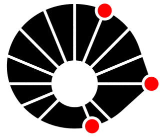

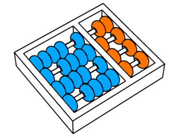

Um arcabouço para visualização e ensino de modelos de aprendizado profundo

Matheus Esteves Zanoto, Marcelo da Silva Reis

Relatório Técnico - IC-PFG-24-44

Projeto Final de Graduação

2024 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

The contents of this report are the sole responsibility of the authors.
O conteúdo deste relatório é de única responsabilidade dos autores.

Um arcabouço para visualização e ensino de modelos de aprendizado profundo

Matheus Esteves Zanoto[1] and Marcelo da Silva Reis[1]

[1]Laboratório Recod.ai; Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)

Resumo

O processo de ensino e aprendizagem no meio acadêmico para áreas em Machine Learning sempre foi um assunto extremamente complexo, principalmente devido a natureza envolvida na criação dos modelos, processo de treinamento e escalabilidade dos dados envolvidos que alimentam esse processo. A complexidade tende a aumentar quando entramos no ramo de Aprendizagem Profunda, em modelos como Transformadores, MLP, CNN, RNN e Encoders, Decoders. É notável a dificuldade encontrada em acompanhar matematicamente o funcionamento dos modelos e garantir o entendimento visual dos passos envolvidos nos cálculos até chegar no resultado esperado e no processo de avaliação. Essa dificuldade causa um entendimento subjetivo de todo o universo de operações com os modelos, e consequentemente dificulda do trabalho do educador e do aluno. Ao realizar um processo de revisão sistemática da literatura, usando artigos científicos nos principais indexadores como Scopus, Elsiever e IEEE, para busca de ferramentas, softwares ou frameworks voltados para visualização e ensino dos processos de treinamento em modelos de Deep Learning, podemos encontrar algumas soluções existentes como por exemplo as bibliotecas BertViz e Ecco em Python, que lidam com processamento de linguagem natural. A maior parte da literatura existente, assim como o BertViz ou Ecco, abordam o tema em uma perspectiva teórica do problema e de uso da ferramenta. Os frameworks existentes conseguem ser usados de forma fácil e objetiva para representação de resultados finais do processo de treinamento em modelos de Deep Learning, usando como base uma execução em um Notebook e uma interação mínima com usuário nas saídas. Porém, as ferramentas existentes não exploram integrações com outras bibliotecas e real capacidade didática para acompanhar o raciocínio matemático nos modelos, com um passo a passo mais compreensível. Além disso, em todos os softwares analisados, temos um alto grau de acoplamento com saídas de execução em células de um Notebook apenas, dificultando integrações com outras plataformas ou uma aplicação Web para o usuário. Este Projeto Final de Graduação tem como principal objetivo o desenvolvimento de um Software que funcione como um arcabouço para visualização e ensino de modelos de aprendizado profundo, através de um canal de comunicação com o usuário onde os parâmetros de treinamento do modelo podem ser

inseridos de forma flexível e o resultado do processo de treinamento e avaliação pode ser exibido em tela através de gráficos e diagramas explicativos. A ferramenta selecionada para aplicação no projeto foi a biblioteca BertViz em Python, que propõe uma demonstração visual do mecanismo de atenção em redes de Transformadores. Construímos um front-end em React para comunicação com usuário, passagem de parâmetros de treinamento e visualização gráfica do resultado dos modelos e as operações, e um back-end em Python responsável pelo tratamento da informação e treinamento do modelo usando um Jupyter Notebook. Como resultado do projeto, temos uma aplicação Web que fornece uma representação visual dos pesos de atenção em um modelo Transformer simples, com uma camada de atenção, onde o usuário seleciona o texto de entrada, o número de cabeças de atenção, a dimensão para o embedding e o modo de visualização do BertViz. Este desenvolvimento foi importante para consolidação prática e revisão de conceitos de Deep Learning e apresentou um bom resultado para a proposta, que era justamente a representação visual para ensino com modelos de aprendizado profundo. Por enquanto, a aplicação ainda se encontra em ambiente local, com perspectivas para uma evolução usando uma integração em nuvem e treinar o modelo em um ambiente remoto com maior capacidade e recursos. Além disso, temos perpectiva de evolução para especializar mais o resultado visual esperado, exibindo os passos intermediários para cálculo das informações Query, Key e Values, bem como uma representação do embedding para dimensões menores. Por ultimo, espera-se que a aplicação possa ser generalizada para representar visualmente outros tipos de modelo como CNNs e RNNs por exemplo.

1 Introdução

Trabalhar com Machine Learning é um assunto inevitavelmente trabalhoso na área da Computação e Ciência de Dados, principalmente devido a complexidade envolvida na criação dos modelos, processo de treinamento e escalabilidade dos dados envolvidos que alimentam esse processo. A complexidade tende a aumentar quando entramos no ramo de Aprendizagem Profunda, em modelos como Transformadores, MLP, CNN, RNN e Encoders, Decoders. É notável a dificuldade encontrada em acompanhar matematicamente o funcionamento dos modelos e garantir o entendimento visual dos passos envolvidos nos cálculos até chegar no resultado esperado e no processo de avaliação, principalmente na área acadêmica e ensino em disciplinas de Maching Larning ou áreas correlacionadas. Quando temos um baixo número de neurônios envolvidos nas redes neurais e poucas camadas no modelo, a representação visual do problema pode ser realizada de forma simples com desenhos demonstrativos, ao contrário de cenários onde encontramos um número exponencial de parâmetros envolvidos. Para modelos dessa natureza, uma simples representação por desenho é inviável de ser compreendida e até mesmo formulada, devido a ordem de grandeza envolvida no processo, mesmo com auxílio de bibliotecas já conhecidas como o Keras ou TensorBoard.

Parte da metodologia deste projeto final de graduação envolveu uma revisão sistemática da literatura, com artigos científicos nos principais indexadores como Scopus, Elsiever e IEEE,

para busca de ferramentas, softwares ou frameworks voltados para visualização e ensino dos processos de treinamento em modelos de Deep Learning. A maior parte da literatura existente aborda o tema em uma perspectiva teórica do problema, com uma introdução ao uso da ferramenta e explicações matemáticas dos algoritmos usados para treinamento do modelo, metodologias para explicar de forma mais clara a representação de redes neurais complexas e seus devidos parâmetros, representações melhores dos cálculos envolvidos para obtenção dos valores dos parâmetros, representação do algoritmo de Backpropagation e uma análise do potencial dessas ferramentas em cenários teóricos.

De todos os artigos científicos levados em consideração na revisão sistemática da literatura, apenas a biblioteca BertViz em Python apresentou real potencial prático para resolver o problema com a visualização do processo de treinamento e uma representação estruturada do modelo. A maior parte desses artigos não explora o potencial técnico da ferramenta para uma implementação objetiva, integrações com outras bibliotecas e real capacidade didática para acompanhar o raciocínio matemático nos modelos, com um passo a passo compreensível. O BertViz em especial explora melhor o mecanismo e as camadas de atenção, e o processo de tokenização através de uma comunicação com o usuário mais explicativa, porém ainda assim não é suficiente para um potencial didático efetivo com o usuário final, visto as limitações arquiteturais da ferramenta para uso através de execução de código em células de um Notebook necessariamente, sem expor uma interface flexível e aberta para o usuário como uma aplicação Web ou estrutura similar. Além de manter um acoplamento alto com um notebook, o BertViz não expõe de forma clara e visível algumas estruturas internas no processo de treinamento do Transformer, como os valores de Query, Key, Values e o embedding, visto que ele só se preocupa com o processo e resultado final após todas as iterações necessárias no algoritmo de treinamento e descarta representações gráficas das operações parciais.

Dado as limitações exploradas nos artigos da comunidade acadêmica sobre as ferramentas para representação de modelos de Deep Learning e as restrições impostas pela utilização do BertViz conforme análise acima, este Projeto Final de Graduação teve como principal inspiração o desenvolvimento de um arcabouço/plataforma para visualização e ensino de modelos de aprendizado profundo, de forma acessível para um usuário qualquer e disponibilização do material implementado para a comunidade acadêmica majoritariamente, principalmente para os cursos de computação e aulas com profissionais na aŕea de Machine Learning ou áreas correlacionadas. O modelo para Tranformer foi adotado como base para potencial didático na construção da aplicação, usando o BertViz como principal ferramenta auxiliar para a representação visual do resultado para os pesos de atenção na camada definida pelo processo de treinamento, bem como promover uma interação mínima com o usuário em tela. Para a organização e implementação efetiva da arquitetura e código do projeto, foi desenvolvida uma aplicação front-end em React e outra aplicação back-end Python, assim como comunicações com o Jupyter Notebook, pytorch para criação do modelo

e dos métodos de treinamento e integrações com a lib BertViz mencionada anteriormente.O restante deste artigo está organizado da seguinte maneira: na Seção 2, mostramos alguns trabalhos relacionados, obtidos através de uma revisão sistemática da literatura; na Seção 3, apresentamos a metodologia adotada no trabalho, incluindo os processos abordados para o desenvolvimento do projeto, provas de conceito, arquitetura, fluxogramas, estrutura da aplicação sugerida e implementação do código; na Seção 4 mostramos os resultados obtidos ao longo do projeto, mais especificamente testes com a aplicação já desenvolvida; por fim, na Seção 5 fazemos as considerações finais sobre o trabalho desenvolvido, recapitulando as contribuições do trabalho e projetando as próximas etapas desta pesqusa.

2 Trabalhos Relacionados

Para a fase de revisão da liteatura neste projeto final de graduação, foram analisados vários artigos científicos, papers e conferências nos principais indexadores do mercado (Elsiever, Scupus, ACM, IEEE, Springer). O diagrama prisma foi utilizado para realização de uma coletânea com o número de artigos encontrados e o número de artigos filtrados após processo de refinamento com o real conteúdo de interesse para esse projeto. O objetivo foi encontrar conteúdos que demonstrassem de forma clara e objetiva ferramentas, frameworks ou metodologias para visualização e ensino de modelos de aprendizado profundo, de forma didática e com potencial para explorar as ferramentas e bibliotecas necessárias em Software para a contrução dessas aplicações. Para o processo de organização da leitura dos artigos encontrados, foi utilizado a plataforma ZOTERO, onde os principais papers de relevância para a nossa análise foram cadastrados (título, autor e anexo PDF com o conteúdo) e observações relevantes e/ou resumos foram adicionados. Para o processo de pesquisa nos indexadores, foi utilizada a busca por termos com palavras-chave e condicionais AND e OR, de acordo com o conteúdo de interesse. Exemplo: ("Deep Learning"OR "Neural Networks") AMD ("Visualization"OR "Education"). A Figura 1 demonstra de forma clara um diagrama no PRISMA com os resultados encontrados no processo de revisão da literatura, com os devidos números para cada processo de refinamento da pesquisa:

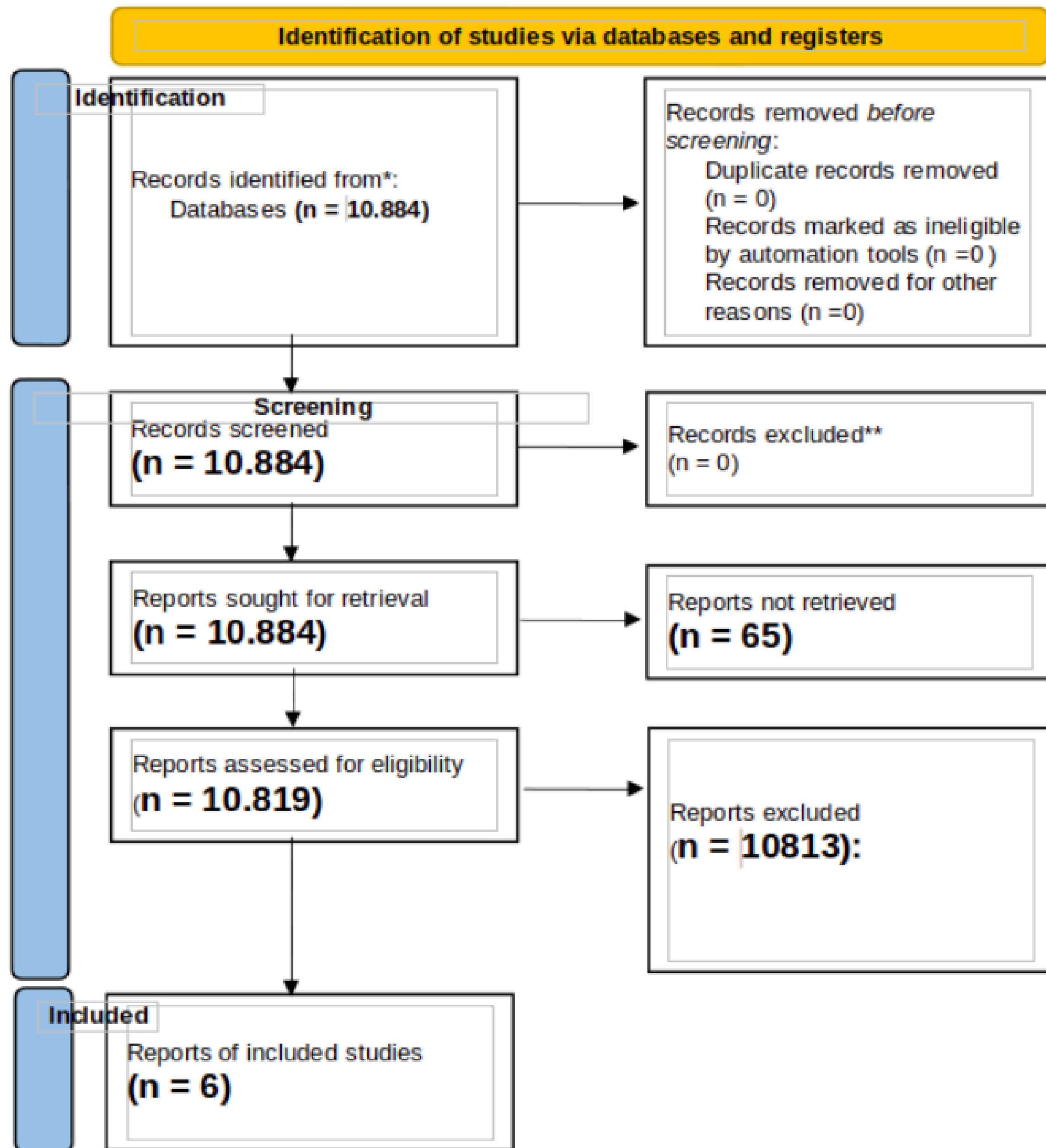

Figura 1: Diagrama Prisma com resultado encontrado no processo de revisão da literatura

Segue abaixo um resumo e uma analise crítica dos principais artigos e papers encontrados aderentes com a proposta para esse projeto final de graduação:

2.1 BertViz: A tool for visualizing Multi-Head Self-Attention in the Bert Model

Esse paper introduz o BertViz, uma ferramenta de software open-source para visualização do processo de self-attention seguindo o modelo da linguagem de representação BERT. A ferramenta proporciona uma visualização do processo de attention em três níveis de granularização: nível attention-head, nível do modelo e nível do neurônio na rede neural. O repositório oficial no github pode ser encontrado no link `https://github.com/jessevig/bertviz?tab=readme-ov-file`. Além disso, um vídeo prático demonstrativo para a plataforma pode ser encontrado em `https://www.youtube.com/watch?v=187JyiA4pyk`. A ferramenta pode ser utilizada por um notebook no Collab ou Jupyter a partir de uma API Python simples que suporta modelos Hugging Face (`https://huggingface.co/models`). Segue abaixo uma descrição referente aos 3 modos de visualização disponíveis do BertViz, ponto-chave para a utilização eficiente da ferramenta

Modo de visualização Head-View No Head-View, apresentamos uma visualização para um ou mais Attention-Heads em uma mesma camada na rede neural e os padrões envolvidos, conforme mostra a Figura 2. O propósito desse modo de visualização é mostrar como o fluxo de attention é distribuído entre os tokens para uma determinada camada/head

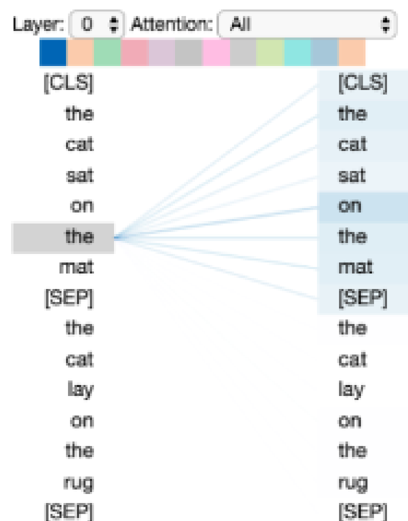

Figura 2: Modo de visualização HeadView no BertViz

Para um processo de visualização personalizado no Head-View, o usuário pode selecionar a camada desejada (Layer), um ou mais attention-heads e o sentence-level attention filter (filtro para attention da sentença de origem para a sentença de destino). O usuário também poderá ter a opção de filtrar o attention por um determinado token. O self-attention é representado através das linhas azuis que conectam os tokens de origem (esquerda) com os tokens de destino (direita) e a espessura das linhas de conexão repreenta o Attention Score para o problema especificado

Modo de visualização Model-View O Model-View disponibiliza um modo de visualização "bid's eye"do attention entre todas as camadas e heads do modelo. Os attention heads são disponibilizados para visualização através de um formato tabular, onde as linhas representam as camadas e as colunas representam os heads. Cada par camada/head é representado através de um formato thumbnail que transmite o formato do padrão do attention. O usuário pode também clicar em qualquer head no formato tabular e visualizar os tokens envolvidos, conforme demonstra a Figura 3. O Model-View possui como objetivo principal habilirar que os usuários consigam facilmente pesquisa todos os head-attention e visualizar como os padrões do attention estão sendo gerados e distribuídos entre as camadas do modelo.

