UB Social


# CS50AI with Python

## 12.Attention

Mateus Schwede

# Problemática

- Escreva IA para prever palavra mascarada em sequência de texto;;
- Este projeto explora Modelos de Linguagem Mascarados (Masked Language Models), como BERT, modelo de linguagem baseado em transformers desenvolvido pelo Google;
- Utiliza biblioteca transformers da Hugging Face;
- Criar programa que:
    - Prediz palavras mascaradas em sentenças usando BERT;
    - Gerar diagramas de pontuações de atenção (attention scores), com um para cada um dos 144 cabeçalhos de atenção (12 camadas x 12 cabeçalhos).
- Analisar diagramas de atenção gerados;
- Objetivo: Compreender que cabeçalhos de atenção do BERT estão "observando" quando processam linguagem natural - ou seja, quais palavras ou estruturas são mais relevantes para previsão.

# Instruções

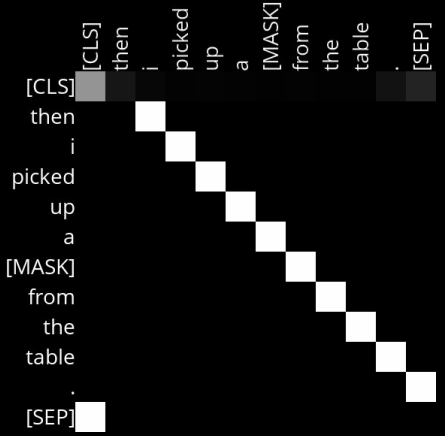
- Baixe código de https://cdn.cs50.net/ai/2023/x/projects/6/attention.zip e descompacte-o;
- No diretório attention, execute "pip3 install -r requirements.txt" para instalar dependências do projeto.

# Funcionamento

- mask.py:
  - Usa BERT para prever palavra mascarada ([MASK]) em frase fornecida pelo usuário e gera diagramas de atenção;
  - Usuário fornece frase contendo token [MASK];
  - AutoTokenizer, da biblioteca Hugging Face, transforma texto em token;
  - Tokens especiais:
    - [MASK]: posição a ser prevista;
    - [CLS]: início da sequência;
    - [SEP]: separa sentenças;
    - Palavras podem ser divididas (ex: “intelligently” → intelligent + ##ly).

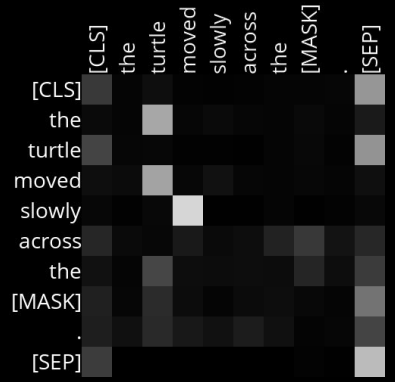
# Funcionamento

- Modelo TFBertForMaskedLM é utilizado para prever palavras na posição [MASK];
- K palavras mais prováveis são mostradas, substituindo [MASK] na frase original;
- Após previsão, função visualize_attentions é chamada para gerar diagramas dos scores de atenção de todos 144 cabeçalhos de atenção do BERT;
- Funções para implementar:
    - et_mask_token_index: localiza posição do [MASK];
    - get_color_for_attention_score: define cores com base nos valores de atenção;
    - visualize_attentions: gera diagramas.



