Revista Brasileira de Informática na Educação – RBIE
Brazilian Journal of Computers in Education
(ISSN online: 2317-6121; print: 1414-5685)
https://sol.sbc.org.br/journals/index.php/rbie

Submission: 19/Sep/2023; 1st round notif.: 01/Jul/2024;
Camera ready: 20/Jul/2024; Edition review: 28/Nov/2024; Available online: 28/Nov/2024; Published: 28/Nov/2024;

Deep Learning para autoria automatizada de modelos de domínios de sistemas tutores baseados em passos

Title: Deep Learning for Automated Authoring of Domain Models for Step-Based Tutoring Systems

Título: Deep Learning para la Creación Automatizada de Modelos de Dominio en Sistemas de Tutorización Basados en Pasos

Fábio Manique de Castilhos
Universidade Federal do Paraná - UFPR
ORCID: 0000-0002-7580-4420
fabiocastilhoss@gmail.com

Patrícia A. Jaques
Universidade Federal do Paraná - UFPR |
Universidade Federal de Pelotas - UFPEL
ORCID: 0000-0002-2933-1052
patricia.jaques@gmail.com

Resumo

Os Sistemas Tutores Inteligentes (STIs) são programas de computador que auxiliam os estudantes durante o processo de aprendizado, através de assistência individualizada às características dos alunos. Eles possuem um módulo de domínio que representa o conhecimento especialista, em uma determinada área de estudo. Nos STIs baseados em passos, o módulo de domínio geralmente é desenvolvido como um sistema especialista baseado em regras, ou seja, um sistema que utiliza regras baseadas no conhecimento para resolver e corrigir os problemas propostos aos alunos. Esses sistemas especialistas baseados em regras acarretam uma maior complexidade na sua criação, manutenção e atualização do conhecimento, podendo gerar erros ou aumentar o tempo de processamento. Desta forma, este trabalho automatiza o módulo de domínio de um sistema tutor inteligente baseado em passos, através da criação de um modelo, que utiliza Deep Learning para realizar a correção de equações de primeiro grau. A correção é realizada sem a utilização de nenhum conhecimento matemático, apenas utilizando o Processamento de Linguagem Natural aplicado a uma base de cerca de 115 mil expressões matemáticas. São apresentadas e comparadas seis diferentes versões utilizando arquiteturas de redes neurais GRU e transformers. O modelo final atinge 95,5% *de acurácia na avaliação/correção dessas equações.*

Palavras-chave: *Sistemas Tutores Inteligentes; PAT2Math; Aprendizagem Profunda; Transformadores.*

Abstract

Intelligent Tutoring Systems (ITS) are computer programs that assist students during the learning process by providing personalized assistance based on the students' characteristics. They have a domain module that represents expert knowledge in a specific area of study. In step-based ITS, the domain module is typically developed as a rule-based expert system, meaning a system that uses knowledge-based rules to solve and correct problems presented to the students. These rule-based expert systems entail greater complexity in their creation, maintenance, and knowledge updates, potentially leading to errors or increased processing time. Therefore, this work automates the domain module of a step-based intelligent tutoring system by creating a model that employs Deep Learning to correct first-degree equations. The correction is performed without the use of any mathematical knowledge, solely relying on Natural Language Processing applied to a database of approximately 115,000 mathematical expressions. Six different versions using GRU and transformer architectures are presented and compared. The final model achieves a 95.5% *accuracy rate in evaluating/correcting these equations.*

Keywords: *Intelligent Tutoring Systems; PAT2Math; Deep Learning; Transformers.*

Cite as: Castilhos, F. M. & Jaques, P. A. (2024). Deep Learning para autoria automatizada de modelos de domínios de sistemas tutores baseados em passos, 32, 871-894. https://doi.org/10.5753/rbie.2024.3702.

Resumen
Los Sistemas de Tutoría Inteligente (STI) son programas de computadora que ayudan a los estudiantes durante el proceso de aprendizaje, brindando asistencia individualizada según las características de los alumnos. Estos sistemas tienen un módulo de dominio que representa el conocimiento experto en un área específica de estudio. En los STI basados en pasos, el módulo de dominio generalmente se desarrolla como un sistema experto basado en reglas, es decir, un sistema que utiliza reglas basadas en el conocimiento para resolver y corregir los problemas planteados a los estudiantes. Estos sistemas expertos basados en reglas pueden ser más complejos de crear, mantener y actualizar, lo que puede dar lugar a errores o aumentar el tiempo de procesamiento. Por lo tanto, este trabajo automatiza el módulo de dominio de un sistema de tutoría inteligente basado en pasos mediante la creación de un modelo que utiliza Deep Learning para corregir ecuaciones de primer grado. La corrección se realiza sin utilizar ningún conocimiento matemático, solo utilizando el Procesamiento del Lenguaje Natural aplicado a una base de datos de aproximadamente 115,000 expresiones matemáticas. Se presentan y comparan seis versiones diferentes utilizando arquitecturas de redes neuronales GRU y transformers. El modelo final alcanza una precisión del 95,5 % *en la evaluación y corrección de estas ecuaciones.*
Palabras clave: *Sistemas de Tutoría Inteligente; PAT2Math; Aprendizaje Profundo; Transformers.*