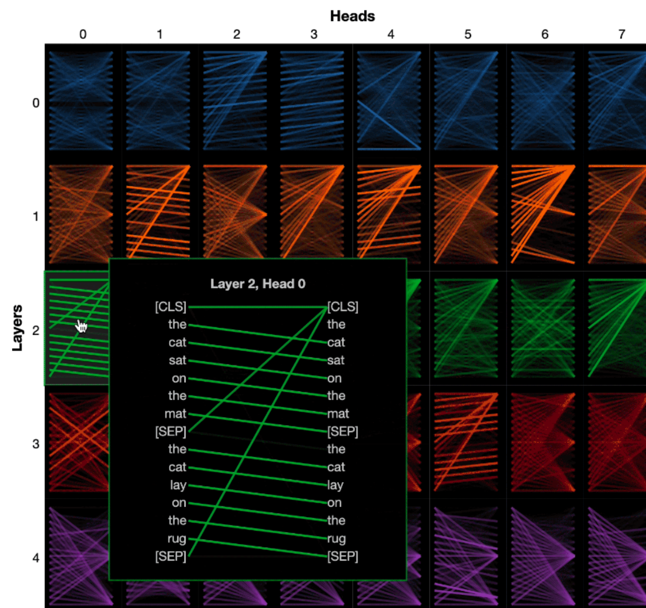

Figura 3: Modo de visualização ModelView no BertViz

Modo de visualização Neuron-View O Neuron-View mostra a visualização dos neurônios individuais na query e vetores-chave e mostra como eles interagem entre si para produzir os scores de attention. Dado um determinado token selecionado pelo usuário, esse modo de visualização mostra o processo de computação do attention desse token para os demais tokens de destino na sequência (direita), conforme demonstra a Figura 4. Enquanto o modo de visualização Model-View mostra quais padrões de attention o BERT está apredendndo, o Neuron-View se preocupa em mostrar como o BERT está formando esses mesmos pedrões, de forma visual.

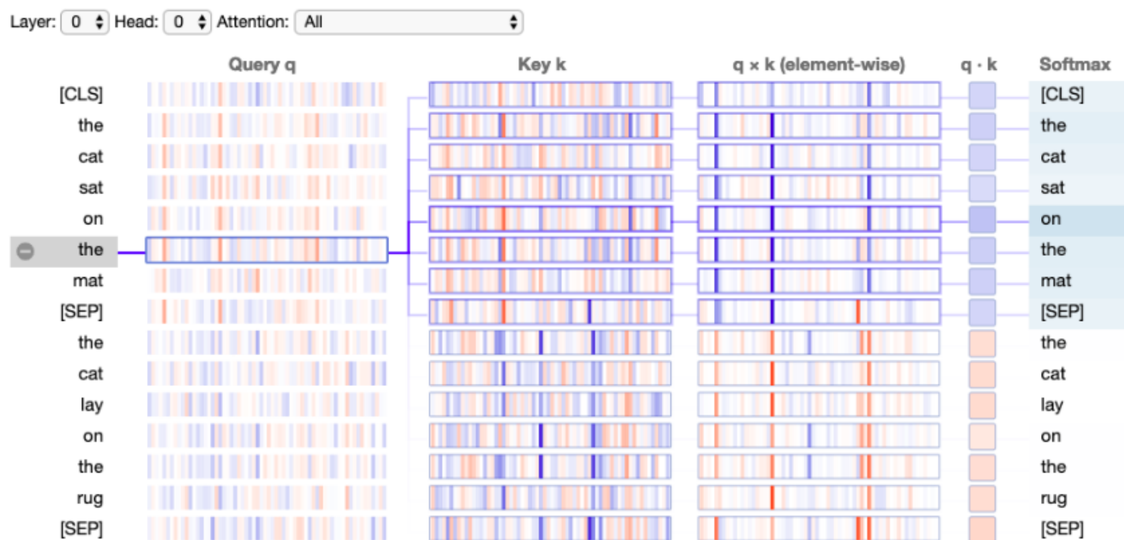

Figura 4: Modo de visualização NeuronView no BertViz

Temos um exemplo de uso da lib BertViz em Python, demonstrando o processo de renderização da informação em um Notebook no Collab, com um modelo pré treinado, na visualização Neuron View, conforme mostra a Figura 5.

```
from bertviz.transformers_neuron_view import BertModel, BertTokenizer
from bertviz.neuron_view import show

model_type = 'bert'
model_version = 'bert-base-uncased'
model = BertModel.from_pretrained(model_version, output_attentions=True)
tokenizer = BertTokenizer.from_pretrained(model_version, do_lower_case=True)
show(model, model_type, tokenizer, sentence_a, sentence_b, layer=4, head=3)
```

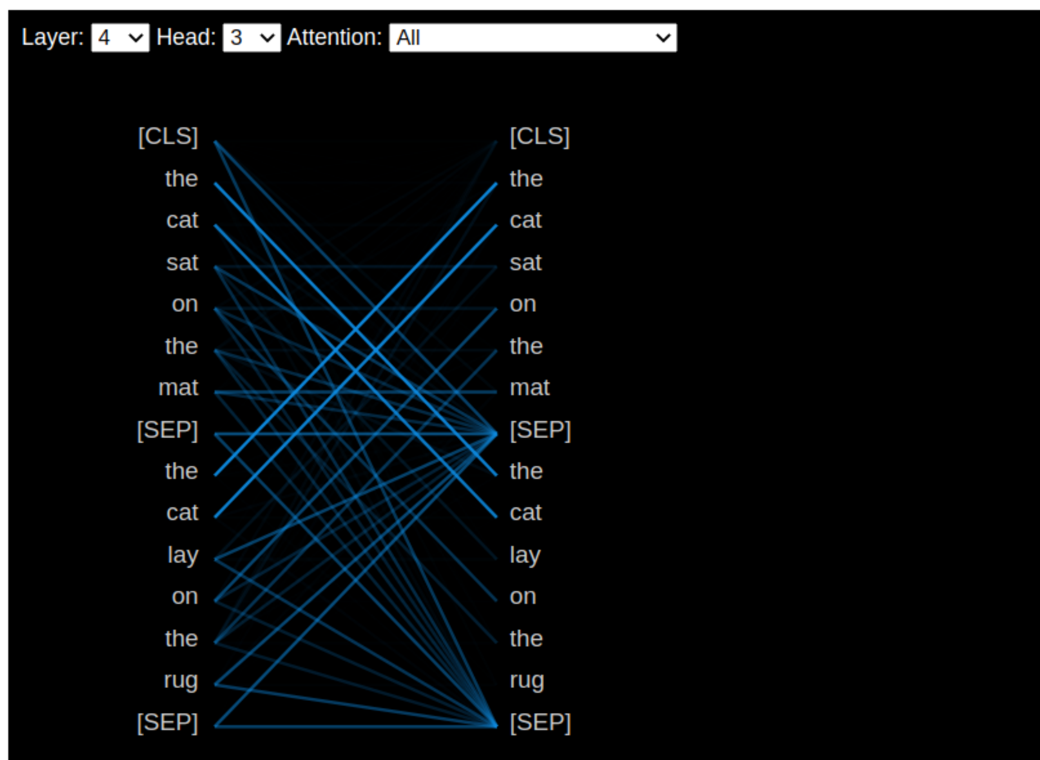

Figura 5: Exemplo de uso da lib BertViz com modelo pré-treinado

2.2 Ecco: An Open Source Library for the Explainability of Transformer Language Models

O artigo apresenta o Ecco, uma biblioteca de código aberto projetada para melhorar a explicabilidade de modelos de linguagem baseados em Transformer. Ele fornece ferramentas para capturar, analisar e visualizar os mecanismos internos desses modelos, ajudando pesquisadores e desenvolvedores a compreender melhor seu funcionamento.

Funcionalidades

- 1. Atribuição de Saliência de Entrada:
  - ○ Analisa a importância de tokens de entrada para a geração de tokens de saída.
  - ○ Visualizações interativas destacam os tokens mais influentes em cada previsão do modelo.
- 2. Exame de Estados Ocultos:
  - ○ Permite explorar a evolução dos estados ocultos ao longo das camadas do modelo.
  - ○ Ferramentas como Análise de Correlação Canônica (CCA) e variações, como SVCCA e PWCCA, permitem comparar representações internas.
- 3. Ativações de Neurônios:
  - ○ Monitora como os neurônios respondem aos tokens de entrada e contribuem para os resultados.
  - ○ Ferramentas de redução de dimensionalidade, como Fatoração de Matrizes Não-Negativas (NMF), ajudam a identificar padrões subjacentes nas ativações.
- 4. Exploração Interativa:
  - ○ Interfaces interativas dentro de notebooks Jupyter permitem análise visual detalhada.
  - ○ Suporte a modelos como GPT-2, BERT e RoBERTa, com expansão planejada para outros modelos.

Estudos

- Probing Classifiers: O Ecco foi usado para explorar as ativações de neurônios em camadas Feed-Forward para identificar informações de classe gramatical (PoS) nos dados.
- Visualização de Rankings de Tokens: Ferramentas permitem rastrear como previsões de tokens evoluem através das camadas do modelo, ajudando a entender decisões complexas, como acordos sujeito-verbo.

Conclusão O Ecco fornece uma plataforma robusta para analisar e interpretar modelos baseados em Transformer. Ele combina facilidade de uso, visualizações ricas e métodos avançados de explicabilidade, contribuindo para maior transparência e compreensão no

aprendizado de máquina. A biblioteca é de código aberto e incentiva contribuições da comunidade para expandir suas capacidades.

2.3 A User-Based Taxonomy for Deep Learning Visualization

O artigo apresenta uma abordagem para categorizar ferramentas e métodos de visualização no aprendizado profundo, destacando sua relevância para melhorar a interpretabilidade de modelos complexos. A proposta inclui uma taxonomia baseada nos tipos de usuários e seus objetivos específicos, assim como as arquiteturas de redes mais comuns. O aprendizado profundo revolucionou áreas como reconhecimento de imagens, tradução automática e sistemas de resposta inteligente, superando métodos tradicionais de aprendizado de máquina. Apesar dos avanços, a falta de interpretabilidade é um grande desafio, especialmente em áreas críticas como medicina e finanças, onde transparência e confiabilidade são essenciais. A visualização é destacada como uma solução para superar a percepção de "caixa preta"dos modelos de aprendizado profundo.O artigo categoriza as ferramentas de visualização em quatro grupos principais de usuários:

- 1. Iniciantes: Ferramentas como TensorFlow Playground e Deep Visualization Toolbox ajudam a ensinar conceitos básicos de aprendizado profundo. Foco em criar uma intuição visual de como as redes funcionam.
- 2. Praticantes: Métodos como TensorBoard e CNNVis auxiliam na avaliação de arquiteturas de redes neurais. Fornecem insights sobre como os dados fluem pelas camadas e como as conexões entre neurônios são estruturadas.
- 3. Desenvolvedores: Ferramentas como ActiVis e DGMTracker ajudam a depurar e melhorar modelos durante o treinamento. Incluem visualizações de pesos, gradientes e métricas, facilitando a detecção de problemas no modelo.
- 4. Especialistas: Estudos focam na explicação visual, com métodos como Grad-CAM e mapas de saliência. Visam interpretar os resultados do modelo e identificar quais partes dos dados influenciam as decisões.

Objetivos

- Ensinar Conceitos: Focado em iniciantes, com ferramentas interativas para demonstrar como redes neurais operam.
- Avaliar Arquiteturas: Para praticantes, visualizando a estrutura das redes e o fluxo de dados.
- Depurar e Melhorar Modelos: Auxiliar desenvolvedores no ajuste fino de parâmetros e na otimização de desempenho.

- Explicação Visual: Permitir que especialistas interpretem as decisões dos modelos, aumentando a confiabilidade e a transparência. Principais Arquiteturas de Redes
- DNN (Redes Neurais Profundas)
- CNN (Redes Neurais Convolucionais): Com foco em tarefas de visão computacional.
- RNN (Redes Neurais Recorrentes): Aplicadas em processamento sequencial.
- DGM (Modelos Gerativos Profundos): Usados em aprendizado não supervisionado, como GANs e VAEs. Desafios e Oportunidades Futuras
- Melhorar Interpretabilidade: Criar métodos mais robustos e confiáveis para explorar os modelos em profundidade.
- Sistemas Progressivos de Análise: Desenvolver ferramentas que combinem visualização interativa com análise detalhada.
- Redes Avançadas: Adaptar métodos de visualização para arquiteturas mais complexas, como DenseNet e redes seq2seq.

Conclusão

O artigo apresenta uma visão abrangente do campo emergente de visualização para aprendizado profundo, destacando a necessidade de métodos acessíveis e confiáveis para usuários com diferentes níveis de experiência. Além disso, ele enfatiza a importância da explicação visual como um passo crítico para superar a falta de interpretabilidade nos modelos.

Impactos

- Facilita o aprendizado de conceitos por iniciantes.
- Melhora a eficiência e a transparência no desenvolvimento de modelos complexos.
- Amplia a adoção de aprendizado profundo em domínios que exigem alta confiabilidade.

2.4 The Educational Resource Management Based on Image Data Visualization and Deep Learning

O artigo explora o impacto do uso de técnicas de visualização de dados e redes neurais convolucionais (CNNs) em sistemas de gestão de recursos educacionais (ERMS). Esses métodos visam resolver problemas comuns como localização imprecisa de recursos e baixa eficiência na utilização. O estudo combina essas tecnologias para otimizar o ERMS, analisando sua aplicação, desafios e desempenho experimental.