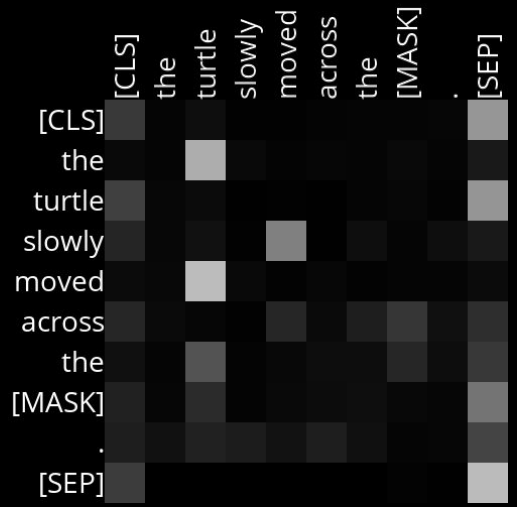
# Funcionamento

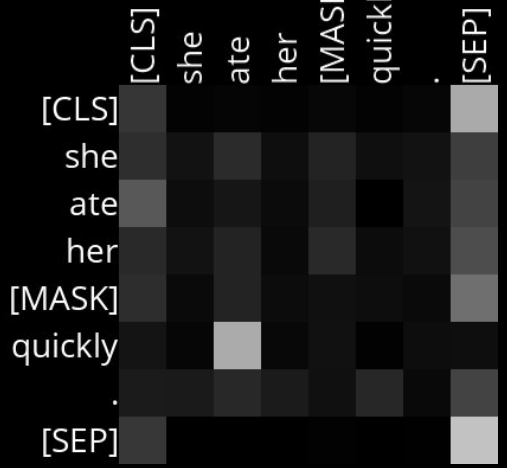
- Cores claras indicam maior peso de atenção; cores escuras, menor
- Layer 3, Head 10, observa-se padrão claro: cada palavra presta atenção à palavra seguinte. Exemplo: "then" presta atenção fortemente a "i";
- Isso faz sentido, pois compreender palavra depende do que vem depois - útil para BERT;
- Nem todos heads são claros; alguns são mais ruidosos e difíceis de interpretar;
- Pode-se investigar padrões específicos, como papel de advérbios:
    - Exemplo: "The turtle moved slowly across the [MASK].";
    - No resultado, Layer 4, Head 11 pode destacar que "slowly" modifica "moved", indicando que BERT percebe essa relação.



# Funcionamento

- Esse head de atenção é mais ruidoso e seu padrão não é imediatamente claro;
- No entanto, nota-se que advérbio "slowly" presta mais atenção ao verbo que ele modifica: "moved";
- Esse comportamento mantém-se mesmo quando ordem das palavras é invertida;
- E isso parece ser verdade até mesmo para frase em que advérbio e verbo que ele modifica não estão diretamente próximos um do outro.



# Funcionamento

- Head de atenção analisado parece identificar relação entre advérbios e verbos que modificam;
- Nem todos heads seguem padrões claros ou correspondem a relações linguísticas humanas previsíveis;
- Mas ainda é possível fazer hipóteses sobre o que cada head está "prestando atenção";
- Projeto inclui essa análise interpretativa para tentar entender comportamento dos heads de atenção do BERT.

# Especificações

- Arquivo mask.py:
  - Função get_mask_token_index:
    - Recebe ID do token [MASK] e objeto BatchEncoding;
    - Retorna índice (base 0) onde token [MASK] aparece na sequência de entrada;
    - Retorna None se token não estiver presente;
    - Assume-se que há no máximo 1 token [MASK] na entrada.

# Especificações

- Função get_color_for_attention_score:
  - Recebe attention score entre 0 e 1;
  - Retorna valor RGB (tupla de 3 inteiros) como cor em escala de cinza;
  - Ex: score 0 → (0, 0, 0) (preto), score 1 → (255, 255, 255) (branco);
  - Para valores intermediários, calcula proporcionalmente (por exemplo, 0.25 ≈ (64, 64, 64)).

# Especificações

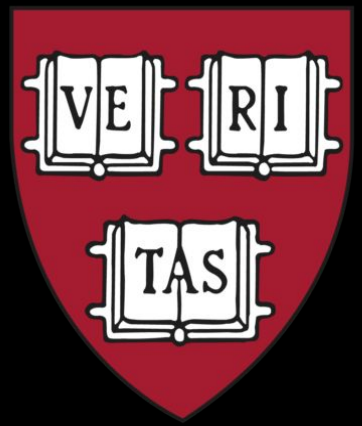
- Função visualize_attentions:
  - Recebe sequência de tokens (lista de strings) e attention scores do modelo;
  - Deve gerar 1 visualização para cada head em cada layer;
  - Usa generate_diagram(layer_number, head_number, ...) com índices baseados em 1;
  - Estrutura de acesso a attentions é: attentions[layer][beam][head].

# Especificações

- Arquivo analysis.md:
  - Após gerar diagramas de atenção, objetivo é analisar e descrever comportamento de pelo menos 2 heads de atenção;
  - Para cada head analisado:
    - Descrever brevemente o que ele parece fazer (qual relação entre palavras ele aprende);
    - Fornecer 2 frases de exemplo usadas para chegar à conclusão.
  - Não repetir exemplos dados (Layer 3, Head 10 e Layer 4, Head 11);
  - É aceitável que alguns heads sejam "barulhentos" ou não tenham interpretação humana clara;
  - Pode-se explorar relações como: Verbos e objetos diretos. Preposições. Pronomes. Adjetivos. Determinantes. Tokens que prestam atenção ao token anterior (em vez do próximo).

# Submissão

- Visual Studio Code online: https://cs50.dev
- Testar precisão da lógica do algoritmo: check50 ai50/projects/2024/x/attention
- Testar estilização do código: style50 mask.py
- Para submissão:
  - Em https://submit.cs50.io/invites/d03c31aef1984c29b5e7b268c3a87b7b, entre com GitHub e autorize CS50;
  - Instale pacote Git, Python 3 (e pip), instalando pacotes: pip3 install style50 check50 submit50
  - Submeta o projeto: submit50 ai50/projects/2024/x/attention
- Verificar avaliação: https://cs50.me/cs50ai.




UB Social

Mateus Schwede
HBS ID 202400167108 - DCE ID @00963203