UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
CURSO: CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL
DISCIPLINA: PROCESSAMENTO DE LINGUAGEM NATURAL
PROFESSOR: YURI DE ALMEIDA MALHEIROS BARBOSA

RELATÓRIO DO PROJETO FINAL DA DISCIPLINA

EQUIPE:

BERGSON GABRIEL DA SILVA OLIVEIRA MELO
IVO CRESCENCIO DE ARAUJO
JOÃO VICTOR BEZERRA DE OLIVEIRA

JOÃO PESSOA, 24/04/2025

1. Apresentação do Problema

Com o crescimento exponencial da quantidade de informações disponíveis na internet, tornou-se cada vez mais difícil para os usuários consumirem grandes volumes de texto de maneira rápida e eficiente. Textos longos, como artigos de notícias, documentos acadêmicos ou relatórios técnicos, exigem tempo e esforço para leitura, dificultando o acesso rápido às informações mais relevantes.

Nesse contexto, técnicas de resumo automático vêm sendo amplamente pesquisadas, com o objetivo de extrair ou gerar versões mais curtas de um texto sem comprometer seu conteúdo essencial. Em especial, os resumos abstrativos, que tentam gerar novos enunciados com base na compreensão do texto original (em vez de apenas extrair frases literais), representam um grande desafio em Processamento de Linguagem Natural (PLN), pois exigem capacidades de compreensão, síntese e geração de linguagem.

Com o avanço dos modelos baseados em transformers, tornou-se possível treinar arquiteturas que lidam com tarefas complexas de PLN, como tradução automática, geração de texto e resumo abstrativo. Este projeto busca explorar essas capacidades com foco na geração de resumos abstrativos.

2. Objetivos

O objetivo principal deste projeto é desenvolver um modelo baseado em transformers para a geração de resumos abstrativos automáticos, a partir de textos de entrada em linguagem natural.

Os objetivos específicos incluem:

- Preparar os dados de entrada com as técnicas aprendidas em sala de aula, realizando a tokenização, vetorização e formatação dos pares texto-resumo.
- Projetar e treinar uma rede neural com arquitetura encoder-decoder utilizando camadas de TransformerEncoder e TransformerDecoder.
- Implementar mecanismos de controle de treinamento como ModelCheckpoint e EarlyStopping para garantir melhor desempenho.
- Avaliar a capacidade do modelo de gerar resumos coerentes, relevantes e concisos.
- Identificar limitações envolvidas no processo de criação e treinamento do modelo, com soluções futuras para melhorar o desempenho e precisão do modelo de geração de resumos abstrativos.

3. Dados Utilizados

Neste projeto, utilizou-se o dataset CNN/DailyMail, versão "3.0.0", que pode ser carregado diretamente através da biblioteca datasets da Hugging Face. Este conjunto de dados foi originalmente construído a partir de artigos de notícias dos sites CNN e Daily Mail, e é amplamente utilizado em tarefas de resumo abstrativo

supervisionado. O dataset é dividido em dados de treinamento, validação e teste, que somados totalizam 311.971 amostras. Ele possui três colunas, ‘article’ com a notícia completa, ‘highlights’ com o resumo abstrativo da notícia, e ‘id’ com o id da notícia.

Os resumos foram escritos por escritores humanos, e representam um bom padrão de resumo abstrativo. Este dataset é amplamente adotado na literatura científica e por desenvolvedores de modelos de PLN por fornecer um grande volume de textos reais com resumos bem estruturados. Sua complexidade torna-o adequado para treinar modelos robustos e avaliar a capacidade de geração de linguagem natural.

3.1 Pré-processamento dos Dados

O pré-processamento dos dados é uma etapa fundamental no desenvolvimento de modelos de Processamento de Linguagem Natural (PLN), especialmente em tarefas de geração de texto como o resumo abstrativo. Essa etapa garante que os dados estejam organizados e padronizados para que o modelo possa compreendê-los e aprender de forma eficiente. No caso deste projeto, o pré-processamento envolveu a redução na quantidade dos dados, a limpeza dos textos, a tokenização (divisão dos textos em unidades linguísticas menores), a vetorização (conversão dos tokens em números inteiros), o truncamento ou preenchimento das sequências para garantir o mesmo tamanho, além da construção de dicionários que mapeiam palavras para índices e vice-versa. Todas essas etapas contribuem para a redução de ruídos nos dados, melhoram a generalização do modelo e são essenciais para garantir um treinamento mais estável e eficaz. Cada uma dessas etapas será descrita a seguir.