Objetivos do Estudo

- Analisar o papel do ERMS: Centralizar e organizar recursos educacionais para facilitar o acesso e compartilhamento.
- Aplicar técnicas de visualização e CNNs: Melhorar a precisão e eficiência do gerenciamento de recursos.
- Validar o modelo proposto: Realizar experimentos para comparar o desempenho do modelo otimizado com métodos tradicionais.

Principais Descobertas

- Papel do ERMS na Educação: Centralização de recursos em uma única plataforma, Promoção do compartilhamento e personalização de materiais educacionais. Melhor suporte a professores e alunos para ensino e autoaprendizado.
- Aplicação de CNNs: Extração de características de imagens, como bordas e texturas, para classificar e localizar recursos. Uso de funções de ativação avançadas, como PReLU, para melhorar a precisão e estabilidade do modelo.
- Técnicas de Visualização de Dados: Transformação de dados educacionais em gráficos e imagens intuitivas. Melhoria na análise e tomada de decisões por professores e administradores.
- Resultados Experimentais: O modelo otimizado alcançou uma precisão de 98,1

Conclusão
O estudo demonstrou que a integração de visualização de dados e CNNs pode melhorar significativamente a eficiência e precisão do gerenciamento de recursos educacionais. O modelo proposto representa uma solução prática para desafios existentes, contribuindo para a personalização e qualidade do ensino. Contudo, futuras pesquisas devem abordar limitações como segurança e estabilidade do sistema.

2.5 Explaining the Black-Box Model: A Survey of Local Interpretation Methods for Deep Neural Networks

O artigo aborda a crescente necessidade de interpretabilidade em modelos de aprendizado profundo, com foco nos métodos de interpretação local. Esses métodos ajudam a entender como as redes neurais profundas (DNNs) tomam decisões, reduzindo a opacidade do modelo e aumentando a confiança dos usuários.

Principais Contribuições

- Taxonomia de Métodos de Interpretação Local: Baseados em Dados (Data-driven): Métodos que analisam perturbações nos dados de entrada para identificar sua influência nas previsões. Exemplo: LIME, SHAP e métodos baseados em conceitos

como TCAV. Baseados em Modelo (Model-driven): Métodos que analisam componentes internos do modelo, como gradientes e pesos, para fornecer explicações. Exemplo: Grad-CAM e propagação de relevância camada por camada (LRP).

- Reprodução de Resultados: O estudo reproduz diversos métodos de interpretação usando ferramentas de código aberto para avaliar sua eficácia.
- Desafios e Oportunidades: Necessidade de padrões claros para avaliar a confiabilidade das interpretações. Avanço em métodos de interpretação robustos que lidem com perturbações adversas.

Resultados e Discussões

- Métodos baseados em gradientes, como Grad-CAM, são eficazes para identificar áreas importantes em imagens.
- Métodos baseados em conceitos, como TCAV, permitem explicações mais intuitivas, baseadas em características compreensíveis para humanos.
- O artigo destaca o potencial das ferramentas de visualização para melhorar a interpretabilidade e a transparência dos modelos de aprendizado profundo.

Conclusão
O trabalho oferece uma visão abrangente sobre métodos de interpretação local, destacando suas aplicações, limitações e futuros direcionamentos para pesquisas. Ele enfatiza que a explicabilidade é essencial para o uso confiável de DNNs, especialmente em domínios sensíveis como saúde e finanças.

2.6 The LRP Toolbox for Artificial Neural Networks

O artigo apresenta o "LRP Toolbox", uma ferramenta para a interpretação de decisões feitas por redes neurais profundas (DNNs) usando o algoritmo Layer-wise Relevance Propagation (LRP). O LRP fornece explicações sobre como os modelos tomam decisões, atribuindo pontuações de relevância aos componentes do dado de entrada com base na topologia do modelo treinado.

Principais Características do LRP Toolbox

- Funcionalidades: Implementações independentes de plataforma em Python, Matlab e C++ (via framework Caffe), Suporte a redes pré-treinadas, incluindo GoogLeNet e VGG, disponíveis no Model Zoo do Caffe, Capacidade de exportar e importar modelos e dados em formatos variados, como .mat, .npy e texto bruto.
- Objetivos: Fornecer um ambiente acessível para explorar o algoritmo LRP, facilitando a compreensão e a modificação de regras de propagação. Produzir mapas de calor que

explicam como as entradas influenciam as previsões do modelo.

Aplicação do LRP

- O LRP decompõe a função de decisão da rede em relevâncias específicas para cada dimensão de entrada, visualizadas como mapas de calor.
- Essas relevâncias podem ser positivas ou negativas, indicando evidências a favor ou contra a classe predita.

Plataformas e Requisitos

- Implementações disponíveis para Linux, Windows e macOS.
- Ferramentas de linha de comando e interfaces gráficas para Matlab, Python e C++. Casos de Uso
- Ideal para interpretar decisões de redes profundas usadas em visão computacional, processamento de linguagem natural e reconhecimento de fala.
- Pode ser aplicado retroativamente a redes treinadas, permitindo explicar decisões de projetos passados.

Conclusão
O LRP Toolbox é uma ferramenta poderosa e acessível para explorar a interpretabilidade de redes neurais profundas. Ele serve como um passo importante na transparência de modelos complexos, especialmente em aplicações que exigem explicações detalhadas para maior confiabilidade e ética.

2.7 Tutorial on Deep Learning Interpretation: A Data Perspective

O tutorial explora métodos de interpretação em aprendizado profundo, considerando dados em diferentes formatos (imagens, texto e grafos). Ele destaca a crescente adoção de modelos de aprendizado profundo em aplicações de grande escala e os desafios relacionados à sua interpretabilidade, essencial para construir confiança em domínios críticos como saúde, justiça e finanças.

Principais Categorias de Métodos de Interpretação

- Modelos Específicos vs. Agnósticos: Específicos: Dependem da estrutura interna do modelo, como Grad-CAM para redes convolucionais. Agnósticos: Tratam o modelo como uma caixa-preta, como LIME e SHAP.
- Baseados em Perturbação vs. Gradiente: Perturbação: Alteram dados de entrada para medir o impacto nas saídas. Gradiente: Usam gradientes para calcular mapas de saliência e identificar características relevantes.

- Interpretação Local vs. Global: Local: Explicam previsões específicas para entradas individuais. Global: Fornecem uma visão geral sobre a importância das características em todo o modelo.

Aplicações em Diferentes Domínios de Dados

- Imagens: Uso de mapas de saliência para entender quais partes das imagens influenciam as previsões. Métodos como Grad-CAM e SHAP destacam as regiões mais relevantes.
- Texto: Técnicas pós-treinamento avaliam contribuições de palavras ou características. Métodos incluem representações baseadas em mapas de saliência e valores de Shapley.
- Grafos: Integram métodos pós-treinamento, como explicadores baseados em perturbação, para atribuir significados a representações de grafos.
- Aprendizado por Reforço Profundo: Uso de mapas de saliência e decomposição de recompensas para entender decisões de agentes em ambientes dinâmicos.

Público-Alvo
O tutorial é projetado para iniciantes e especialistas com interesse em aprendizado profundo. Ele oferece uma compreensão prática e teórica sobre métodos de interpretação, incluindo suas aplicações e limitações, além de inspirar futuras pesquisas no campo.

2.8 Flow: Differentiable Simulations for PyTorch, TensorFlow, and JAX

O artigo apresenta o Flow, uma ferramenta de código aberto que facilita a implementação de simulações diferenciáveis para aprendizado de máquina (ML) em contextos científicos e de engenharia. O Flow é compatível com bibliotecas como PyTorch, TensorFlow, JAX e NumPy, e inclui uma ampla gama de recursos para simulações baseadas em malhas estruturadas (eulerianas) e partículas (lagrangianas).

Principais Recursos

- Compatibilidade Multiplataforma: Integra-se com PyTorch, TensorFlow e JAX, permitindo alternância entre bibliotecas sem copiar dados. Oferece suporte nativo a GPU/TPU e compilação Just-In-Time (JIT).
- Operadores Diferenciáveis: Implementa operadores como gradiente, divergência, laplaciano e vorticidade para malhas estruturadas e não estruturadas. Suporte a condições de contorno (Dirichlet, Neumann, periódicas) com adaptação automática do esquema numérico.
- Código Dimensionalmente Agnóstico: Permite que o mesmo código funcione em 1D, 2D, 3D ou dimensões superiores sem modificações.

- Precisão de Ponto Flutuante: Gerencia precisão global ou localmente, prevenindo erros relacionados a tipos de dados.
- Resolução de Sistemas Lineares: Inclui solucionadores lineares diferenciáveis (densos e esparsos) com precondicionadores compatíveis com GPU.
- Visualização Simples: Gera gráficos de alta qualidade com uma única chamada, usando Matplotlib ou Plotly.

Exemplos de Aplicação

- Condutividade Térmica: Determinação de composição material por meio de simulação de condução térmica diferenciável. O Flow foi usado para reconstruir a distribuição de condutividade em uma simulação 2D.
- Velocimetria de Imagem de Partículas (PIV): Reconstrução de campos de velocidade a partir do movimento de partículas rastreadas. O Flow permite resolver problemas inversos usando otimizadores como L-BFGS.
- Treinamento de Redes Neurais para Simular Fluidos: Redes U-Net foram treinadas para imitar simulações de fluidos incompressíveis, reduzindo erros de longo prazo.
- Outros Casos de Uso: Simulações de bilhar, empacotamento de esferas e propagação de ondas.

Conclusão
O Flow é uma ferramenta poderosa para criar simulações diferenciáveis que combinam física e aprendizado de máquina. Ele prioriza a modularidade e a legibilidade do código, permitindo integração com bibliotecas populares de ML e acelerando o desenvolvimento de soluções complexas.

2.9 A NoC-based Simulator for Design and Evaluation of Deep Neural Networks

O artigo apresenta o DNNoC-Sim, um simulador de alta precisão para avaliar o desempenho de redes neurais profundas (DNNs) baseadas em Network-on-Chip (NoC). Ele aborda a crescente complexidade computacional das DNNs modernas e os desafios relacionados à interconexão, consumo de energia e latência.

Principais Recursos

- 1. Técnica de "DNN Flattening": Converte operações complexas como convolução e pooling em operações do tipo multiplicação-acumulação (MAC), permitindo maior flexibilidade computacional.

- 2. Método de Fatiamento Dinâmico (Dynamic Slicing): Permite dividir modelos DNN de grande escala em partes menores, viabilizando sua execução em plataformas NoC com recursos limitados.
- 3. Simulador Ciclo-Preciso: Avalia consumo de energia, largura de banda, latência e precisão da classificação em plataformas baseadas em NoC.

Resultados

- 1. Redução de Acessos à Memória Off-Chip: Comparado com aceleradores DNN convencionais, o design baseado em NoC reduziu os acessos à memória externa entre 87
- 2. Eficiência Computacional: Simulações demonstraram que o DNNoC-Sim é eficiente ao equilibrar o uso de memória on-chip e off-chip, ajustando parâmetros como tamanho da NoC e agrupamento de neurônios.
- 3. Análise de Trade-offs: Mostrou que tamanhos maiores de NoC e grupos de neurônios resultam em menor número de acessos à memória, mas aumentam a latência de entrega de dados na NoC.

Conclusão O DNNoC-Sim é uma ferramenta inovadora para projetar e avaliar DNNs em plataformas NoC, oferecendo maior flexibilidade e eficiência computacional. Ele é ideal para simular modelos modernos, como VGG-16 e MobileNet, e facilita a análise de parâmetros críticos para projetar aceleradores eficientes.

2.10 Neural Network Visualization

O artigo "Neural Network Visualization"de Jakub Wejchert e Gerald Tesauro apresenta técnicas gráficas para visualizar informações estáticas e dinâmicas em sistemas de aprendizado de redes neurais baseados em camadas. O objetivo principal é proporcionar insights sobre o processo de aprendizado das redes neurais utilizando o algoritmo de retropropagação.

Principais Contribuições

- 1. Diagramas Visuais:
  - Bond Diagram: Representa a estrutura geométrica de uma rede neural, destacando pesos e topologia da rede. Os pesos são visualizados como triângulos ou linhas que conectam os nós, indicando magnitude e direção.
  - Trajectory Diagram: Mostra a evolução temporal do aprendizado em um espaço de pesos reduzido, representando a trajetória do gradiente descendente em um hiperplano de erro.

- 2. Ambiente de Visualização: Um sistema multi-janela exibe diferentes aspectos da simulação simultaneamente, incluindo erros totais, estados de saída e trajetórias de unidades ocultas.
- 3. Animações: Utilizadas para estudar como os pesos evoluem ao longo do aprendizado e como diferentes parâmetros influenciam as decisões do modelo.

Aplicações Práticas

- 1. Função de Maioria: A rede foi treinada para determinar se a maioria dos nós de entrada estava ativa. A visualização demonstrou pesos uniformes e um grande viés na saída, refletindo a solução.
- 2. Função de Simetria Simples: A rede aprendeu a diferenciar padrões simétricos e antissimétricos, utilizando dois nós ocultos para tomar decisões.
- 3. Função de Simetria Geral: Exigiu a detecção de simetria em um conjunto mais amplo de padrões. A rede adotou uma hierarquia de pesos para resolver o problema, começando com pares externos e avançando para pares internos.

Benefícios observados

- Clareza na Representação Interna: Ajuda a entender como os pesos e parâmetros interagem no aprendizado.
- Detecção de Erros: Visualizações facilitaram a identificação de inconsistências no código ou no processo de aprendizado.
- Insights Sobre Dinâmica de Aprendizado: Revelaram como ajustes iniciais nos pesos podem determinar o sucesso ou fracasso na convergência.

Conclusão O trabalho destaca a utilidade da visualização gráfica na pesquisa e no desenvolvimento de redes neurais, proporcionando novas ferramentas para estudar e depurar modelos. Os autores sugerem que as técnicas poderiam ser estendidas para redes maiores com milhares de nós e pesos, ampliando sua aplicabilidade.

2.11 Investigating the Usability of a New Framework for Creating, Working, and Teaching Artificial Neural Networks Using Augmented Reality (AR) and Virtual Reality (VR) Tools

O artigo aborda um novo framework chamado RKNet, que utiliza ferramentas de Realidade Aumentada (AR) e Realidade Virtual (VR) para criar, treinar e ensinar redes neurais artificiais (ANNs). O objetivo é melhorar a acessibilidade e o entendimento de ANNs por meio de representações visuais interativas, permitindo que até usuários com conhecimento limitado de ciência da computação possam explorar e operar redes neurais. Objetivos

Objetivos

- 1. Identificar as principais dificuldades enfrentadas pelos alunos ao aprender sobre ANNs.
- 2. Definir os requisitos de uma ferramenta visual eficaz para ensinar redes neurais.
- 3. Avaliar a usabilidade do framework RKNet para fins educacionais.

Funcionalidades do Framework RKNet

- 1. Visualização de Redes Neurais em AR/VR: Representação gráfica das redes como grafos direcionados com etiquetas, mostrando conexões, pesos e estados. Interação direta em VR, permitindo manipular, rotacionar e explorar nós da rede.
- 2. Sistema Distribuído: Construção de redes baseadas na linguagem de programação Erlang, com comunicação sem memória compartilhada entre os nós. Redes altamente tolerantes a falhas e que permitem aprendizado colaborativo entre nós.
- 3. Foco em Educação: Ferramentas para criar redes simples para estudantes entenderem conceitos básicos, como backpropagation e função de perda. Permite desacelerar o processo de aprendizado da rede para melhor visualização dos resultados.

