lapply(history$train, function(x) x$loss))
+   loss_valid <- unlist(lapply(history$valid, function(x) x$loss))
+   #Perplexidade: PP(x) = exp(Entropy(x))
+   ppl_train <- exp(loss_train)
+   ppl_valid <- exp(loss_valid)
+   df_metrics <- data.frame(
+     Epoca = epochs,
+     Loss_Treino = round(loss_train, 4),
+     Loss_Valid = round(loss_valid, 4),
+     PPL_Treino = round(ppl_train, 2),
+     PPL_Valid = round(ppl_valid, 2)
+   )
+   print(df_metrics, row.names = FALSE)
+   par(mfrow = c(1, 2), mar = c(5, 4, 4, 2) + 0.1)
+   y_range_loss <- range(c(loss_train, loss_valid))
+   plot(epochs, loss_train, type = "o", pch = 16, col = "blue", lwd = 2,
+        ylim = y_range_loss, xlab = "Épocas", ylab = "Loss (Cross-Entropy)",
+        main = "Função de Perda")
+   lines(epochs, loss_valid, type = "o", pch = 18, col = "red", lwd = 2, lty = 2)
+   grid()
+   legend("topright", legend = c("Treino", "Validação"),
+          col = c("blue", "red"), pch = c(16, 18), lty = c(1, 2), bty = "n", cex=0.8)
+   y_range_ppl <- range(c(ppl_train, ppl_valid))
+   plot(epochs, ppl_train, type = "o", pch = 16, col = "darkgreen", lwd = 2,
+        ylim = y_range_ppl, xlab = "Épocas", ylab = "Perplexidade",
+        main = "Perplexidade")
+   lines(epochs, ppl_valid, type = "o", pch = 18, col = "orange", lwd = 2, lty = 2)
+   grid()
+   legend("topright", legend = c("Treino", "Validação"),
+          col = c("darkgreen", "orange"), pch = c(16, 18), lty = c(1, 2), bty = "n", cex=0.8)
+   par(mfrow = c(1, 1))
+ }
> analise_resultados_base_r(fitted_model)

> Generate_Text_Base <- function(model_fitted, start_str, max_new_tokens = 300, temperature = 0.8) {
+   model <- model_fitted$model
+   model$eval()
+   device <- if(torch::cuda_is_available()) "cuda" else "cpu"
+   model$to(device = device)
+   start_tokens <- Encoder(unlist(strsplit(start_str, "")), vocab)
+   idx <- torch::torch_tensor(start_tokens, dtype = torch::torch_int(), device = device)$unsqueeze(1)
+   cat(sprintf("\n--- Gerando texto inspirado em Shakespeare (Start: '%s') ---\n", start_str))
+   cat(start_str)
+
+   torch::with_no_grad({
+     for (i in 1:max_new_tokens) {
+       idx_cond <- if (idx$size(2) <= config$block_size) idx else idx[,
+ (idx$size(2) - config$block_size + 1):idx$size(2)]
+       # Predição: P(X_t | X_{<t})
+       logits <- model(idx_cond)
+       logits <- logits[, logits$size(2), ] / temperature
+       probs <- torch::nnf_softmax(logits, dim = -1)
+       idx_next <- torch::torch_multinomial(probs, num_samples = 1)
+       cat(Decoder(as.integer(idx_next$cpu()), vocab))
+       idx <- torch::torch_cat(list(idx, idx_next), 2)
+     }
+   })
+   cat("\n\n")
+ }
> Generate_Text_Base(fitted_model, start_str = "MARCIUS: ", max_new_tokens = 2000)
```

⟹ Código R Exercício 3

```
> #================
> library(torch)
> library(luz)
> library(magrittr)

> GPT_NoAttn <- torch::nn_module(
+   name = "GPT_NoAttn",
+
+   initialize = function(block_size, n_embd, N_Layers, nvoc, Head, p0 = 0.1) {
+     self$N <- N_Layers
+     self$block_size <- block_size
+     self$wpe <- torch::nn_embedding(block_size, n_embd)
+     self$wte <- torch::nn_embedding(nvoc, n_embd, padding_idx = 1)
+     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$drop0 <- torch::nn_dropout(p = p0)
+   },
+   forward = function(x) {
+     B <- x$size(1)
+     k <- x$size(2)
+
+     x1 <- torch::torch_arange(1, k, dtype = torch::torch_int(), device = x$device)$unsqueeze(1)
+     output <- self$wte(x) + self$wpe(x1)
+     output <- self$drop0(output)
+     for (j in 1:self$N) {
+       output <- output + self$FFN[[j]](self$scale2[[j]](output))
+     }
+
+     output <- self$ln_f(self$scale3(output))
+     return(output)
+   }
+ )
> #
> config <- list(
+   file_name = "Shakespeare.txt",
+   block_size = 64,
+   n_embd = 128,
+   N_Layers = 2,
+   Head = 2,
+   lr = 0.001,
+   batch_size = 64,
+   epochs = 3,
+   p0 = 0.2
+ )
> #
> Encoder <- function(File_chars, Vocabulary) {
+   lookup <- setNames(seq_along(Vocabulary), Vocabulary)
+   unname(lookup[File_chars])
+ }
> Decoder <- function(File_indices, Vocabulary) Vocabulary[File_indices]
```