Inicialmente, foi necessário reduzir a quantidade de amostras do conjunto de dados, uma vez que o treinamento utilizando a maior parte das 311.971 amostras — sendo 80% destinadas ao treinamento — mostrou-se inviável, tanto em termos de tempo quanto de recursos computacionais. Considerando essa limitação, optou-se por restringir o dataset a 40.000 amostras, que foram posteriormente divididas entre os conjuntos de treino e validação. A seleção dos 40 mil textos foi realizada de forma aleatória, com a utilização de uma semente (seed) fixa para garantir a reprodutibilidade dos resultados em todas as execuções do código. As colunas utilizadas foram a ‘article’, que possui os textos completos, e a coluna ‘highlights’, que possui os resumos. O nome da coluna ‘highlights’ foi alterado para ‘summary’.

Em seguida, foi realizada a etapa de tokenização, que consiste em dividir os textos em unidades linguísticas menores, como palavras. Optou-se por não remover as stopwords das amostras, considerando que esses elementos são essenciais para manter a coesão e a fluidez nos resumos gerados. Duas novas colunas foram criadas no dataset, uma com o texto tokenizado e outra com o resumo tokenizado. A tokenização foi feita utilizando a função word_tokenize da biblioteca NLTK. Além disso, foram inseridos os tokens especiais <start> e <end> no início e no final de

todos os textos e resumos, respectivamente, com o objetivo de sinalizar ao modelo os limites das sequências durante o treinamento. Inicialmente, o processo gerou um total de 31.579.842 tokens, número que foi reduzido para 155.884 após a aplicação de um filtro que manteve apenas os tokens com ocorrência mínima de duas vezes. Outro token especial, <unk>, também foi adicionado ao vocabulário para representar quaisquer tokens desconhecidos durante a inferência. Dessa forma, o vocabulário final passou a ser composto pelos tokens mais frequentes nos textos, juntamente com os tokens especiais <start>, <end> e <unk>.

O passo seguinte consistiu na vetorização, ou seja, na conversão dos tokens (palavras) em números inteiros. Essa etapa é essencial, pois redes neurais não trabalham diretamente com texto, mas sim com representações numéricas. Para isso, foram criados dois dicionários: um que associa cada token a um ID numérico, e outro que realiza o mapeamento inverso. Com base nesses dicionários, foram adicionadas duas novas colunas ao dataset: uma contendo os textos vetorizados e outra com os resumos vetorizados. Também foi necessário limitar o tamanho das sequências — 500 IDs para os textos e 60 para os resumos — sempre garantindo que o token <end> esteja presente ao final de cada sequência.

Na preparação final dos dados para a arquitetura Transformer Encoder-Decoder, foram criadas duas novas colunas no dataframe: uma para a entrada do decoder e outra para a saída esperada do decoder, os resumos-alvo. A entrada do decoder contém os resumos vetorizados com o token <start> no início, mas sem o token <end> no final, pois o modelo precisa aprender a prever o próximo token a partir de uma sequência inicial. Já a saída esperada do decoder contém os mesmos resumos com o token <end> no final e sem o token <start> no início. Além disso, apesar das sequências originais de texto e resumo terem sido previamente ajustadas para um comprimento fixo, a criação das entradas e saídas do decoder (que removem os tokens <start> e <end>, respectivamente) faz com que essas novas sequências tenham tamanhos variados. Por isso, ocorreu a aplicação de zero padding, que garante que as sequências possuam o mesmo tamanho.