Pesquisa Formativa

- Participantes: 5 instrutores universitários e 31 estudantes (26 de graduação e 5 de pós-graduação).
- Descobertas: As principais dificuldades relatadas foram conectar matemática com estrutura das redes, entender o papel de pesos e dimensões, e compreender as camadas de rede. Requisitos para ferramentas visuais incluíram interatividade, explicações de cálculos e animações que ilustram o funcionamento das redes.

Resultados da Avaliação

- Usabilidade:
  - 21 estudantes testaram o framework, avaliando-o positivamente em termos de facilidade de uso e entendimento.
  - As interações em VR receberam a maior pontuação, destacando o aumento no engajamento e na experiência de aprendizado.
- Limitações: Não é possível alterar hiperparâmetros diretamente no ambiente AR/VR, mas essa funcionalidade está planejada para futuras versões.

Conclusão e Trabalhos Futuros

- O RKNet é uma ferramenta promissora para integrar VR e AR no ensino e no uso de ANNs, especialmente para não especialistas.
- Desenvolvimentos futuros incluirão interatividade ampliada, suporte a AR para ambientes reais e lançamento como software de código aberto.

2.12 Posicionamento em Relação aos Trabalhos Relacionados

Os trabalhos relacionados utilizados para leitura e análise nesse projeto final de graduação foram fundamentais para consolidação e entendimento teórico de algumas ferramentas para visualização de modelos de Redes Neurais profundas e particularidades no seu processo de treinamento. A maior parte desses artigos explora o processo de treinamento de algoritmos para Transformers, Mecanismo de Atenção, Camadas de redes neurais convencionais e Redes Neurais Convolucionais (CNNs) e apresenta uma reflexão teórica em cima das ferramentas que auxiliam a representação gráfica dos parâmetros de treinamento, métricas de avaliação, passo a passo dos algoritmos, visualização das camadas de um modelo e desempenho do modelo.

Artigos levados adiante na implementação do projeto - Pontos positivos

Dentre todos os artigos mencionados anteriormente ao longo do processo de revisão da literatura, o BertViz foi o mais aderente com o propósito prático desse projeto final de graduação.

A ferramenta conseguiu explorar visualmente o mecanismo de auto-atenção com múltiplos heads de atenção em um modelo BERT especificamente. Isso foi crucial para entender como o modelo processa entradas e direciona decisões, além de apresentar 3 formas de visualização gráfica diferentes. Dessa forma, foi possível explorar a tokenização de uma sentença e como a atenção é aplicada em cima desses tokens no processo de treinamento e resultado final ao rodar o modelo. O BertViz tem uma utilização bem facilitada através da biblioteca Python, com exposição dos métodos head view, model view e show (para visualização em modo Neurônio, com mais detalhes na camada de rede neural). Além disso, conseguimos passar duas sentenças de texto comparativas como entrada dos métodos, e explorar como são organizados os pesos de atenção. A interface renderizada pelo BertViz no Jupyter Notebook apresenta um HTML contendo um seletor para o usuário identificar qual será a camada de atenção desejada para visualização e qual de-para de sentença do texto. Além disso, é possível selecionar os tokens individualmente para observar o padrão de atenção. Com o uso do BertViz, conseguimos gerar resultados de saídas gráficas seguindo esses padrões de exibição após execução das células com utilização de um modelo BERT pré treinado como

teste ou treinamento de um modelo próprio de Transformer com Multi-Head Attention e definição dos parâmetros necessários para esse processo de treinaento como número de heads de atenção, tamanho da sentença, dimensão para o embedding, número de iterações do algoritmo (epochs) e número de camadas de atenção. Além disso, um ponto positivo do BertViz é a sua utilização prática para renderização do resultado do modelo e descoplado do processo de treinamento, que poderá ser realizado de forma indepnedente ou exportado a partir do modelo pré-treinado BERT. Justamente pela sua arquitetura desacoplada, o BertViz pode ser usado em conjunto com outros frameworks Python como por exemplo o Keras ou Python, já que ele espera apenas uma matriz de 4 dimensões e valores numéricos como parâmetro de entrada para interpretação dos pesos de atenção.

Para análise crítica da ferramenta BertViz, após todas as análises teóricas necessárias do artigo, foi criado uma prova de conceito através de uma implementação de uso do BertViz no Collab, onde obtemos um tokenizador e um modelo pré treinado do BERT, duas sentenças de texto para tokenização. A implementação é open source e se encontra nesse link:

`https://colab.research.goo\breakgle.com/drive/`
`1hXIQ77A4TYS4y3UthWF-Ci7V7vVUoxmQ?usp=sharing`

A partir desse teste, conseguimos verificar a comprovar a eficácia para utilização do BertViz neste projeto final de graduação e validar o resultado esperado nas células.

Artigos levados adiante na implementação do projeto - Pontos limitantes

Por mais que o BertViz seja a ferramenta analisada pela revisão da literatura que mais comprovou eficácia com o propósito do projeto final de graduação, ela apresentou algumas limitações que foram encontradas ao longo do desenvolvimento da aplicação principal no projeto e rodar os métodos necessários da bilioteca no Jupyter Notebook (independentemente de onde estamos rodando, seja no Collab de forma remota ou localmente na máquina para testes):

- Problema em relação a dimensionalidade na estrutura dos pesos de atenção gerados pelo treinamento de um Tranformer próprio (independentemente do número de heads de atenção computados como parâmetros de entrada) em comparação com a dimensionalidade dos pesos de atenção esperado como entrada para os métodos do BertViz que renderizam o output com as simulações de atenção e os tokens. Foi observado que essas estruturas estavam divergentes. O BertViz espera uma lista de 4 dimensões com os pesos da atenção, além dos devidos tokens gerados pelas sentenças finais passadas como entrada. O treinamento o modelo próprio de Transformer com camdas de MultiHead-Attention constrói a estrutura dos pesos de atenção a partir de manipulação de matrizes e obtenção dos valores de Query, Key e Values (necessários para

o Tranformador) no método forward para o treinamento do modelo. O resultado após treinamento é uma estrutura de pesos de atenção com 3 dimensões apenas, conforme demonstrado abaixo:

Processo de construção das estruturas Query, Key e Values

```
Wq = nn.Linear(embed_size, embed_size)
Wk = nn.Linear(embed_size, embed_size)
Wv = nn.Linear(embed_size, embed_size)

def forward(self, x):
    x = self.embedding(x)
    x = x.transpose(0, 1)

    batch_size, seq_len, d_model = x.shape

    Q = self.Wq(x)
    K = self.Wk(x)
    V = self.Wv(x)

    output, attention_weights = self.attention(Q, K, V)
    ....
    ....
```

Onde Wq, Wk e Wv são estruturas lineares usadas como base e Q, K e V são as estruturas obtidas para Query, Key e Values, respectivamente, no processo de treinamento do Transformer, e que são usadas na obtenção dos pesos de atenção. Podemos observar a estrutura dos pesos de atenção que está sendo montada segue uma dimensionalidade [seq len, batch size, embed size], ao contrário da dimensionalidade e formato esperado pela entrada no BertViz:

Processo de renderização dos pesos de atenção no BertViz

```
from bertviz import head_view

def visualize_attention(attention_weights, tokens,
        sentence_b_start):
    head_view(attention_weights, tokens, sentence_b_start)
```

A complexidade dessas estruturas no processo de treinamento e no formato esperado pelo BertViz ocasionou pontos de dificuldade ao longo da implementação do projeto. A solução adotada foi transformar a estrutura de attention weights ao longo do treina-

mento e deixá-la compatível com um formato que o BertViz aceitasse como parâmetro de entrada e consequentemente ao tentar obter cada camada de MultiHead-Attention dessa lista com objeto Tensor.

- O BertViz trabalha apenas com modelos do tipo BERT, mais especificamente funciona com o tokenizador BertTokenizer da biblioteca transformers em Python. Isso limita a construção e treinamento de outros modelos próprios de Transformer que não sigam o modelo BERT, de forma a conseguir utilizar a ferramenta para visualização gráfica dos pesos de atenção. Para esse projeto final de graduação, seguimos com o modelo do tipo BERT e o BertTokenizer:

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

- A renderização dos pesos de atenção no BertViz é de certa forma engessada e limitada para explicar o funcionamento interno do processo de treinamento, como por exemplo exibição visual dos Embeddings conforme a dimensão especificada (com uma provável redução de dimensionalidade caso necessário e caso o número de dimensões seja maior que 3) e a exibção das estruturas para Query, Key e Values que transformamos com as operações lineares ao longo do treinamento. Além disso, ele desconsidera o passo a passo do treinamento em iterações, apenas o resultado final com os pesos de atenção e outputs gerados. O BertViz é uma ótima ferramenta para ser utilizada em conjunto com libs padrões como o Keras e tensorboard que conseguem trazer representações visuais mais minuciosas dentro dos loops de treinamento e uso dos conjuntos de treinamento, teste e validação por exemplo.

- As saídas visuais do BertViz são adaptadas para implementações e execução de código de treinamento em Notebooks ou Collab, nas células de execução. A saída esperada é um trecho de HTML e Javascript contendo uma renderização integral da representação esperada com os pesos de atenção e a possibilidade de manipulação com o usuário em cima desse HTML (para as operações de seleção da camada de atenção por exemplo). Isso dificuldade integrações da lib com serviços externos ou até mesmo embedar o resulatado gráfico do BertViz em uma aplicação web client-side por exemplo, já que as operações e resulatdos finais são renderizadas server-side. Para resolver esse problema, no projeto foi feito uma detecção de sinais de execução nas células do Jupyter Notebook, de forma a receber uma mensagem de finalização de execução do código na célula e captura do output recebido após a execução em formato string (contendo o HTML gerado pelo método do BertViz). Isso foi intermediado através de uma comunicação entre um servidor Python e o Jupyter Notebook através do jupyter-client em Python. O resultado gerado pelo BertViz em string foi convertido para um conteúdo HTML, devolvido ao client front-end desenvolvido para o projeto e no final do processo renderizado na aplicação web para que o usuário conseguisse ter

essa visualização na página web, diretamente pelo Browser, conforme os parâmetros de entrada para o modelo.

3 Metodologia

O processo de desenvolvimento da aplicação foi dividido em algumas etapas com o objetivo de facilitar a organização dos módulos do projeto e ter uma boa estruturação de arquitetura inicial para sustentar as soluções pretendidas. Como um primeiro passo que compõe a metodologia para elaboração da aplicação, foram implementadas duas provas de conceito no próprio Collab para estressar o potencial teórico e didático do nicho a ser explorado na plataforma. A primeira prova de conceito foi o desenvolvimento de um modelo Transformer com uma camada de Multi-Head Attention e um número flexível de heads de atenção e dimensionalidade do embedding, com o objetivo de extrair informações importantes no processo de treinamento do Transformer e as operações parciais envolvidas ao longo das iterações do algoritmo. A segunda prova de conceito foi o desenvolvimento de um modelo simples de Transformer com apenas 1 camada e 1 head de atenção, para representação visual dos pesos de atenção através de um teste com a lib BertViz pretendida para implementação da aplicação final. O objetivo da segunda prova de conceito foi estudar o funcionamento do BertViz e as possibilidades para representação de um modelo após treinamento, estrutura de dados e transformações necessárias envolvidas, e real potencial didático da biblioteca.

As provas de conceito serviram como base para criação dos algoritmos finais para treinamento do modelo Transformer e integração com a lib do BertViz para representação visual do resultado. Toda a parte lógica do treinamento, envio de parâmetros e capturação do output gerado pelo BertViz foi desenvolvida na aplicação back-end Python do projeto e uma comunicação com o Jupyter Notebook. Toda a interface visual com o usuário através da exposição uma página web foi desenvolvida na aplicação front-end React do projeto. As interações em tempo real entre as ações solicitadas pelo usuário no front-end e as operações de treinamento e obtenção do resultado final no back-end foram implementadas através de um canal de comunicação WebSocket entre o servidor e o client. As comunicações mencionadas fazem parte do escopo de funcionamento geral na arquitetura do projeto, conforme demonstrado na Figura 6, onde representamos em alto nível as 3 pontas principais para a compreensão completa da arquitetura da aplicação. Cada uma das integrações podem ser visualizadas na 6 a partir dos círculos em vermelho:

- Integração 1 : Interface com o usuário final mediante exposição de uma aplicação Web de forma flexível para passagem do texto a ser tokenizado e todos os parâmetros de treinamento, assim como a renderização do BertViz
- Integração 2 : Comunicação WebSocket entre o client front-end e o servidor Python para tráfego de mensagens em tempo real em cima das operações solicitadas

- Integração 3 : Algoritmo de treinamento do modelo Transformer conforme os devidos parâmetros e captura do output gerado pelo BertViz com os pesos de atenção

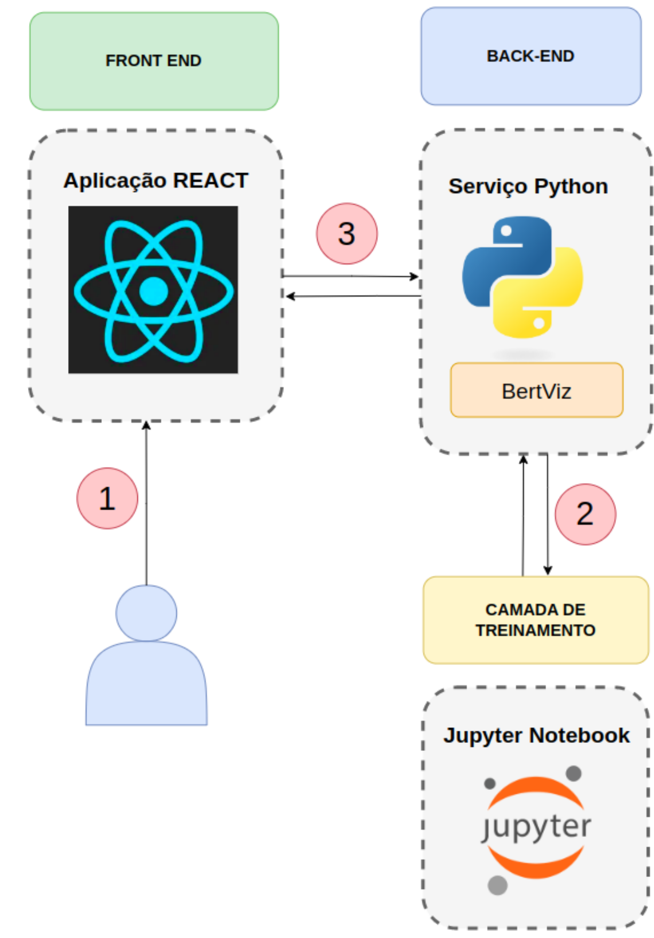

Figura 6: Fluxograma de funcionamento geral da aplicação em alto nível

As próximas sub-seções da Metodologia descrevem com um maior nível de detalhamento sobre as 2 provas de conceito implementadas, a estruturação de arquitetura e código da aplicação conforme as camadas envolvidas e uma visão de fluxograma das operações com usuário final na página web, treinamento do modelo e exibição da representação final.