1 Introdução

Os Sistemas Tutores Inteligentes (STIs) são programas de computador que auxiliam os estudantes durante o processo de aprendizado, através de assistência individualizada às características dos alunos (Woolf, 2007). Os STIs baseados em passos (*Step-Based*) são sistemas tutores inteligentes que fornecem auxílio em cada um dos passos do problema que está sendo resolvido, através de *feedbacks* e dicas. Conforme VanLehn (2011), os STIs baseados em passos, aprimoram o desempenho obtido pelos estudantes, em níveis aproximados aos obtidos através de tutoria humana.

Os sistemas tutores são formados, geralmente, por quatro módulos: o módulo de domínio, que representa o conhecimento especialista, em uma determinada área de estudo; o módulo do aluno, que representa o aprendizado dos alunos sobre o domínio; o módulo de tutoria, responsável pelas estratégias de ensino; e o módulo de comunicação, representando os métodos de comunicação entre os alunos e o sistema. Nos STIs baseados em passos, o módulo ou modelo de domínio geralmente é desenvolvido como um sistema especialista baseado em regras (*Rule-Based Expert System*), ou seja, um sistema que utiliza regras baseadas no conhecimento para resolver e corrigir os problemas propostos aos alunos (Woolf, 2007).

O PAT2Math (*Personal Affective Tutor to Math*) é um STI baseado em passos que busca ajudar os estudantes na resolução de equações de primeiro grau. O *PAT2Math* possui um sistema especialista, baseado em regras, cujo objetivo é a resolução das equações e avaliação/correção das respostas submetidas pelos alunos, fornecendo o adequado *feedback* (Jaques et al., 2013).

Os sistemas especialistas baseados em regras são eficientes quando as regras podem ser derivadas e codificadas, nas situações específicas para os quais foram projetados. Entretanto, devido à grande quantidade de regras necessárias, esses sistemas acarretam uma maior complexidade na sua criação, manutenção e atualização do conhecimento, podendo gerar erros ou aumentar o tempo de processamento (Nagori & Trivedi, 2012).

Desse modo, a automatização do módulo de domínio é uma forma de solucionar esses problemas e aprimorar o desenvolvimento dos sistemas tutores. Uma solução eficiente para a au-

tomatização desse processo é a utilização de *Deep Learning*, uma subárea do aprendizado de máquina que utiliza redes neurais profundas para modelar e resolver problemas complexos, que permite uma implementação mais rápida e simples, solucionando a complexidade de atualização e permitindo um menor tempo de execução.

Quanto à resolução de problemas matemáticos e simbólicos, a utilização de *Deep Learning*, tem sido foco de artigos, que publicaram importantes avanços na área. O trabalho de Wangperawong (2019), alcançou $84,90\%$ de acerto na resolução de frases matemáticas, com escopo restrito. Saxton et al. (2019) e Schlag et al. (2019) atingiram 76 e 82% de acurácia, respectivamente, na resolução de problemas matemáticos. Hendrycks et al. (2021), em um conjunto de dados bem mais complexo, chegou a $6,9\%$ de acerto. Já Lample e Charton (2019) e Charton et al. (2020) conseguiram atingir uma performance de $99,7$ e 95% de acurácia, em questões com maior nível de complexidade, resultados que podem ser comparados a *frameworks* matemáticos atuais.

Neste contexto, diferente dos trabalhos citados, o objetivo deste trabalho é a criação de um *step analyser*, um componente que analisa cada passo dado pelo estudante ao resolver um problema, verificando a correção de cada etapa (VanLehn, 2006). O modelo recebe como entrada uma equação de primeiro grau e o passo seguinte, conforme digitado pelos estudantes, seja esse um passo intermediário ou a solução final da equação, e verifica se o passo é uma resposta correta ou incorreta para a equação. Dessa forma, objetiva-se a automatização do módulo de domínio de um sistema tutor inteligente baseado em passos, através de um modelo que utiliza *Deep Learning* para realizar a correção de equações de primeiro grau.

Para geração do modelo, foram criadas seis diferentes versões, de duas arquiteturas distintas: redes neurais GRU e *transformers*, treinados em uma base de 115 mil expressões matemáticas, originalmente extraídas do *log* de interações dos estudantes com o tutor inteligente *PAT2Math*. A versão que atingiu os melhores resultados possui a arquitetura de *transformers* e chegou a $95,5\%$ de acurácia na correção dos passos digitados pelos estudantes.

2 Fundamentação Teórica

Os dados das interações dos estudantes junto ao sistema tutor *PAT2Math*, são utilizados neste trabalho para propiciar a automatização do módulo de domínio e criação de um *step analyzer*, para correção de equações de primeiro grau. Essa tarefa é realizada sem conhecimento matemático prévio, utilizando o processamento de linguagem natural, através de técnicas de *deep learning*. Tais conceitos são citados nas subseções seguintes.