Por fim, os conjuntos foram divididos em treino e validação utilizando a função train_test_split. Foram separados os dados de entrada do encoder (os textos vetorizados), bem como as entradas e saídas do decoder (os resumos vetorizados, com e sem os tokens especiais). Essa separação é essencial para garantir que o modelo seja treinado em uma parte dos dados (80%) e avaliado em outra (20%), possibilitando a verificação de seu desempenho em exemplos que ele nunca viu. O uso do parâmetro random_state assegura que a divisão seja sempre a mesma a cada execução do código, permitindo reprodutibilidade nos resultados.

4. Metodologia

4.1 Técnica Utilizada

A arquitetura implementada segue o modelo Transformer do tipo Encoder-Decoder, amplamente utilizado em tarefas de geração de texto, como a sumarização

automática. O modelo começa com a definição dos seguintes hiperparâmetros: a dimensão dos vetores de embedding foi fixada em 128, permitindo representar cada token como um vetor denso; o número de cabeças de atenção foi definido como 4, permitindo que o modelo aprenda diferentes relações contextuais em paralelo; a dimensão da camada feedforward foi estabelecida em 512; e o número de camadas para o encoder foi fixado em 2, assim como para o decoder.

As entradas do modelo consistem em duas sequências de inteiros: encoder_inputs, que representam os textos originais, e decoder_inputs, que representam os resumos-alvo. Ambas as sequências passam por uma camada de Embedding compartilhada, responsável por converter os IDs dos tokens em vetores densos. O parâmetro mask_zero=True garante que o modelo desconsidere os tokens de padding durante o treinamento.

A parte do encoder da arquitetura é composta por duas camadas TransformerEncoder. Cada camada aplica mecanismos de atenção multi-head, seguidos por uma subcamada feedforward. Essas camadas têm a função de processar os textos de entrada, extraindo representações contextuais profundas que capturam as relações entre os tokens da sequência. O resultado desse processamento são vetores numéricos que representam cada token com base em seu contexto no texto completo — essas são as chamadas representações geradas pelo encoder. Elas são essenciais para que o decoder consiga compreender o conteúdo do texto original ao gerar os resumos. Foi utilizado um dropout de 0.1 para evitar overfitting e a função de ativação ‘relu’.

Na sequência, o decoder também é composto por duas camadas TransformerDecoder, que recebem como entrada tanto a sequência de resumos (decoder_inputs) quanto as representações geradas pelo encoder. Durante o treinamento, utiliza-se teacher forcing, na qual o modelo recebe como entrada o resumo real com o token <start> no início e sem o token <end>. Com isso, o modelo aprende a prever o próximo token da sequência com base nos anteriores, guiado pelo resumo verdadeiro. Essa abordagem permite que o decoder gere os resumos passo a passo, considerando tanto os tokens previamente gerados quanto o conteúdo codificado do texto original.

Por fim, a saída do decoder é conectada a uma camada Densa com ativação softmax, que retorna uma distribuição de probabilidade sobre todo o vocabulário. A partir dessa distribuição, o modelo seleciona o token mais provável a ser gerado, compondo assim a sequência final do resumo.

Para treinar a arquitetura, foram utilizadas duas estratégias de controle de desempenho: o ModelCheckpoint e o EarlyStopping. O ModelCheckpoint foi configurado para salvar automaticamente o melhor modelo com base na acurácia de validação, garantindo que a versão mais eficiente do modelo fosse preservada durante o processo de treinamento. Já o EarlyStopping foi empregado para interromper o treinamento caso a acurácia de validação não apresentasse melhorias

após cinco épocas consecutivas, prevenindo o overfitting e economizando recursos computacionais.

O modelo final foi então construído a partir da junção dos inputs do encoder e do decoder, tendo como saída uma camada densa com ativação softmax. A compilação foi feita utilizando o otimizador Adam, com taxa de aprendizado de 0.001, a função de perda sparse categorical crossentropy, adequada para problemas de classificação com rótulos inteiros, e a métrica de acurácia.