3.1 Prova de Conceito 1 - Implementação de um modelo Transformer

Antes de iniciar de fato a implementação desse projeto final de graduação, construí duas provas de conceito no Collab. Uma delas foi para movimentar um teste de criação de um modelo Transformer com heads de atenção de uma camada MultiHead-Attention. O objetivo foi exercitar o processo de treinamento do modelo e tokenizar uma sentença de texto para entrada. Segue abaixo o link para essa implementação:

`https://colab.research.google.com/drive/1BCEaYN1rZlLxvUHUPZkJf9p59rVI8-Ad#scrollTo=hniyOX8zszpi`

Implementação no Collab: Camadas necessárias para o Transformer

- Embedding: transformação de uma determinada palavra para um vetor com pesos numéricos
- Positional Enconding: geração de vetores posicionais para cada palavra, representação vetorial de uma determinada frase e determinar informações sobre as posições das palavras no contexto da frase
- Self Attention: Mecanismo adotado para aprendizado contextual na frase, dado as relações entre as palavras na sentença. Modificamos as representações vetoriais das palavras, de forma que elas incluam seus significados contextuais.
- Multi-Head Attention: Mecanismo de atenção com múltiplos reconhecimentos de padrões de atenção na sentença
- Camada MLP: rede neural densa para treinamento da informação
- Layer Normalization e Residual Connection: normalização e camada residual após saída esperada pela camada MLP
- Decoder: processamento dos embeddings, camadas de atenção multi-head e MLP
- Transformer: processamento final da sentença usando o Decoder
- Pre-processamento da informação: tokenização da sentença e preparação para uso do Transformer
- Treinamento: Uso do transformer e Decoder para trainamento, atualização dos devidos parâmetros

Explicando o mecanismo de atenção em mais detalhes: Imagine que estamos em uma sala com todas as palavras da nossa frase. Cada palavra pergunta em voz alta — "Ei, quais de vocês tem a ver comigo?". Em seguida, as palavras relacionadas a ela respondem — "Eu!". A palavra que fez a pergunta olha para as que responderam e atualiza seu significado

com base nelas. Na prática, essas "perguntas", "respostas" e "atualizações de significado" são feitas por meio de operações entre vetores. No mecanismo de atenção, cada palavra da frase ganha mais três representações em vetores: o query, o key e o value. O query é o vetor que permite a cada palavra fazer a pergunta de quais outras estão relacionadas a ela. O key é o vetor que permite às palavras responderem se elas são relacionadas a alguma outra. O value é o vetor que será usado para atualizar os embeddings com os novos significados contextuais aprendidos. Quanto mais relacionadas duas palavras são, maior será o produto interno entre os vetores query e key dessas palavras, conforme mostra a Figura 7. Depois que fizermos o produto entre os vetores query e key de todas as palavras da frase, teremos uma matriz onde cada elemento indica o quão relacionadas duas palavras da frase estão, o chamado padrão de atenção.

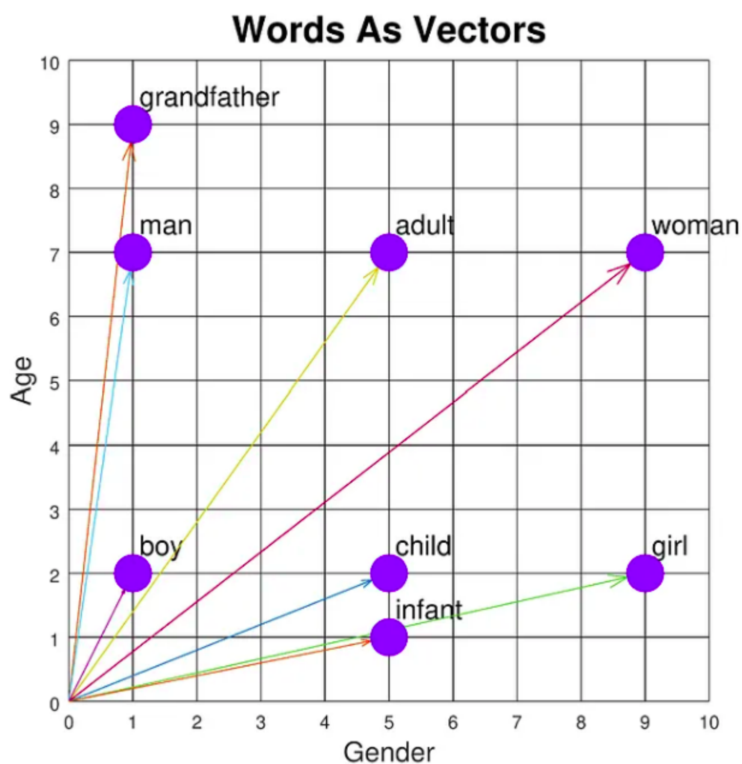

Figura 7: Representação das palavras como vetores query e key

A Figura 8 pode demonstrar o quão forte ou fraca é uma relação entre duas palavras conforme as posições dos tokens. Para gerenciar a ordem posicional, foi utilizado um Posi-

tionalEncoding para o modelo.

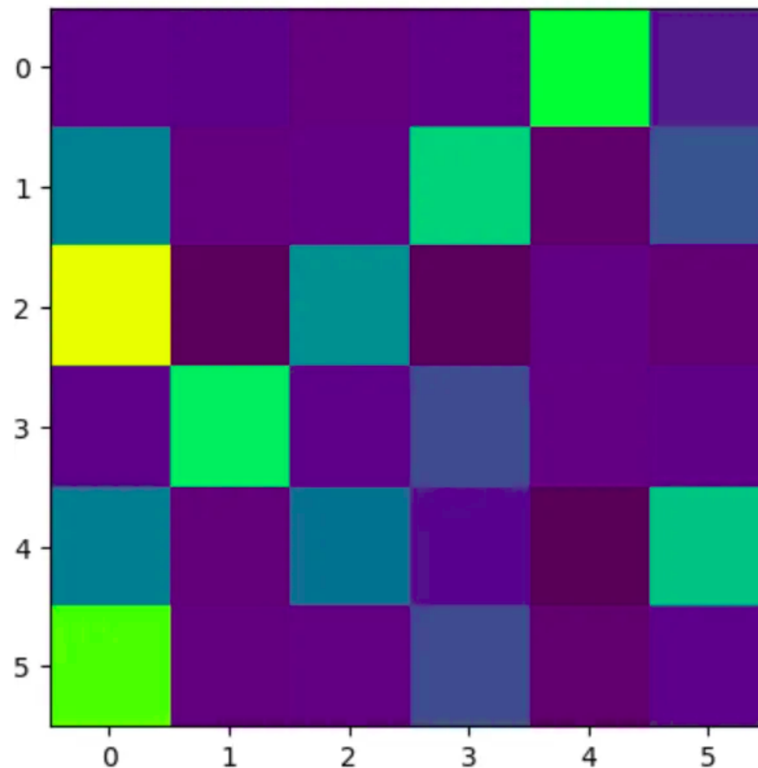

Figura 8: Nível de relação entre as palavras conforme as posições dos tokens

A implementação do Transformer envolveu a criação de um MultiHead-Attention, cálculo dos valores de Query, Key e Values e transformações lineares para retorno dos pesos de atenção, conforme exemplifica a Figura 9.

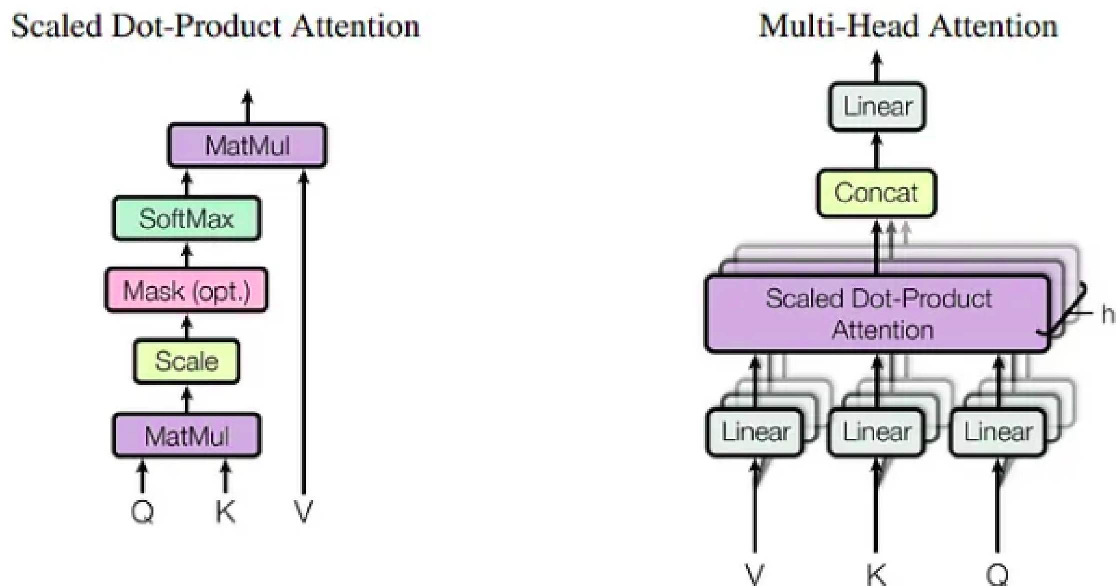

Figura 9: Operações com os valores de Query, Key e Values no MultiHead-Attention

Ao final da implementação, temos o processo de treinamento do modelo com um texto separado em tokens e métricas de avaliação para obtenção do valor referente a Função de Perda (Loss) em cada iteração do algoritmo.

3.2 Prova de Conceito 2 - Treinamento de um modelo Transformer com renderização do resultado e os pesos de atenção através do BertViz

Nesta segunda prova de conceito, foi utilizado um modelo de treinamento simples com um Transformer, uma única camada de MultiHead-Attention, 1 cabeça de atenção e embeddings com dimensão igual a 3, de forma que o processo de treinamento fosse mais simples e a interface para representação do resultado final não necessitasse de uma redução de dimensionalidade para correta visualização. Segue link da implementação abaixo:

`https://colab.research.google.com/drive/1g41JspPmxDyHCOBd_OJQYnPjuvl3ASGo?usp=sharing`

Temos o treinamento com esse Transformer usando duas sentenças de palavras que são tokenizadas e unificadas em uma única sentença. Em seguida, o pesos de atenção gerados são passados para renderização no BertViz, mais especificamente no método head-view que visualiza as relações entre os tokens e o padrão de atenção gerado conforme a camada de

attention e o de-para entre as sentenças. Segue abaixo a implementação usada para a função de treinamento, usando a classe para o Transformer simples descrito acima, uma função de otimização e um critério para perda:

```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

def train(model, input_ids, target, optimizer, criterion):
    model.train()
    optimizer.zero_grad()
    output, attention_weights = model(input_ids)
    output = output.transpose(0, 1)
    output = output.view(-1, vocab_size)
    target = target.view(-1)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()
    return loss.item(), attention_weights

loss, attention_weights =
train(model, input_ids, input_ids, optimizer, criterion)
```

Com base nos pesos de atenção gerados (attention-weights) temos a correta renderização desses pesos com o método head-view do BertViz, conforme implementação abaixo e a resultado exemplificado pela Figura 10:

```
def visualize_attention(attention_weights, tokens, sentence_b_start):
    head_view(attention_weights, tokens, sentence_b_start)

visualize_attention(attention_weights, tokens, sentence_b_start)
writer.close()
```

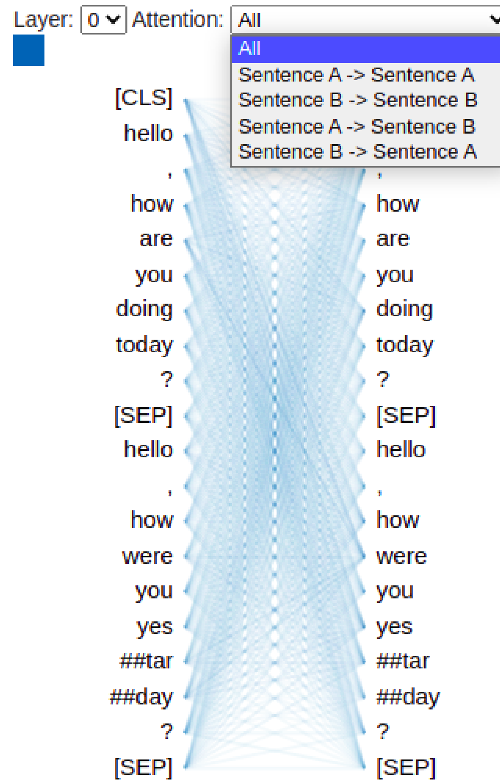

Figura 10: Resultado gerado pelo BertViz na Prova de conceito 2

3.3 Fluxograma da aplicação sugerida

O fluxograma da aplicação sugerido para esse projeto final de graduação foi separado conforme o tipo de modelo que o usuário poderá selecionar para exibição visual do resultado do treinamento e métricas de avaliação. Foi proposto 5 tipos diferentes de exibição:

- Rede Neural de Transformador
- Rede Neural Convolucional (CNN)
- Rede Neural Auto-Encoder
- Rede Neural Recorrente (RNN)
- Rede Neural Multi-Camada (MLP)

O princípio comum a todos para esse fluxograma é a possibilidade do usuário poder seleci-

onar os parâmetros para o processo de treinamento (exemplo: número de camadas na rede neural, taxa de aprendizado, número de iterações, filtro, pooling, etc) a partir de uma interface visual no front-end, com campos para preenchimento. Com base nesses parâmetros preenchidos, rodamos o modelo específico da opção selecionada a partir de algum Notebook (exemplo: Jupyter Notebook) e exibimos o output do processo de treinamento, com as particularidades do modelo e também o resultado das métricas de avaliação. Foram propostos 5 fluxogramas distintos para o funcionamento da aplicação, conforme o tipo do modelo: Transformador, CNN, Auto-Encoder, RNN e MLP, representados através das Figuras 11, 12, 13, 14 e 15 respectivamente.