```
> if (!file.exists(config$file_name)) {
+   download.file("https://github.com/AGPatriota/GPT4R/blob/main/Shakespeare.txt", config$file_name)
+ }
> raw_text <- readChar(config$file_name, file.info(config$file_name)$size)
> vocab <- sort(unique(unlist(strsplit(raw_text, ""))))
> vocab <- c("<PAD>", vocab)
> nvoc <- length(vocab)
> full_tokens <- Encoder(unlist(strsplit(raw_text, "")), vocab)
> set.seed(42)
> limit_data <- floor(0.3  length(full_tokens))
> subset_tokens <- full_tokens[1:limit_data]
> train_len <- floor(0.9  length(subset_tokens))
> train_ds <- ShakespeareDataset(subset_tokens[1:train_len], config$block_size)
> test_ds <- ShakespeareDataset(subset_tokens[(train_len + 1):length(subset_tokens)], config$block_size)
> train_dl <- torch::dataloader(train_ds,
batch_size = config$batch_size, shuffle = TRUE, num_workers = 0)
> test_dl <- torch::dataloader(test_ds,
batch_size = config$batch_size, num_workers = 0)
> #
> nn_cross_entropy_wrapper <- torch::nn_module(
+   initialize = function() self$loss <- torch::nn_cross_entropy_loss(),
+   forward = function(input,
target) self$loss(input$reshape(c(-1, input$size(3))), target$reshape(c(-1)))
+ )
> model_setup_no_attn <- GPT_NoAttn %>%
+   luz::setup(
+     loss = nn_cross_entropy_wrapper(),
+     optimizer = torch::optim_adamw
+   ) %>%
+   luz::set_hparams(
+     block_size = config$block_size, n_embd = config$n_embd,
+     N_Layers = config$N_Layers, nvoc = nvoc, Head = config$Head, p0 = config$p0
+   ) %>%
+   luz::set_opt_hparams(lr = config$lr)
> cat("Iniciando treinamento SEM ATENÇÃO...\n")
Iniciando treinamento SEM ATENÇÃO...
> fitted_no_attn <- luz::fit(
+   model_setup_no_attn,
+   train_dl,
+   epochs = config$epochs,
+   valid_data = test_dl,
+   accelerator = luz::accelerator(),
+   callbacks = list(luz::luz_callback_gradient_clip(max_norm = 1.0)),
+   verbose = TRUE
+ )

> analise_comparativa <- function(fitted_obj) {
+   history <- fitted_obj$records$metrics
+   loss_train <- unlist(lapply(history$train, function(x) x$loss))
+   loss_valid <- unlist(lapply(history$valid, function(x) x$loss))
+   ppl_valid <- exp(loss_valid)
+   cat(sprintf("Loss Final (Validação): %.4f\n", tail(loss_valid, 1)))
+   cat(sprintf("Perplexidade Final: %.2f\n", tail(ppl_valid, 1)))
+
+   # Plot simples
+   plot(loss_train, type="l", col="blue", ylim=range(c(loss_train, loss_valid)),
+        main="Loss: Modelo Sem Atenção", ylab="Cross Entropy", xlab="Época")
+   lines(loss_valid, col="red", lty=2)
+   legend("topright", c("Treino", "Validação"), col=c("blue", "red"), lty=1:2)
+ }
> analise_comparativa(fitted_no_attn)

> Generate_Text_Base <- function(model_fitted, start_str, max_new_tokens = 1000, temperature = 0.8) {
+   model <- model_fitted$model
```

```
+   model$eval()
+   device <- if(torch::cuda_is_available()) "cuda" else "cpu"
+   model$to(device = device)
+
+   start_tokens <- Encoder(unlist(strsplit(start_str, "")), vocab)
+   idx <- torch::torch_tensor(start_tokens, dtype = torch::torch_int(), device = device)$unsqueeze(1)
+   cat(sprintf("\n--- Gerando texto (SEM ATENÇÃO) - Seed: '%s' ---\n", start_str))
+   cat(start_str)
+   torch::with_no_grad({
+     for (i in 1:max_new_tokens) {
+       idx_cond <- if (idx$size(2) <= config$block_size) idx else idx[,
(idx$size(2) - config$block_size + 1):idx$size(2)]
+       logits <- model(idx_cond)
+       logits <- logits[, logits$size(2), ] / temperature
+       probs <- torch::nnf_softmax(logits, dim = -1)
+       idx_next <- torch::torch_multinomial(probs, num_samples = 1)
+       cat(Decoder(as.integer(idx_next$cpu()), vocab))
+       idx <- torch::torch_cat(list(idx, idx_next), 2)
+     }
+   })
+   cat("\n\n")
+ }
> Generate_Text_Base(fitted_no_attn, start_str = "MARCIUS: ")
```

References

Alisson G. Patriota. *Fundamentos da Estatística Clássica e do Aprendizado de Máquinas: notas de aula*. IME-USP, São Paulo, 2025.

Alisson Gomes Patriota. The statistical model behind large language models based on the transformer architecture. Technical report, Departamento de Estatística, IME, Universidade de São Paulo, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: a unified understanding of transformer's attention via the lens of kernel. *arXiv preprint arXiv:1908.11775*, 2019.