O treinamento foi realizado com lotes de 64 amostras ao longo de 141 épocas (treinamento interrompido pelo EarlyStopping), utilizando como entradas o texto completo (para o encoder) e o resumo-alvo com o token especial <start> (para o decoder). A saída esperada era o restante do resumo, deslocado uma posição à esquerda, encerrando com o token <end>. Essa estratégia permite que o modelo aprenda, durante o treinamento, a prever o próximo token do resumo com base no conteúdo original e nos tokens já gerados. A validação foi conduzida de forma semelhante, com um conjunto separado de dados, permitindo monitorar o desempenho em dados não vistos a cada época.

Durante o treinamento, foi adotada a técnica de teacher forcing, amplamente utilizada em modelos de geração de sequência. Essa abordagem consiste em alimentar o decoder com os tokens corretos do resumo-alvo, em vez de utilizar as previsões anteriores do próprio modelo. Na prática, isso foi implementado ao fornecer, como entrada do decoder (y_decoder_input), o resumo deslocado iniciado com o token especial <start>, enquanto a saída esperada (y_decoder_output) é utilizada para avaliar a acurácia do modelo. Essa estratégia permite acelerar o processo de aprendizagem e reduzir a propagação de erros durante o treinamento, contribuindo para uma melhor convergência do modelo.

4.2 Experimento para Avaliar a Técnica Utilizada

Para uma avaliação geral da capacidade de aprendizagem do modelo, os gráficos de acurácia e de perda durante o treinamento foram gerados e podem ser vistos ao final desta seção.

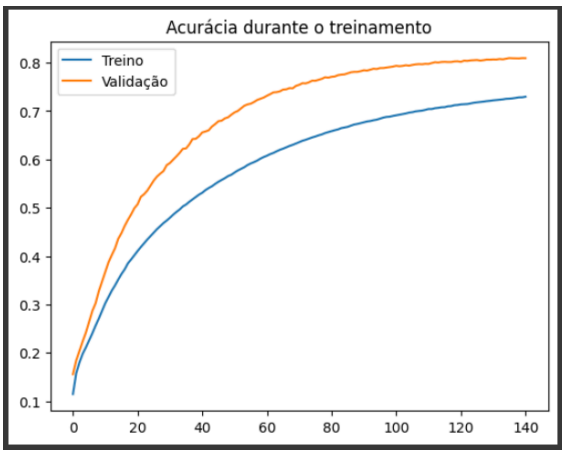
Para a geração de resumos abstrativos durante a inferência, foram implementadas três funções, com abordagens diferentes. A primeira função foi a decode_transformer, determinística. O processo tem início com o token especial <start>, que sinaliza o início da sequência de saída. Em cada iteração, a sequência parcial gerada até o momento é passada como entrada para o decoder, juntamente com a representação codificada do texto original. O modelo, então, prediz a distribuição de probabilidade do próximo token, a partir da qual é selecionado o token mais provável utilizando a operação argmax.

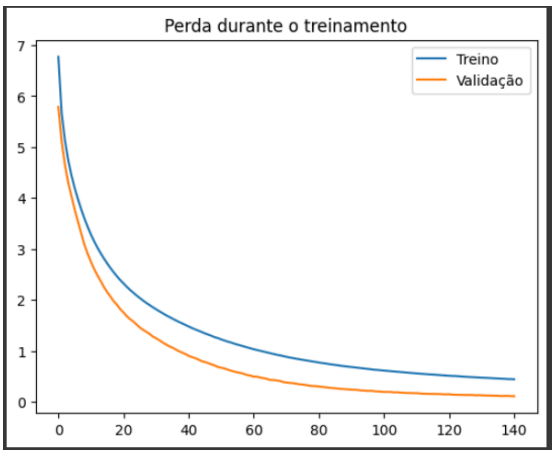
A função decode_transformer_sampling implementa uma abordagem estocástica para a geração de resumos, introduzindo variabilidade nas sequências geradas. A cada etapa de decodificação, o modelo prediz a distribuição de

probabilidade para o próximo token, com base na sequência parcial gerada até o momento e na entrada codificada. Em vez de selecionar diretamente o token mais provável (como no método determinístico), a função seleciona aleatoriamente um dos *top-k* tokens com maiores probabilidades, ponderando essa escolha conforme a distribuição softmax normalizada. Essa estratégia, conhecida como *top-k sampling*, favorece a diversidade nas saídas, gerando resumos mais variados e criativos.