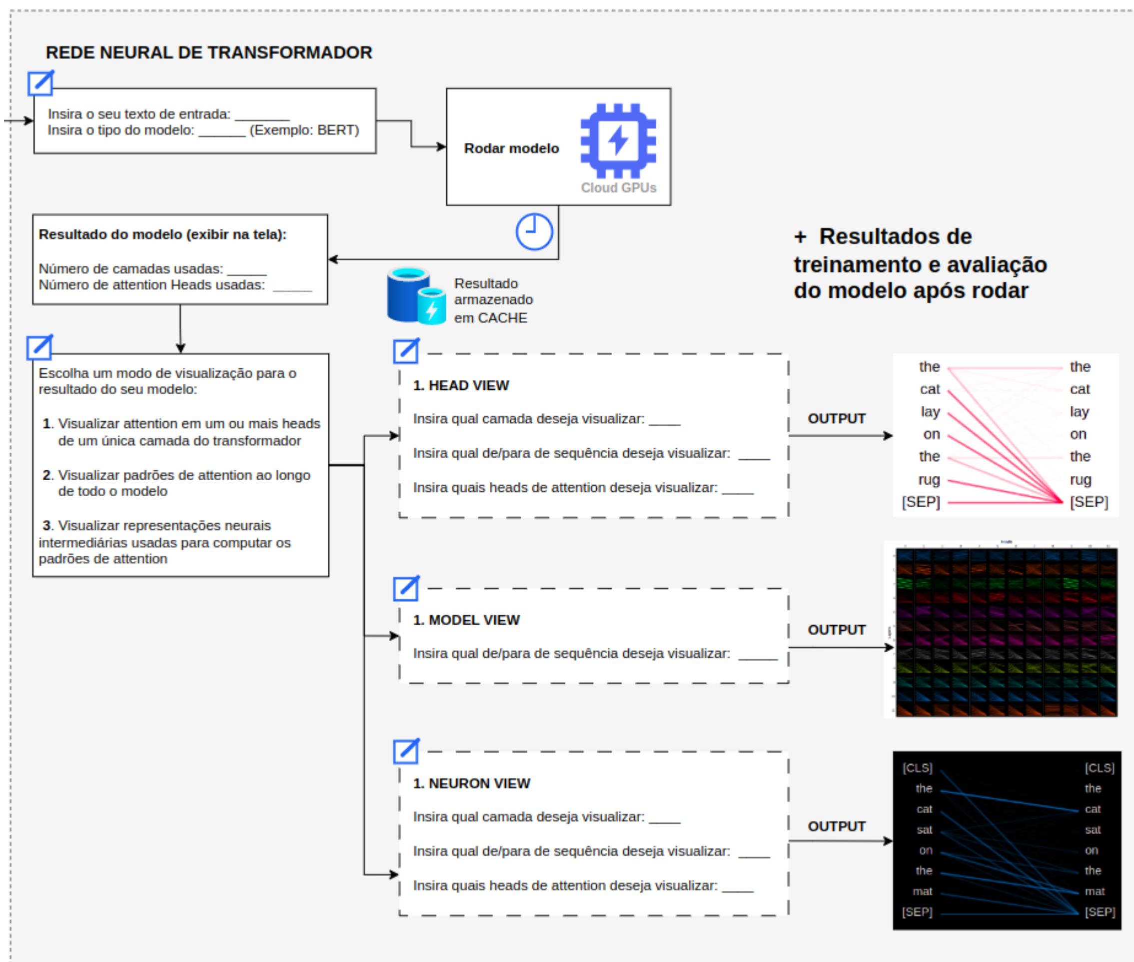

Figura 11: Fluxograma sugerido para rede neural de transformador

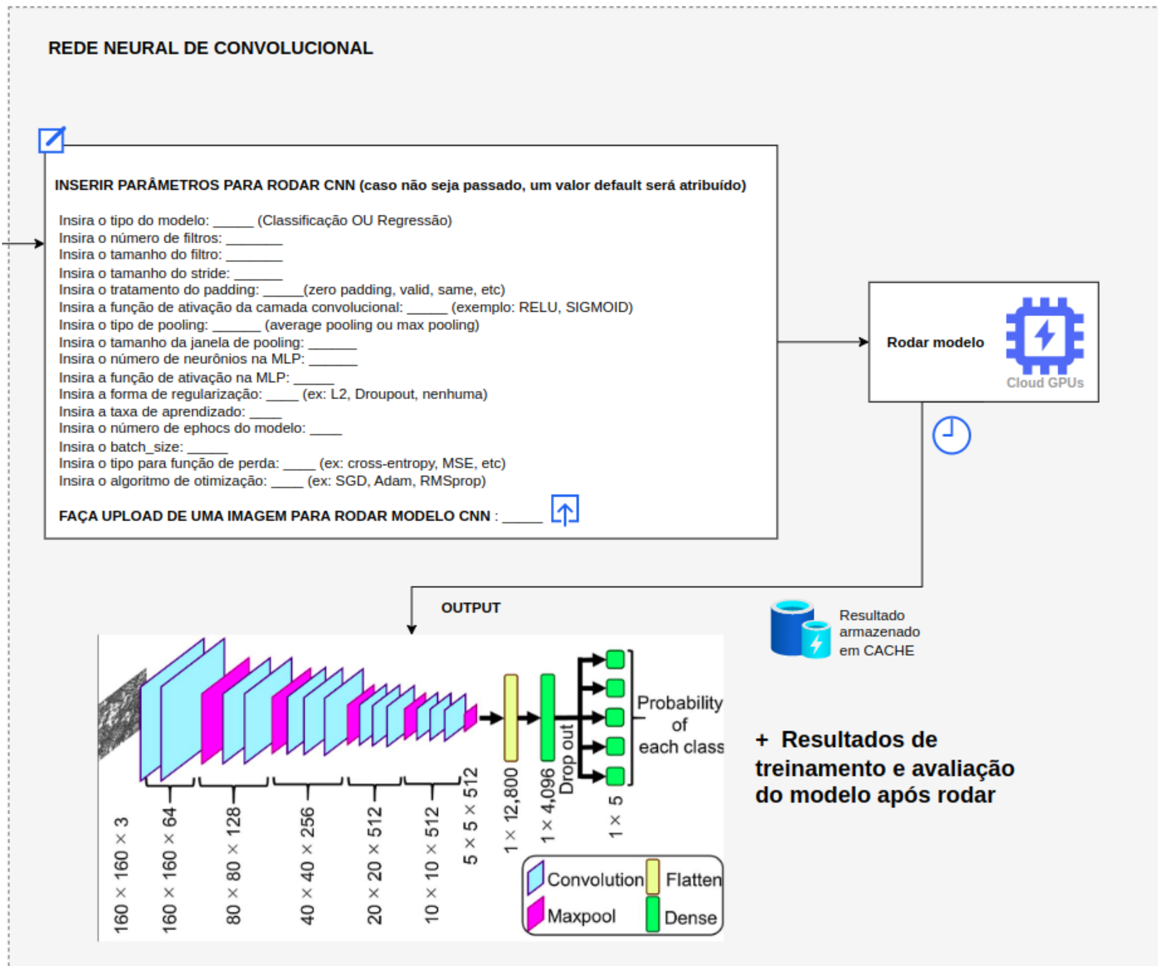

Figura 12: Fluxograma sugerido para rede neural convolucional

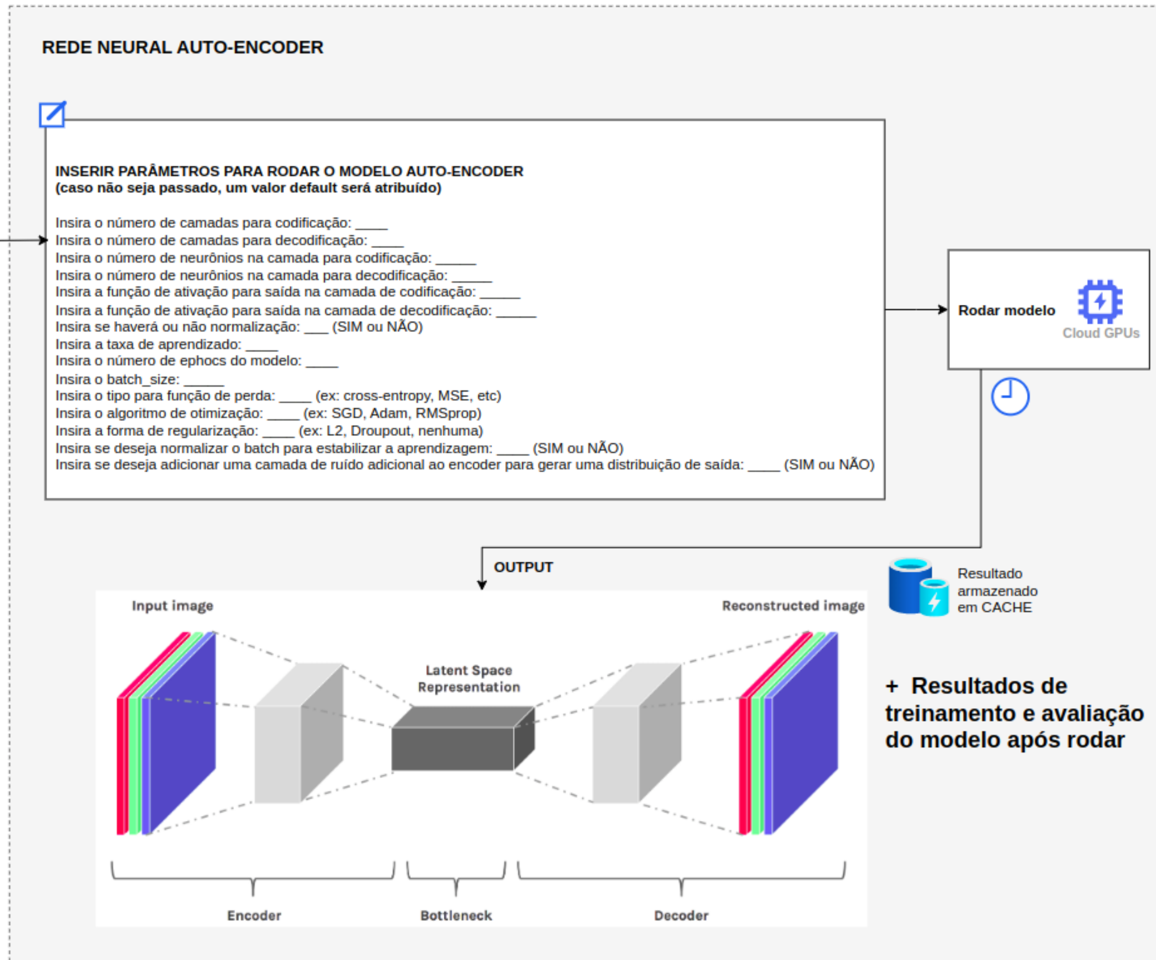

Figura 13: Fluxograma sugerido para rede neural auto encoder

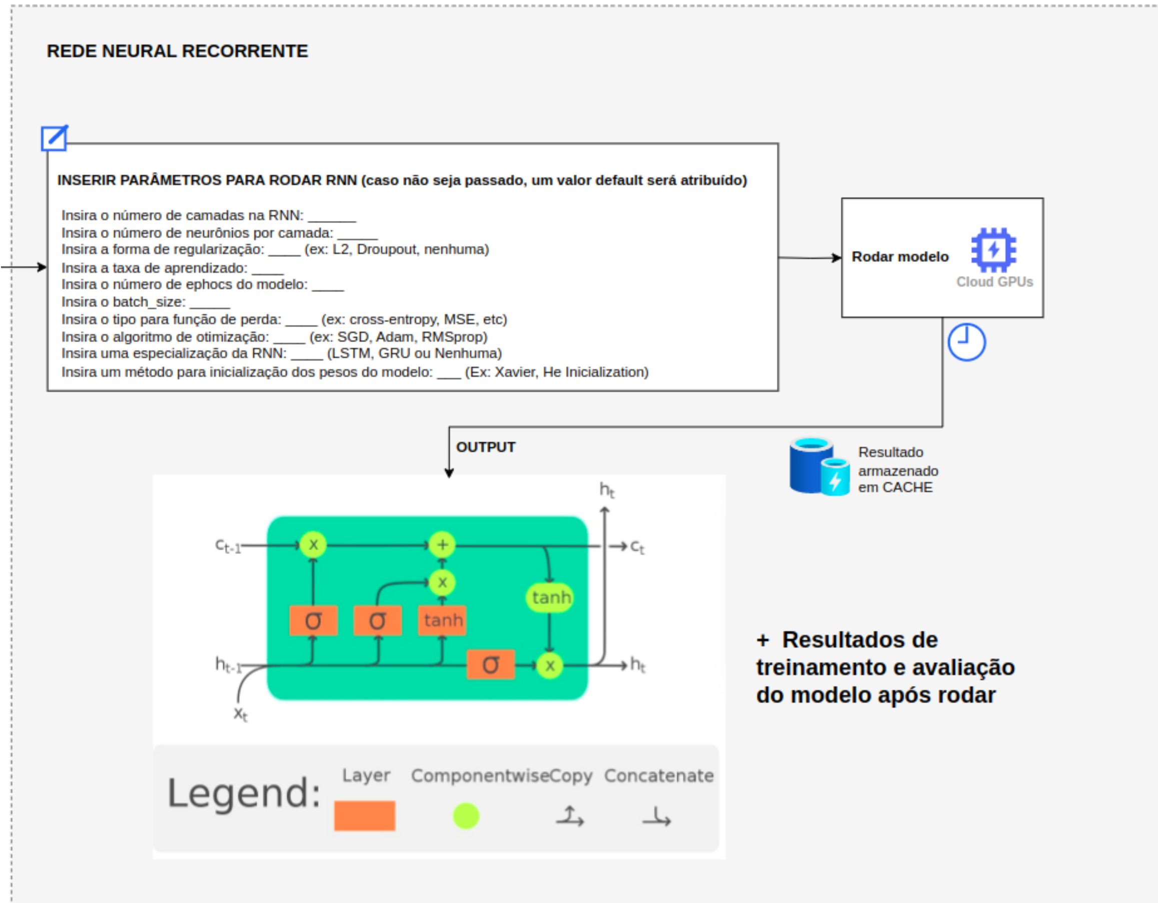

Figura 14: Fluxograma sugerido para rede neural recorrente

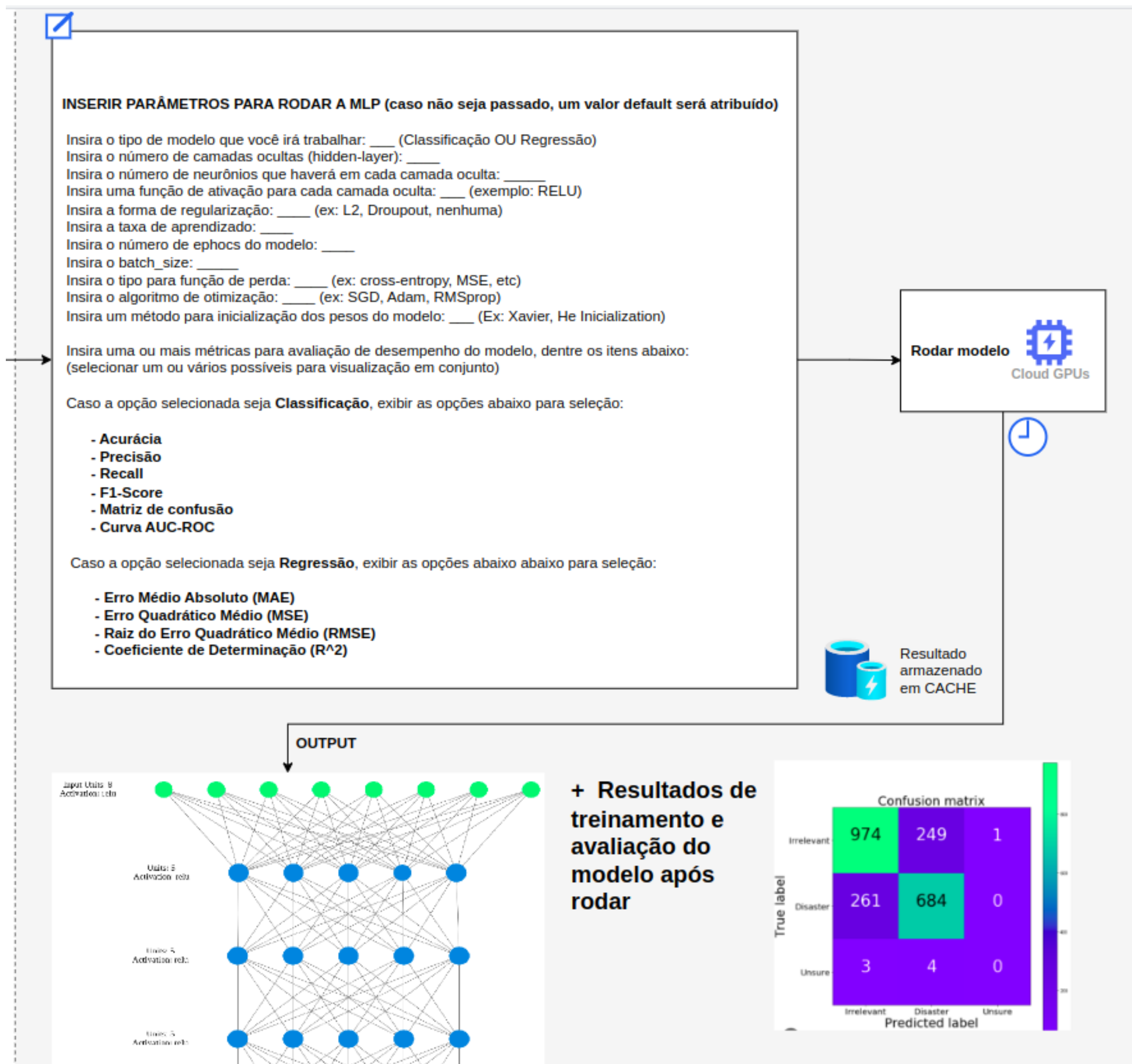

Figura 15: Fluxograma sugerido para rede neural MLP

3.4 Arquitetura sugerida/proposta para a aplicação

A arquitetura de software sugerida para implementação nesse projeto final de graduação foi separada em 4 grandes camadas, que desempenham funções independentes, conforme demonstrado na Figura 16:

- Camada de front-end: foi sugerido uma aplicação em REACT com HTML, CSS e Javascript, responsável por toda a parte de renderização e representação visual referente ao processo de treinamento e resultado do modelo. A ideia principal foi desenvolver uma aplicação WEB em React com campos para o preenchimento correto dos parâmetros de treinamento pelo usuário e blocos de exibição do resultado.