2.1 Deep Learning

Deep Learning, ou aprendizado profundo, é um subcampo específico de *machine learning*, que da mesma forma que o aprendizado de máquina, busca aprender representações/padrões a partir de dados, sem ser explicitamente programada para isso (Trask, 2019). Enquanto nos programas clássicos são inseridos dados e regras, retornando respostas, aqui são inseridos dados e respostas (*labels*) e o programa retorna as regras ou modelo, que pode ser aplicado a novos dados (Chollet, 2021).

O seu nome refere-se a ideia de camadas sucessivas de redes neurais. Enquanto algumas abordagens de *machine learning* tendem a se concentrar na aprendizagem de apenas uma ou duas camadas de representações dos dados, sendo, por vezes, chamadas de aprendizado raso, as abordagens modernas de *deep learning* envolvem dezenas ou até centenas de camadas sucessivas de representações, aprendidas automaticamente a partir da exposição aos dados de treinamento (Chollet, 2021).

Essas representações em camadas são geralmente aprendidas por meio de redes neurais, estruturadas em camadas empilhadas umas sobre as outras. *Deep Learning* é usada para resolver tarefas práticas em uma variedade de campos, como visão computacional (imagens), processamento de linguagem natural (texto) e reconhecimento automático de fala (áudio) (Trask, 2019).

2.2 Redes Neurais

As redes neurais são sistemas compostos por nós conectados por ligações direcionadas, inspiradas no funcionamento dos neurônios do cérebro humano. Seus nós possuem pesos e funções de ativação, que são propagados pela rede através de suas ligações. Essas redes podem reconhecer padrões escondidos, fazer correlações, agrupamentos e classificação de dados, além de aprender e melhorar continuamente a cada período de tempo (Russell & Norvig, 2010).

Embora alguns dos conceitos centrais relacionados a redes neurais ou *deep learning* tenham sido inspirados pela compreensão do cérebro humano, tais modelos não são modelos do cérebro. Não há evidências de que o cérebro implemente algo semelhante aos mecanismos dos modelos modernos de redes neurais e *deep learning* (Chollet, 2021).

Existem tipos diferentes de redes neurais, cada um possuindo vantagens e desvantagens, dependendo do uso. Como exemplo, pode-se citar as Redes Neurais Convolucionais (RNCs), popularmente utilizadas para classificação de imagens e a detecção de objetos; e as Redes Neurais Recorrentes (RNRs), utilizadas na previsão e aplicação de séries temporais, análise de sentimentos e outras aplicações de processamento de linguagem natural.

As Redes Neurais Recorrentes (RNRs) são redes projetadas para utilização com informações sequenciais. Elas são capazes de memorizar parte das entradas e utilizá-las para realizar previsões mais precisas. Diferentemente das redes neurais convencionais, as redes recorrentes persistem as informações, passando a mensagem aos seus sucessores.

Dentre alguns modelos de redes recorrentes, pode-se citar a LSTM e a GRU. As LSTM (*Long Short-Term Memory*) são um tipo especial de redes neurais recorrentes, capazes de aprender as dependências de longo prazo e guardar informações por longos períodos. Elas foram introduzidas por Hochreiter e Schmidhuber (1997) e foram refinadas e popularizadas nos anos seguintes. As GRU (*Gated Recorrent Unit*), introduzidas por Chung et al. (2014), são redes similares às LSTM, mas possuem um único estado oculto, para manter as dependências de longo e curto prazo, ao mesmo tempo.

Uma evolução importante nas redes neurais recorrentes foi possível com a criação do mecanismo de atenção. A atenção foi proposta pela primeira vez por Bahdanau et al. (2014) para tradução automática. É particularmente útil para essa aplicação, pois as palavras mais relevantes para a saída geralmente ocorrem em posições semelhantes na sequência de entrada. Há três tipos de atenção possíveis em um modelo: atenção Codificador-Decodificador: atenção entre a sequên-

cia de entrada e a sequência de saída; *self-attention* na sequência de entrada: atende a todas as palavras na sequência de entrada; *self-attention* na sequência de saída (Luong et al., 2015).

Com a evolução dos mecanismos de atenção, surgiu uma nova arquitetura que revolucionou o processamento de tarefas sequenciais e trouxe melhorias significativas no tratamento de dependências de longo alcance: os *Transformers*. Diferente das redes recorrentes, os *Transformers* não dependem de uma estrutura sequencial para processar dados, o que os torna mais eficientes e escaláveis para diversas aplicações. Essa arquitetura será explorada em detalhes na Seção 2.3.

2.3 Tranformers

O *Transformer* é uma arquitetura que visa resolver tarefas sequenciais enquanto lida com dependências de longo alcance. Ele se baseia inteiramente em *Self-Attention* para transformar uma sequência em outra a partir de um codificador e decodificador, mas difere dos modelos *Sequence-to-Sequence* porque não implica na utilização de nenhuma rede recorrente (GRU, LSTM, etc.) (Vaswani et al., s.d.).