A função decode_transformer_beam_search utiliza a técnica de *beam search* para buscar sequências de saída mais prováveis durante a inferência. Em vez de manter apenas a sequência mais promissora em cada passo, a função mantém múltiplas hipóteses simultaneamente, com base no parâmetro beam_width. Cada sequência parcial recebe uma pontuação acumulada, baseada na soma dos logaritmos das probabilidades dos tokens gerados. Em cada etapa, são consideradas as melhores extensões de todas as hipóteses anteriores, permitindo que o algoritmo explore diversas combinações de tokens. Ao final, a sequência com a maior pontuação é selecionada como resumo final.

No processo de inferência, o teacher forcing não é utilizado em momento algum, diferente do processo de treinamento. Abaixo estão os gráficos de acurácia e de perda do modelo, no treinamento e na validação.





5. Resultados

Para avaliar a precisão do modelo, utilizamos uma amostra que foi vista durante o treinamento, e uma amostra que não foi vista. Abaixo está o resultado da amostra que foi vista, com o resumo original (presente no dataset), e os resumos gerados pela função decode_transformer (determinística), decode_transformer_sampling e decode_transformer_beam_search, respectivamente.

Resumo Original:
<start> troops had n't even finished painting their emblem when they got orders to move . redeployment put strain on battalion , trained to clear mines and bombs from roads . cnn traveled with troops as they flew from one war zone to the next . soldiers say they are ready for the new mission in the new battlefield

Resumo Gerado Determinístico:
troops had n't even finished painting their emblem when they got orders to move . redeployment put strain on battalion , trained to clear mines and bombs from roads . cnn traveled with troops as they flew from one war zone to the next . soldiers say they are ready

Resumo Gerado com Sampling:
last battle valery breyer , for comrade spent the permission since they got idea with planes to climb . they have visited new mh17 tragedy . he leaves them around . vice president picks six astronauts who travels to be left a group to hold their equipment .

Resumo Gerado com Beam Search:
troops had n't even finished painting their emblem when they got orders to move . redeployment put strain on battalion , trained to clear mines and bombs from roads . cnn traveled with troops as they flew from one war zone to the next . soldiers say they are ready

O resultado com dados vistos no treinamento foi muito bom, a função determinística e a função beam search reproduziram exatamente o mesmo resumo original presente no dataset, entretanto, como os resumos gerados foram limitados a 50 tokens, acabaram sendo interrompidos antes do fim. Contudo, com dados não vistos durante o treinamento, o resultado já não foi tão bom, como pode-se verificar abaixo:

Resumo Original:
President Bush to address the Veterans of Foreign Wars on Wednesday .
Bush to say that withdrawing from Vietnam emboldened today's terrorists .
Speech will be latest White House attempt to try to reframe the debate over Iraq .

Resumo Gerado Determinístico:
the obama is not only major cause in iraq and iraq , source of the political cause major cause in iraq , says . obama said on major depression act is not only major source of these issues asia and iraq has chemical weapons .

Resumo Gerado com Sampling:
the obama reacts its `` spy who can only human regulation '' to put pressure over the political observers ' success in the speech convention against isis has nasa in isis , the house speaker and iraq . afghanistan and the political observers have previously only theory ?

Resumo Gerado com Beam Search:
the obama is not only major cause in iraq and iraq , source of the political economies . political concern the political observers believe in iraq has major cause in iraq and afghanistan 's nuclear weapons . the political cause marine corps is not only major cause in iraq and

As principais limitações do modelo estão associadas à restrição de recursos computacionais disponíveis para o seu treinamento. Esse processo demanda um tempo significativo e depende de recursos limitados, como as GPUs fornecidas pelo Google Colab. Além disso, o conjunto de dados utilizado foi reduzido, assim como o comprimento dos textos, com o objetivo de viabilizar a execução do treinamento

dentro das limitações impostas pelo ambiente computacional. Uma arquitetura mais complexa e mais amostras de treinamento devem aumentar a precisão dos resumos gerados, à medida que o modelo aprende a extrair melhor o contexto dos textos originais.