- Camada back-end: foi sugerido a implementação de um servidor python responsável por todo o meio de campo lógico para captura dos parâmetros de treinamento, rodar os modelos necessários através de uma comunicação com a camada de dados, estruturar os dados necessários, converter informação e obter os resultados de treinamento e representações visuais por intermédio das bibliotecas Python Keras/TensorBoard, BertViz (para representação de modelos com Transformadores)e PlotNeuralNet (para representação de redes neurais MLP). Foi proposta uma comunicação TCP entre o client front-end e o servidor Python através da implementação de um canal WebSocket para tráfego da informação em tempo-real e envio de mensagens entre o servidor o client. Ou seja, conseguimos notificar o backend para receber uma solicitação do client para rodar um processo de treinamento para um determinado tipo e modelo com os parâmetros preenchidos pelo usuário, o servidor processa a solicitação, roda o modelo e após finalização, uma mensagem pode ser enviada novamente ao client em tempo real com todos os resultados necessários do processo de treinamento, assim como métricas de avaliação ou conteúdos HTMLS renderizados com a representação visual das informações necessárias no modelo, sob intermédio das libs mencionadas (Keras/Tensorboard, BerViz e PlotNeuralNet). Também foi proposta uma comunicação assíncrona entre o servidor python e a camada de dados e camada de treinamento em nuvem para processo de treinamento do modelo (por questões de performance, estando preparado para grande volume de dados) e monitoramento da base de dados remota em tempo real para obter informações sobre a finalização do resultado de treinamento de algum modelo ou resultado parcial obtido periodicamente.

- Camada de treinamento: responsável pelo treinamento do modelo em nuvem a partir da comunicação iniciada pelo servidor Python e os parâmetros necessários para esse processo de treinamento. A tecnologia sugerida para isso foi o Google AI Platform

- Camada de dados: onde os resultados finais e intermediários do processo de treinamento do modelo são gravados a partir de tabelas escaláveis e preparadas para receber grande volume de dados.

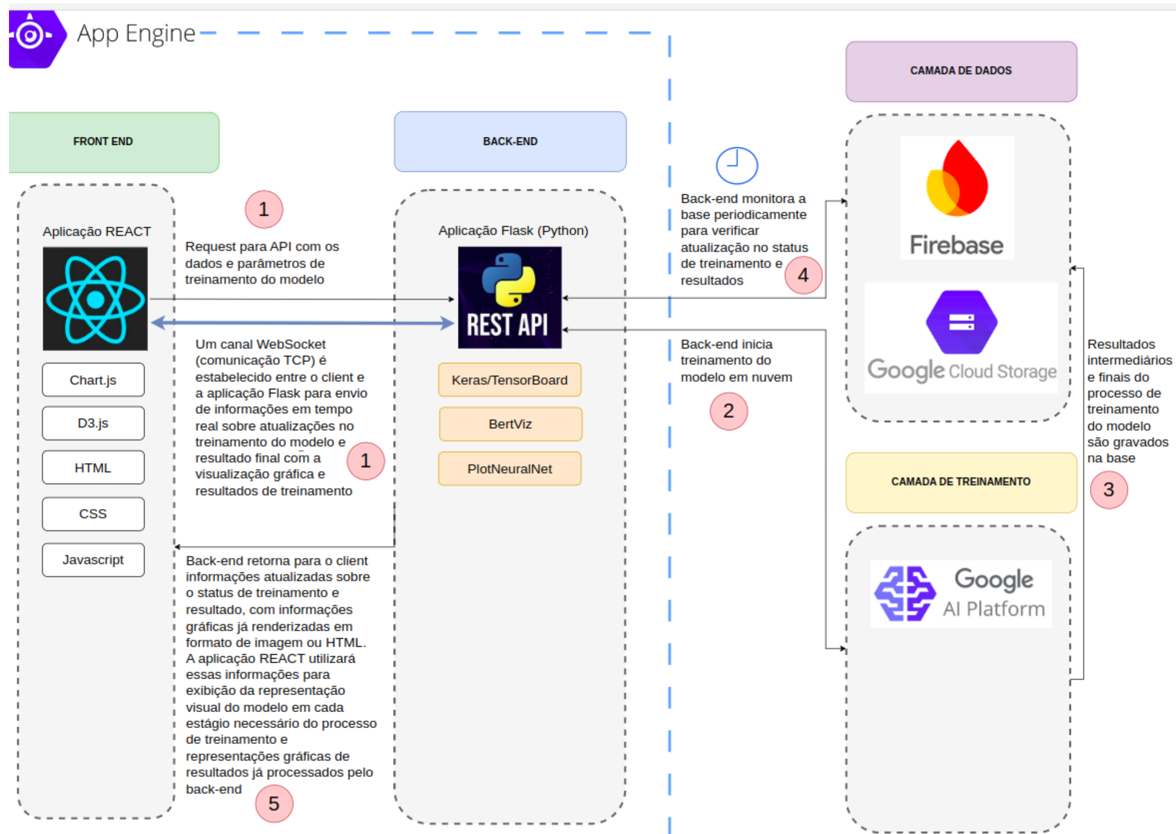

Figura 16: Arquitetura da aplicação sugerida, considerando 4 camadas

3.5 Arquitetura simplificada e levada adiante para implementação na aplicação

Para fins de simplificação, por questões de tempo de projeto e aderência com o real objetivo teórico e prático desse projeto final de graduação, arquitetura descrita e proposta acima foi reduzida apenas para um tipo de modelo: Tranformer com camadas de auto-atenção. A mudança foi direcionada para o Tranformer apenas devido ao potencial prático e didático da lib BertViz, que garantiu maior aproveitamento prático dentre todos os frameworks Pyhthon analisados na revisão da literatura, testes e provas de conceito, além de fornecer uma representação visual que faz mais sentido para o propósito deste projeto. Além disso, houve uma simplificação nas camadas da aplicação, com a exclusão da camada de dados e unificação da camada back-end com a camada de treinamento em um único escopo. Ou seja, visando otimizar tempo para o funcionamento core da aplicação e representação do modelo

e habilitando o objetivo principal do projeto final de graduação, foi implementada uma solução que nesse primeiro momento, está preparada apenas para ambiente local na própria maáquina e não se comunica com nenhum servidor remoto nem camada de treinamento externa.

Foi implementado um client front-end que roda em am ambiente local, um servidor python e uma comunicação com o Jupyter Notebook também em ambiente local para rodar o modelo e executar processo de treinamento. O servidor python se comunica com o Jupyter Notebook de forma programática com envio de mensagens através do uso da lib jupyter-client, que fornece uma interface programática com a execução de trechos de código nas células do Notebook, para treinar o modelo Transformer. O back-end python também recebe mensagens do notebook através do canal de comunicação criado sob intermédio do jupyter-client, com strings contendo o HTML renderizado pelas saídas de execução das células após rodar os métodos do BertViz para exibição do resultado com os pesos da atenção.

3.6 Estrutura da aplicação e fluxo de execução

O código da aplicação desenvolvida contém o client front-end em REACT e servidor back-end python em um único repositório, porém separados em módulos diferentes. O projeto se encontra versionado nesse repositório do gihub e contempla toda a implementação necessária:

`https://github.com/MatheusEsteves/deep_learning_simulator`

Estrutura para aplicação do client front-end React e fluxo de execução

Para o client React, foi criada uma página simples com dois containers de visualização. Um primeiro container para os campos com os parâmetros do modelo a serem preenchidos e dois campos para preenchimento do texto com as sentenças necessárias com palavras (que serão tokenizados). O segundo container contém um espaço para renderização do HTML com o resultado gráfico do BertViz. A aplicação front-end inicializa uma comunicação websocket com o servidor python (que precisa estar rodando também para o teste integrado de ponta a ponta) e a partir desse momento eles estão aptos para troca de mensagens em tempo real:

```
const socketUrl = "ws://localhost:8765";

const { sendMessage, _ } = useWebSocket(socketUrl, {
    onOpen: () => console.log("Conectado ao servidor WebSocket"),
    onMessage: (event) => {
```

```
        const response = JSON.parse(event.data);
        setLoading(false);
        setReceivedMessage(response)
    },
});
```

O client envia uma mensagem ao back-end Python contendo um payload com todas as informações necessárias nos parâmetros de treinamento do modelo de Transformer Simples com uma camada de MultiHead-Attention: dimensão do embedding, número de heads de atenção, duas sentenças para tokenização e modo de visualização do BertViz (que poderá variar entre head-view, model-view e neuron-view). Essas informações são enviadas em uma mensagem para que o servidor consiga processar e inicializar uma comunicação com o Jupyter Notebook e rodar o modelo em etapa de treinamento

```
const handleSendMessage = (viewMode) => {
    const message = {
        'event_action_type': 'run_transformer_with_attention',
        'event_payload': {
            'text_sentence_train': textSentenceTrain,
            'text_sentence_train_b': textSentenceTrainB,
            'visualization_mode': viewMode,
            'embed_size': embedSize,
            'num_heads': numHeads
        }
    }
    setLoading(true);
    sendMessage(JSON.stringify(message));
};
```

Em outro diretório nesse mesmo repositório, temos o servidor Python que receberá esse payload com parâmetros na mensagem e iniciará uma preparação para execução de código python nas células do Jupyter Notebook, através do canal de comunicação criado pelo jupyter-client. Essa preparação ocorre em 3 etapas:

- Comandos para os imports das libs (bertviz e matplotlib)
- Comandos para atribuição dos argumentos que serão injetados como atribuição de constantes na execução da célula no Notebook, com base nos parâmetros enviados pelo client
- Comando para rodar o treinamento e viualização do Transformer com o BertViz, com base em um arquivo Python centralizado apenas com a lógica de criação do

SimpleTranformer, processo de treinamento e plot das informações com o BertViz

```
def run_transformer_with_attention(payload):
 with open('notebook_cells_code/bertviz_model.py', 'r')
   as bertviz_model_code_file:
     try:
        arguments = {
           'TEXT_SENTENCE_TRAIN':
               f"'{payload['text_sentence_train']}'",
           'TEXT_SENTENCE_TRAIN_B':
               f"'{payload['text_sentence_train_b']}'",
           'VIEW_MODE':
               f"'{payload['visualization_mode']}'",
           'EMBED_SIZE':
               f"{payload['embed_size']}",
           'NUM_HEADS':
               f"{payload['num_heads']}"
        }
        imports = [
           '!pip install bertviz',
           '!pip install matplotlib'
        ]
        run_code_cell(
           bertviz_model_code_file.read(),
           arguments,
           imports
        )

        output_html_file_str = ''

        with open('outputs/cell_execution_output.html', 'r')
           as output_html_file:
              output_html_file_str = output_html_file.read()

        return {
         'event_response_type': 'run_transformer_with_attention',
         'event_response_payload': output_html_file_str
        }
    except Exception as e:
    ....
```

....
....

O processo de treinamento do modelo seguiu a mesma abordagem implementada na Prova de Conceito de uso do BertViz apresentado anteriormente nesse relatório, calculando explicitamente os valores de Query, Key, Values, com transformações lineares e extração dos pesos de atenção.

```
class SimpleTransformer(nn.Module):
    def __init__(self, vocab_size, embed_size, num_heads):
        super(SimpleTransformer, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.attention = nn.MultiheadAttention(
            embed_size, num_heads)
        self.fc = nn.Linear(embed_size, vocab_size)

        self.d_k = int(embed_size/num_heads)
        self.n_heads = num_heads
        self.embed_size = embed_size

        self.Wq = nn.Linear(embed_size, embed_size)
        self.Wk = nn.Linear(embed_size, embed_size)
        self.Wv = nn.Linear(embed_size, embed_size)

    def forward(self, x):
        x = self.embedding(x)
        x = x.transpose(0, 1)

        batch_size, seq_len, d_model = x.shape

        Q = self.Wq(x)
        K = self.Wk(x)
        V = self.Wv(x)

        output, attention_weights = self.attention(Q, K, V)
        attention_weights = torch.tensor(
          [[attention_weights.tolist()]]
        )
        attention_output = output
```

```
        output = self.fc(attention_output)
        return output, attention_weights

def train(model, input_ids, target, optimizer, criterion,
      vocab_size):
    model.train()
    optimizer.zero_grad()
    output, attention_weights = model(input_ids)
    output = output.transpose(0, 1)
    output = output.view(-1, vocab_size)
    target = target.view(-1)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()
    return loss.item(), attention_weights
```

4 Resultados

Como resultado desse projeto final de graduação, temos a implementação de uma aplicação Web responsável por representar visualmente as camadas e pesos de atenção de um modelo de Transformer, parametrizável conforme um determinado conjunto de parâmetros (número de heads de atenção, dimensão do embedding, textos para atenção e modo de visualização do BertViz) que são fornecidos pelo usuário. Para que o fluxo esteja funcionando corretamente, precisamos rodar o servidor Python para inicialização de um serviço websocket:

```
PFG/deep_learning_simulator/simulator_service$ python server.py
```

Como resultado esperado, conforme demonstra a página Web da Figura 17, temos a visualização do servidor websocket sendo inicializado na porta 8765 em ambiente local (ws://localhost:8765). Em seguida, podemos rodar a aplicação web front-end a partir do seguinte comando:

```
/PFG/deep_learning_simulator/simulator_portal$ npm run start
```

Como resultado esperado, o client React é inicializado na porta 3000 em localhost e a página web mencionada anteriormente será renderizada com a visão inicial da aplicação e os campos para preenchimento das informações necessárias pelo usuário (parâmetros do modelo e sentenças)