Como exemplos de aplicações, podemos citar o resumo automático de texto, preenchimento automático, resposta automática de perguntas e a tradução automática. Além disso, também são utilizados em *chatbots* e em outras tarefas, como Análise de Sentimentos, Inteligência de Mercado, Classificação de Texto e outras (DeepLearning.ai, 2020).

Sua arquitetura provém diversas vantagens, tais como: não faz suposições sobre as relações temporais/espaciais entre os dados, o que é ideal para o processamento de conjunto de objetos; possibilidade das saídas das camadas serem calculadas em paralelo, ao invés de calcular sequencialmente, como uma RNR, melhorando o tempo de execução; os itens distantes podem afetar cada um dos outros sem precisar passar por muitas etapas; o aprendizado de dependências de longo alcance, que é um desafio em muitas tarefas sequenciais (TensorFlow, 2020).

Tanto o codificador quanto o decodificador são compostos de módulos que podem ser empilhados um sobre o outro várias vezes. Os módulos consistem principalmente nas camadas *Multi-Head Attention* e *Feed Forward*. As entradas e saídas são primeiro incorporadas em um espaço n-dimensional, pois não se pode usar strings diretamente (Vaswani et al., s.d.).

O modelo possui uma codificação posicional das diferentes palavras (*Positional encoding*). Como não existem redes recorrentes, é necessário dar a cada palavra/parte da sequência uma posição relativa, pois uma sequência depende da ordem de seus elementos. A saída de codificação posicional coloca valores a serem adicionados aos *embeddings*, vetor onde cada palavra de entrada fornecida ao modelo, possui informações sobre sua ordem e posição (Vaswani et al., s.d.).

Dentre os modelos mais recentes de *transformers*, pode-se citar GPT-2 (*Generative Pre-training for Transformer*) (Radford et al., 2019) e GPT-3 (Brown et al., 2020), da *Open AI*, que atingiram resultados comparáveis a humanos na geração de texto. O BERT (*Bidirectional Encoder Representations from Transformers*) (Devlin et al., 2018), que foi criado pela equipe do *Google AI Language*, é outro *transformer* famoso, usado para aprender representações de texto. Por fim, o T5 (*Text-to-Text Transfer Transformer*)(Raffel et al., 2019), também criado pelo *Google*, é um *transformer* multitarefa, podendo realizar várias tarefas distintas como tradução, resposta automática e classificação, tudo através do mesmo modelo.

3 Trabalhos Relacionados

Esta seção descreve alguns dos trabalhos publicados, que utilizam *Deep Learning* para a resolução de problemas matemáticos e simbólicos.

O artigo de Wangperawong (2019) utiliza modelos de redes neurais para a leitura e avaliação de frases matemáticas ("*word problems*", em inglês). As frases contém expressões matemáticas envolvendo adição, subtração e multiplicação de números positivos e negativos. Os valores utilizados ficam restritos entre -1000 e 1000. Por exemplo, a frase $x = 85; y = -523; x * y$, tem como resultado -44455.

O trabalho usou cerca de 12 milhões de exemplos, gerados e divididos randomicamente entre conjuntos de treinamento e de teste. Cada entrada foi lida e codificada para o nível de caractere e cada saída foi decodificada do nível de caractere. O modelo chegou a uma acurácia de $76,1\%$ no conjunto de testes.

Além disso, também foi utilizado o modelo de *transformer* universal, proposto por Dehghani et al. (2018), chegando a uma precisão de $78,8\%$, no mesmo conjunto de teste. Por fim, o *transformer* universal adaptativo, atingiu quase a perfeição na avaliação de $a - b$, mas teve resultados piores na multiplicação, chegando a uma acurácia total de 84.9%.

Já num escopo mais amplo, o artigo de Saxton et al. (2019) apresenta conjuntos de problemas matemáticos envolvendo perguntas e respostas sequenciais em um formato de texto livre. Esse conjunto é avaliado através de redes neurais, visando resolver os problemas matemáticos e generalizar seus conhecimentos. Um exemplo de questão do conjunto, seria descrita como: $Calculate - 841880142.544 + 411127$ e a resposta, como: -841469015.544.

Saxton et al. (2019) utilizou as arquiteturas de LSTM simples, LSTM com mecanismo de atenção, *Relational Memory Core* (RMC) (Santoro et al., 2018) e *transformers* (Vaswani et al., s.d.). O *Transformer* teve desempenho significativamente melhor do que os modelos recorrentes, alcançando 76% nos testes de interpolação e 50% nos testes de extrapolação.

Da mesma forma, o artigo de Schlag et al. (2019) alcança melhores resultados na resolução de questões matemáticas em texto-livre, utilizando o mesmo conjunto de dados proposto por Saxton et al. (2019). Isso é feito através de uma modificação no mecanismo de atenção dos *transformers*, chamada de *Tensor Product Multiheaded Attention (TPMHA)*. Essa mudança origina um novo modelo de *transformers*, denominado *Tensor-Product Transformer (TP-Transformer)*. Assim, o modelo atingiu $81,92\%$ de acurácia nos testes de interpolação e $54,67\%$ de acurácia nos testes de extrapolação, superando seus resultados originais.

Num escopo mais bem complexo, o artigo de Hendrycks et al. (2021) apresenta o conjunto de dados matemáticos, chamado *MATH (Mathematics Aptitude Test of Heuristics)*. Esse conjunto possui 12.500 enunciados de problemas matemáticos (7.500 para treinamento e 5.000 para teste), categorizados em sete tipos: Álgebra, Cálculo, Estatística, Geometria, Álgebra Linear e Teoria dos números.

Além do conjunto de dados MATH, o trabalho também apresenta outro conjunto, para pré-treinamento, chamado *AMPS (Auxiliary Mathematics Problems and Solutions)*, utilizado para melhorar a acurácia final do modelo. O conjunto de dados contém 100 mil problemas da *Khan Academy* e mais cerca de 5 milhões de problemas gerados pelo *software Mathematica*.

Devido ao nível de dificuldade imposto pelo conjunto de dados *MATH*, o melhor resultado obtido pelo modelo *GPT-2* foi de $6,9\%$, em média. Entretanto, os modelos são capazes de gerar soluções passo a passo que são coerentes e adquirem algum conhecimento matemático, alcançando até 15% de precisão no nível de dificuldade mais fácil.

Para finalizar, pode-se citar dois artigos envolvendo os mesmo autores. O trabalho de Lample e Charton (2019), propõe a utilização de Redes Neurais em tarefas mais elaboradas em matemática, especificamente, na integração de funções e na resolução de equações diferenciais, de primeira e segunda ordem.

Para criar os dados de treinamento, foram gerados conjuntos de expressões matemáticas aleatórias, baseados em três diferentes abordagens: *Forward Generation (FWD)*, *Backward generation (BWD)* e *Integration by parts (IBP)*. O modelo utilizado no artigo é o modelo de *transformers*, descrito por Vaswani et al. (s.d.), com 8 cabeças de atenção, 6 camadas e uma dimensionalidade de 512, utilizando o otimizador Adam. As expressões foram geradas por *beam search*, com larguras de 1, 10 e 50.

Os resultados alcançados ficaram próximos de $100,0\%$ para integrais e chegaram a $94,0\%$ e $91,2\%$ para equações diferenciais de primeira e segunda ordem, respectivamente. Os autores também compararam os resultados do modelo ao obtido por softwares comerciais. Os resultados dessa comparação superaram os resultados de *frameworks* matemáticos em equações geradas pela abordagem BWD.

Por fim, o artigo de Charton et al. (2020) investigou o uso de modelos de *Deep Learning* para tarefas matemáticas complexas envolvendo cálculos simbólicos e numéricos. Os modelos criados previram as propriedades qualitativas e quantitativas de objetos matemáticos, sem possuir conhecimento matemático prévio. O trabalho considerou três problemas avançados de matemática: a estabilidade local e a controlabilidade de sistemas diferenciais, além da existência e comportamento no infinito de soluções de equações diferenciais parciais.

Em todos os experimentos, foi utilizada uma arquitetura de *transformers* com 8 cabeças de atenção, variando a dimensão de 64 a 1024 e o número de camadas de 1 a 8. O modelo final atingiu mais de 95% de precisão em todas as tarefas qualitativas e entre 65 e 85% em cálculos numéricos.

3.1 Considerações Finais

Diferente dos trabalhos citados, o objetivo deste trabalho é a criação de um *step analyser* e a automatização do módulo de domínio de um sistema tutor inteligente baseado em passos, através de um modelo, que utiliza *Deep Learning* para realizar a correção de equações de primeiro grau. Essa correção é feita sem a utilização de nenhum conhecimento matemático prévio, apenas utilizando o Processamento de Linguagem Natural aplicado a 115 mil expressões matemáticas. O modelo proposto busca atingir bons resultados de acurácia e evitar a dependência de ferramentas externas.

4 Método

O objetivo principal deste trabalho é automatizar o módulo de domínio de um sistema tutor inteligente baseado em passos, através da criação de um modelo, comumente chamado de *step analyser* (VanLehn, 2006), que utiliza *Deep Learning* para realizar a correção dos passos de resolução de equações de primeiro grau dos estudantes. Ao contrário do modelo convencional, criado a partir de regras inseridas em um sistema especialista, o modelo automatizado permite que as atualizações sejam feitas simplesmente através da inclusão de novos dados, reduzindo muito o tempo e a complexidade dessa tarefa.

O modelo proposto recebe como entrada uma equação de primeiro grau e o passo seguinte, conforme digitado pelos estudantes no STI, seja esse, um passo intermediário ou a solução final. Feito isso, verifica se o passo digitado é uma resposta correta ou incorreta para a equação. Os dados utilizados para o treinamento do modelo, foram obtidos diretamente do *log* do sistema tutor inteligente *PAT2Math*, contendo a equação inicial, o passo digitado e o rótulo que descreve o passo como correto ou incorreto. O modelo criado utiliza o processamento de linguagem natural para corrigir as equações, sem possuir nenhum conhecimento matemático prévio.