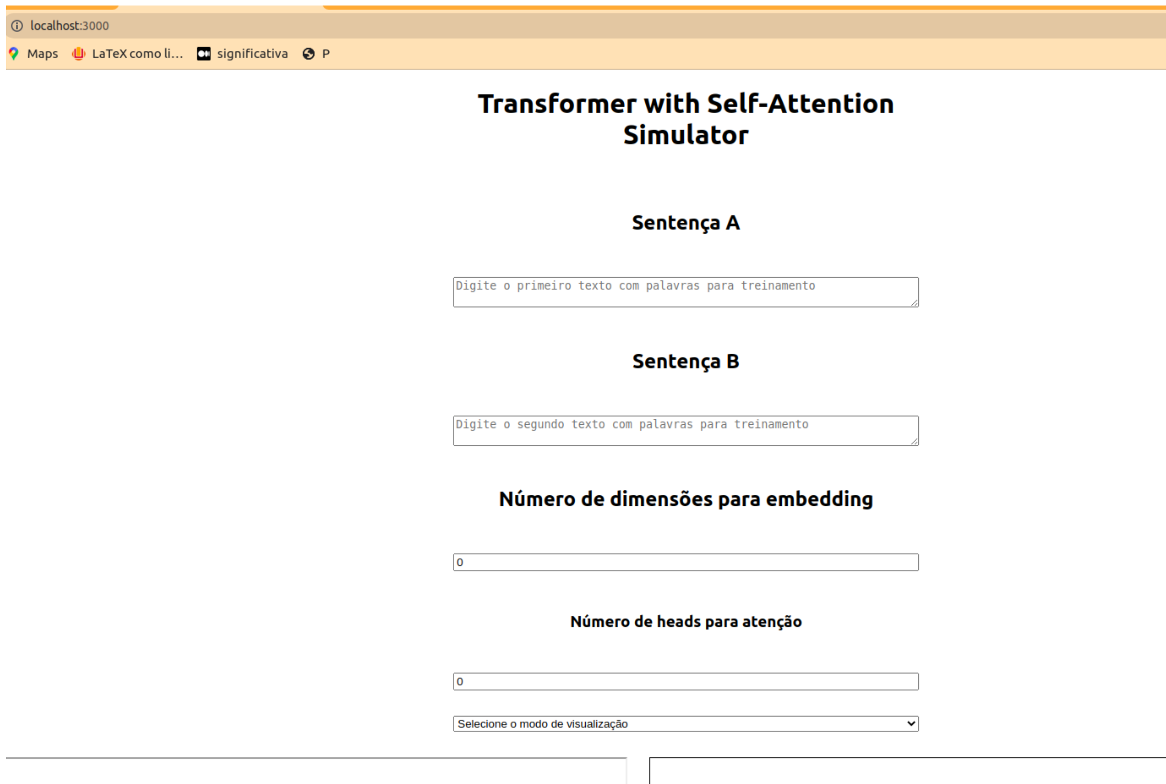

Figura 17: Página web principal da aplicação desenvolvida

Após preenchimento dos campos necessários para rodar o modelo e o modo de visualização do BertViz, a mensagem é enviada ao servidor Python via canal websocket e um gif com loading é exibido para aguardar o témino do processamento e treinamento do modelo no servidor, conforme mostra a Figura 18 (momento no qual esse servidor retornará com outra mensagem para o client em tempo real, sinalizando o resultado do treinamento com o HTML renderizado pelo BertViz para a represetaçaõ gráfica dos pesos de atenção)

Sentença A

O gato foi para casa da minha tia e nunca mais voltou

Sentença B

A minha tia foi na padaria com o gato e depois voltou

Número de dimensões para embedding

3

Número de heads para atenção

1

Head View

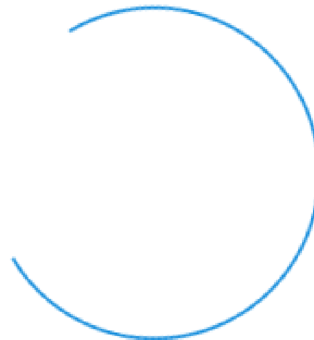

Figura 18: Preenchimento dos parâmetros de treinamento e carregamento das operações de treinamento para o modelo

Foram preenchidas duas sentenças de palavras (A e B), e parâmetros mais simples para o Transformmer (com 3 dimensões para o embedding e apenas 1 head de atenção). Porém, a arquitetura permite a passagem de dimensões maiores e mais heads de atenção para exibição no BertViz apos paassar pelo Transformer no servidor. Após selecionar o modo de visualização Head View, enviamos a mensagem ao back-end Python, a comunicação com o Jupyter Notebook é feita e em seguida as células são executadas com o processo de treinamento do transformer, passando esses parâmetros. Podemos visualizar nos logs do servidor o resultado de execução das células sendo sinalizado pelo Notebook em cada estágio de execução e como o conteúdo HTML é retornado pelo output de execução do BertViz na célula, conforme demonstra a Figura 19:

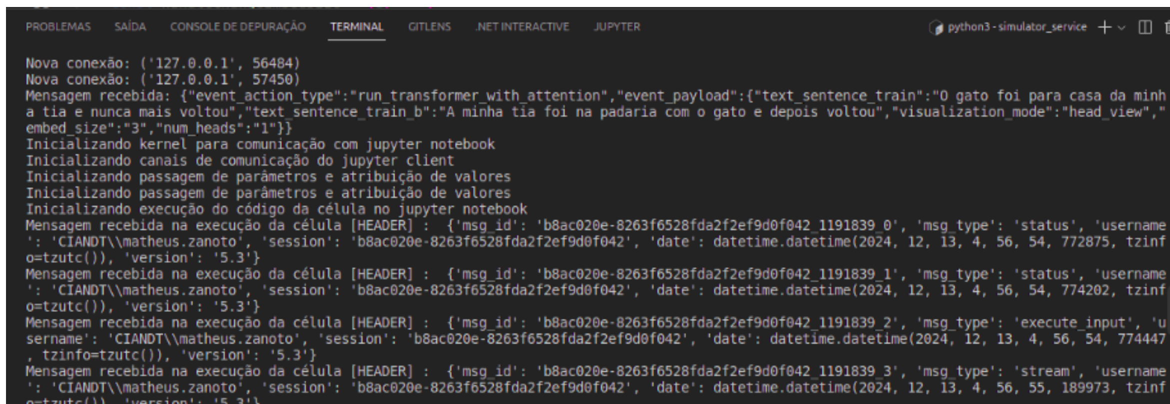

Figura 19: Logs de execução das células para o treinamento no Jupyter Notebook

Nos logs do servidor Python podemos observar as mensagens que chegaram para o Jupyter Client e a execução das células de forma programática para cada bloco de comando programado. Após a finalização dos blocos e o output final da saída no BertViz, o resultado exibido na Figura 20 é capturado e uma mensagem enviada novamente ao client para sinalizar uma renderização na página web com a imagem produzida pelo BertViz:

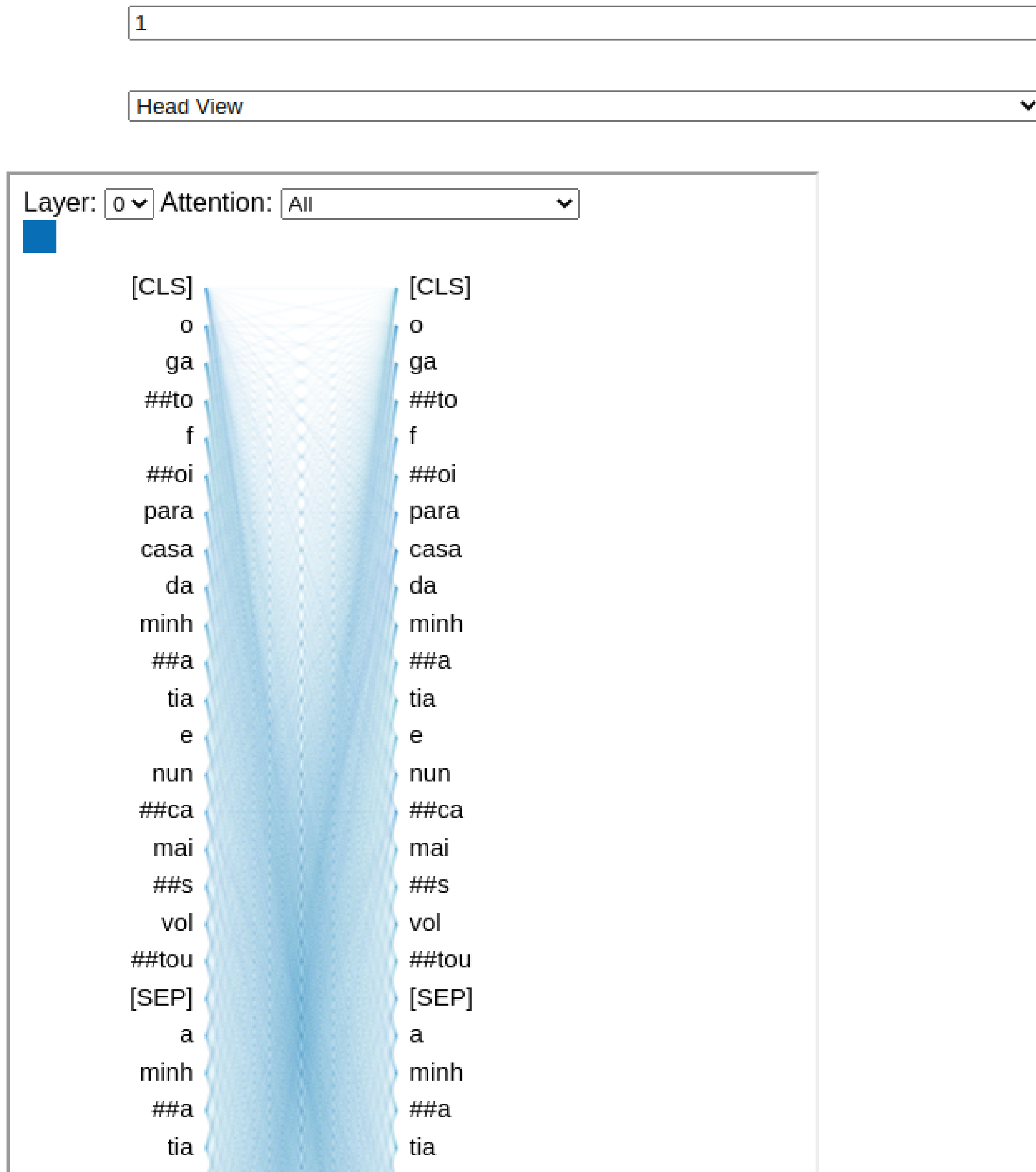

Figura 20: Resultado do BertViz após treinamento do modelo

Também podemos selecionar quais sentenças usaremos como base comparativa para visualizar a atenção, conforme demonstra a Figura 21:

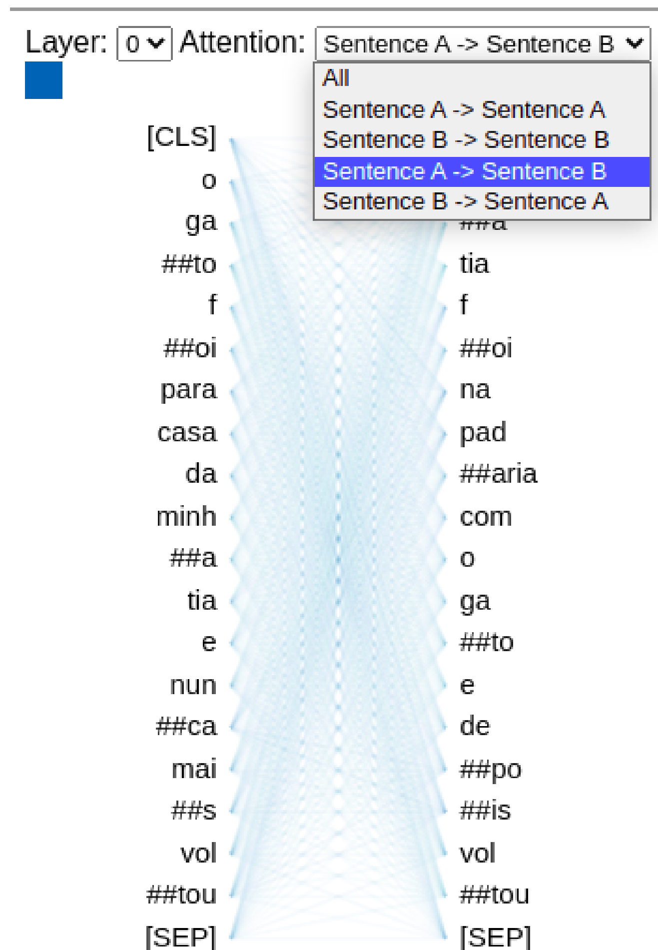

Figura 21: Resultado do BertViz com possibilidade para seleção das sentenças

5 Conclusões

O desenvolvimento desse projeto final de graduação foi de extrema importância para a consolidação prática e revisão de todos os conceitos de Deep Learning absorvidos na disciplina de Aprendizado de Máquina, como Transformadores e Mecanismo de Auto Atenção. A arquitetura inicial proposta precisou ser revista devido a limitações de tempo e escopo

para desenvolvimento do projeto, tornando-se em uma abordagem mais especializada para treinamento de Transformers com modelo BERT apenas. Porém, dada a complexidade dos modelos e o processo de comunicação entre a execução das células no Notebook e o resultado final na pagína Web, obteve-se um bom resultado no objetivo final do Projeto, que era justamente a representação visual para ensinos de modelo de aprendizagem profunda, independentemente de qual modelo ou algoritmo usado para o processo de treinamento

Como ponto de evolução desse projeto para uma possível Iniciação Científica, gostaríamos de uma abordagem visual mais estruturada para o usuário final na aplicação Web, com uma possibilidade maior e mais detalhada de parâmetros, representações visuais intermediárias e em tempo real ao longo das iterações no treinamento, e expandir/generalizar o mecanismo para outros modelos como CNN, RNN e MLP.

Referências

[1] Rulei Yu, Lei Shi. *A user-based taxonomy for deep learning visualization*. Journal Visual Informatics, Volume 2, Issue 3, Setembro/2018, Páginas 147-154.

[2] Xudong Liu. *The Educational Resource Management Based on Image Data Visualization and Deep Learning*. Journal Heliyon, Volume 10, Issue 13, Julho/2018, e32972.

[3] Yu Liang, Siguang Li, Chungang Yan, Maozhen Li, Changjun Jiang. *Explaining the Black-Box Model: A Survey of Local Interpretation Methods for Deep Neural Networks*. Journal Neurocomputing, Volume 419, 2/Janeiro/2021, Páginas 168-182.

[4] Sebastian Lapuschkin, Alexander Binder, Grégoire Montavon, Klaus-Robert Müller, Wojciech Samek. *The LRP Toolbox for Artificial Neural Networks*. Journal of Machine Learning Research, Volume 17, Número 114, 2016, Páginas 1-5.

[5] Zhou Yang, Ninghao Liu, Xia Ben Hu, Fang Jin. *Tutorial on Deep Learning Interpretation: A Data Perspective*. CIKM '22: Proceedings of the 31st ACM International Conference on Information and Knowledge Management, 17/Outubro/2022, Páginas 5156-5159.

[6] Jesse Vig. *BERTViz: A Tool for Visualizing Multi-Head Self-Attention in the BERT Model*. ICLR 2019 Debugging Machine Learning Models Workshop, Maio/2019.

[7] J Alammar. *Ecco: An Open Source Library for the Explainability of Transformer Language Models*. Association for Computational Linguistics, Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations, Agosto/2021, Páginas 249-257.

[8] Philipp Holl, Nils Thuerey. *ϕFlow: Differentiable Simulations for PyTorch, TensorFlow and Jax.* Proceedings of Machine Learning Research, Proceedings of the 41st International Conference on Machine Learning, Volume 235, 21-27/Julho/2024, Páginas 18515-18546.

[9] Kun-Chih (Jimmy) Chen, Masoumeh Ebrahimi, Ting-Yi Wang, Yuch-Chi Yang, Yuan-Hao Liao. *A NoC-based Simulator for Design and Evaluation of Deep Neural Networks.* Journal Microprocessors and Microsystems, Volume 77, Setembro/2020, 103145.

[10] Wejchert, Jakub, Tesauro, Gerald. *Neural Network Visualization.* Advances in Neural Information Processing Systems, Volume 2, 1989.

[11] Roland Kiraly, Sandor Kiraly, Martin Palotai. *Investigating the Usability of a New Framework for Creating, Working, and Teaching Artificial Neural Networks Using Augmented Reality (AR) and Virtual Reality (VR) Tools.* Journal Education and Information Technologies, Volume 29, 2024, Páginas 13085–13104.