Esta seção descreve todo o processo de desenvolvimento do trabalho. Primeiramente, são descritas a coleta e a análise dos dados coletados. Em seguida, apresenta os procedimentos do tratamento de dados e pré-processamento. Concluindo, a seção descreve a estrutura e configuração de cada versão utilizada para atingir os resultados e objetivos do trabalho. Ao todo, foram criadas seis versões principais, sendo três versões utilizando a arquitetura de redes neurais recorrentes, como a GRU, e três versões utilizando a arquitetura de *Transformers*.

4.1 Coleta de dados

A coleta de dados em *Deep Learning* é o processo de obtenção e preparação de conjuntos de dados que serão usados para treinar e avaliar os modelos. Isso envolve a aquisição de dados brutos de várias fontes, como imagens, textos, áudio ou, no caso do presente trabalho, equações de primeiro grau. Além disso, envolve a organização desses dados em um formato adequado para o posterior treinamento. A qualidade e a quantidade dos dados coletados desempenham um papel fundamental na eficácia e no desempenho dos modelos gerados (Escovedo & Koshiyama, 2020).

Neste trabalho, os dados utilizados para a criação dos modelos foram coletados diretamente do banco de dados do sistema tutor inteligente *PAT2Math* (Jaques et al., 2013). Esses dados estão dispostos em tabelas, onde é possível obter diversas informações sobre a interação dos estudantes com o tutor.

Na tabela *log* são descritos todos os passos efetuados pelos estudantes, durante a interação com o *PAT2Math*. Cada linha contém um diferente passo e cada passo contém 39 caraterísticas ou atributos, como segue: *id*, *current step*, *date*, *element clicked*, *error feedback*, *error points*, *got hint before*, *hint*, *hint last minute*, *hints per operation*, *idle log time*, *idle seconds*, *idle server request time*, *initial equation*, *isComplete*, *is pending log*, *key pressed*, *key pressed char*, *last correct step*, *last log time*, *last server request time*, *mouse screenX*, *mouse screenY*, *mouse screen diff*, *mouse x diff*, *mouse x speed*, *mouse y diff*, *mouse y speed*, *points*, *stepCount*, *step feedback*, *step is correct*, *step is final*, *system version*, *timestamp*, *type*, *user id script*, *verified with* e *user id*.

Os registros foram posteriormente filtrados, deixando apenas as instâncias nas quais existe a submissão de um passo pelo estudante, informado pela característica *current step*. Dentre as informações disponibilizadas, além do passo atualmente executado, cabe destaque para equação inicial proposta ao aluno, o registro do último passo digitado corretamente, os *IDs* da interação e dos estudantes, os *feedbacks* emitidos pelo sistema, a verificação que checa se o passo está correto e a verificação que checa se o passo é final. Além disso, demais informações como identificação, tempo e data, interação com elementos da tela e pedido de dicas, também estão disponíveis.

O banco de dados utilizado possui mais de 500 mil interações, realizadas entre os meses de Março e Novembro de 2018, por cerca de 125 estudantes. Desse total, cerca de 137 mil interações envolvem a digitação de um passo.

4.2 Análise de Dados

A análise de dados é o processo de examinar e interpretar os dados, para descobrir informações úteis, identificar tendências e padrões relevantes. É um processo que envolve técnicas e metodologias para interpretar dados de várias fontes em diferentes formatos, para apropriação das informações e auxílio na tomada de decisões (Crabtree, 2023).

A partir dos dados coletados, um arquivo parcial contendo 137 mil instâncias, com as interações dos estudantes, foi carregado no *Google Colaboratory*, contendo apenas as quatro colunas utilizadas no desenvolvimento do código: *initial equation*, que contém a equação inicial disponibilizada no *PAT2Math*; *last correct step*, que contém o último passo digitado pelo aluno corretamente; *currentStep*, que refere-se ao passo atualmente digitado pelo estudante; e *step is correct*, que indica se o passo atualmente digitado está correto ou incorreto, rotulados respectivamente como 1 e 0.

Pode-se notar num primeiro momento, que as colunas *initial equation* e *last correct step*, por conterem equações apresentadas pelo sistema ou, no segundo caso, equações que o sistema já informou como corretas, possuem dados mais robustos, com equações bem formadas. Por outro lado, a coluna *currentStep*, por conter dados digitados diretamente pelos estudantes, possui diversos problemas, como equações mal formadas, múltiplos sinais de "=", caracteres inválidos ou não esperados. Por fim, a coluna *step is correct*, possui diversas informações incorretas, avaliadas com erro pelo sistema. A informação de *step is correct* é bastante importante, visto que essa coluna é o rótulo do modelo a ser criado, ou seja, é o valor que será previsto pelo modelo final.

As colunas também possuem muitos dados faltantes. Enquanto *step is correct* possui 5 instâncias sem valor atribuído, a coluna *last correct step* possui 38.916 instâncias sem valor. Isso ocorre porque *last step correct* só possuirá um valor atribuído nas sequências em que o estudante já digitou um passo corretamente.

Como a ideia do modelo é avaliar somente se determinado passo é correto/incorreto, dado uma equação inicial, é possível utilizar passos intermediários, digitados corretamente, como pontos de partida para os passos seguintes. Dessa forma, sempre que as colunas *initial equation* e *last correct step* contém equações diferentes, uma nova linha é adicionada aos dados, inserindo o valor de *last correct step* como equação inicial e mantendo os demais atributos da linha.

O resultado após a junção das duas colunas pode ser visto na figura 1. A coluna *last correct step* foi excluída, mantendo uma tabela inicial com apenas 3 colunas, mas aumentando o número

de instâncias do conjunto para cerca de 251 mil interações contendo o passo atualmente digitado.

	initial_equation	currentStep	step_is_correct
0	x+4=9	x=9-4	1.0
1	x+4=9	x=-5	0.0
2	x+4=9	x=+5	1.0
3	x+5=8	x=8-3	0.0
4	x+5=8	x=8-3	0.0

Figura 1: Exemplo de dados coletados com as colunas importantes para o modelo..

Essa alteração, é bastante útil para aumentar significativamente a base de dados, mas cria um outro problema, visto o grande número de dados faltantes da coluna *last step correct*, que com a junção, são replicados para *initial equation*. O tratamento de tais inconsistências é descrito na subseção 4.3.

4.3 Tratamento de Dados

Esta subseção inclui a exclusão dos dados faltantes, dados inválidos e dados duplicados, além da correção dos rótulos anotados de forma incorreta pelo sistema especialista do tutor inteligente.

O primeiro passo foi realizar a exclusão de dados duplicados, ou seja, diferentes instâncias que continham os mesmos valores nas duas colunas: *initial equation* e *currentStep*. A exclusão reduziu drasticamente o número de instâncias, passando das cerca de 251 mil, para pouco menos de 35 mil interações.

Essa exclusão é bastante importante, porque se não for realizada, pode acarretar contaminação dos conjuntos, ou seja, dados iguais podem aparecer tanto no conjunto de treinamento quanto no conjunto de validação e testes, o que por sua vez, pode levar o modelo a atingir resultados não confiáveis na avaliação, visto que estará apenas copiando as repostas. O modelo pode atingir percentuais de acurácia muito altos no conjunto de treinamento, mas não vai repetir esse desempenho quando for aplicado a dados novos, não vistos pelo modelo (Chollet, 2021).

Após a exclusão das duplicatas, o próximo passo foi a exclusão dos dados faltantes, ou seja, das instâncias que possuem uma das três colunas com valores nulos ou sem valor atribuído. Esse procedimento resultou na redução do número de instâncias para cerca de 30 mil.

As instâncias contendo resultados inválidos, formações inválidas, que possuíam múltiplos sinais de '=', foram mantidas no conjunto de dados, de forma que o sistema aprenda a reconhecer essas ocorrências e retornar que o passo está incorreto, da forma como foi digitado.

Por fim, os dados incorretos da coluna *step is correct* foram corrigidos. A correção utiliza a biblioteca *SymPy* na resolução de equações. *SymPy* é uma biblioteca de matemática simbólica em *Python*. Ela permite a realização de cálculos matemáticos com símbolos e expressões ao invés de valores numéricos.

A correção foi realizada em quatro passos: verificação de cada instância, buscando caracteres inválidos ou não esperados na coluna *current step*; padronização da coluna *current step* para posterior resolução das equações; resolução das equações nas colunas *initial equation* e *current step*, utilizando a biblioteca *SymPy*; comparação entre os dois resultados.

No caso de resultados iguais, o passo foi considerado correto. Já para resultados diferentes, o passo foi considerado incorreto. Esse resultado, por sua vez, foi comparado ao rótulo constante em *step is correct*, encontrando um total de 2.061 rótulos anotados incorretamente pelo sistema especialista, no banco de dados do *PAT2Math*. Esse número corresponde a quase 7% do total dos rótulos.

Essa identificação tem ainda mais importância, no caso do aumento de dados. Como esse procedimento parte dos dados originais, se os rótulos originais estiverem incorretos, eles também serão replicados de forma incorreta, criando um ruído ainda maior nos dados analisados pelo modelo.

Após a identificação dos rótulos incorretos, eles foram invertidos, ou seja, rótulos contendo o valor 1 (corretos) foram alterados para 0 (incorretos) e rótulos contendo valor 0, receberam o valor 1. Feito isso, foi realizada uma nova verificação, que encontrou 30.279 instâncias, todas elas com o rótulo ajustado. Dessas instâncias, 20.919 estão rotuladas como incorretas (*step is correct* igual a 0) e 9.360 estão rotuladas como corretas (*step is correct* igual a 1), remetendo a um novo problema: o desbalanceamento de dados, tratado na subseção 4.4.

4.4 Balanceamento de Dados

O desbalanceamento de dados ocorre quando as classes em um conjunto de dados não estão igualmente representadas, ou seja, uma ou algumas classes têm muito mais exemplos do que outras. Esse desequilíbrio na distribuição das classes pode ocorrer em conjuntos de dados de diversas naturezas, como classificação binária ou multi-classe, detecção de anomalias, entre outros (Chawla, 2005).

O desbalanceamento pode levar a uma avaliação enganosa do desempenho do modelo, a um baixo desempenho das classes minoritárias e a um viés do modelo, em favor da classe majoritária. Por exemplo, em um conjunto de dados que contém 80% de dados de determinada classe, o modelo pode optar por selecionar sempre essa classe, garantindo uma acurácia de 80%.

Esse problema pode ser tratado utilizando métricas diferentes, como *F1 Score*, que é mais robusta que a acurácia. Outra forma de lidar com o desbalanceamento é a utilização da validação cruzada. No caso deste trabalho, os dados foram balanceados através da técnica conhecida como *data augmentation*.

Essa técnica é amplamente utilizada em aprendizado de máquina, em particular em tarefas de visão computacional, para aumentar o tamanho e a diversidade do conjunto de dados de treinamento, tornando-o mais robusto e capaz de generalizar melhor. Essa técnica envolve a geração de novos exemplos de treinamento a partir dos dados originais, aplicando transformações aleatórias a esses dados, de modo que os exemplos gerados sejam semelhantes, mas não idênticos aos dados originais. O objetivo principal da *data augmentation* é melhorar o desempenho e a capacidade de generalização dos modelos, reduzindo o risco de *overfitting* (Chollet, 2021).

A função de *data augmentation* utilizada neste trabalho, parte das instâncias rotuladas como corretas e altera a coluna *initial equation* ou a coluna *currentStep*, criando novas equações rotuladas como corretas ou incorretas. No caso da geração de equações corretas, a função inclui operações de multiplicação, divisão e adição (ou um conjunto dessas operações) dos dois lados da equação. Depois, também de forma aleatória, pode inverter os lados da equação. Na geração

de equações incorretas, apenas o lado esquerdo da equação é modificado, também podendo ter os lados invertidos posteriormente.

A partir dessa função, foram criadas cerca de 86 mil novas instâncias. Mesmo com uma nova exclusão de dados duplicados, o conjunto de dados passou a ter aproximadamente 115 mil instâncias, sendo metade rotuladas como corretas e a outra metade como incorretas, garantindo um conjunto de dados balanceado.

4.5 Entrada Única

Para utilização nas redes neurais e *transformers*, os dados foram modificados, unindo as colunas *initial equation* e *current step*, além de um separador central, na forma: **initial equation [SEP] current Step**. O resultando dessa união foi atribuído à coluna *input sequence*, passando o conjunto de dados a ter somente duas colunas, a entrada (*input sequence*) e o *label* (*step is correct*). O rótulo também foi modificado de forma a possuir somente os números inteiros 0 ou 1, ao invés de números de ponto flutuante.

A tabela 1 exemplifica a modificação realizada. As equações presentes nas colunas *initial equation* e *currentStep* são unidas na coluna *input sequence*, com o separador central. Já a coluna *step is correct* original (terceira coluna) passa a receber somente números inteiros (quinta coluna).

Tabela 1: Exemplo de como os dados foram combinados e ajustados para o modelo..

initial_equation	currentStep	step_is_correct	input_sequence	step_is_correct
x+4=9	x=9-4	1.0	x+4=9 [SEP] x=9-4	1
x+4=9	x=-5	0.0	x+4=9 [SEP] x=-5	0
x+4=9	x=+5	1.0	x+4=9 [SEP] x=+5	1
x+5=8	x=8-3	0.0	x+5=8 [SEP] x=8-3	0
x+5=8	x=8-5	1.0	x+5=8 [SEP} x=8-5	1

4.6 Tokenização e Encoding

A tokenização é o processo de dividir um texto ou uma sequência de caracteres em unidades menores chamadas *tokens*. Um *token* é uma unidade textual significativa, que pode ser uma palavra, parte de uma palavra (sub-palavra), frase, símbolo ou qualquer outra unidade que faça sentido para a análise de texto. A tokenização é uma etapa fundamental no processamento de linguagem natural e é frequentemente usada em tarefas como análise de texto, tradução automática, sumarização de texto, classificação de documentos, entre outras (Lane et al., 2019).

Este trabalho realiza a tokenização de cada caractere das equações. A partir deste ponto, há diferenças nos procedimentos realizados no conjunto de dados para o treinamento com as versões que utilizam redes neurais e para treinamento das versões que utilizam *transformers*. Com relação à tokenização, a versão de redes neurais usa a função *Tokenizer*, da biblioteca *keras*.

Já na versão que usa *transformers*, a tokenização é realizada utilizando o *DistilBertTokenizerFast*. Essa função faz parte da biblioteca *Transformers*, usada para trabalhar com modelos de linguagem pré-treinados, incluindo modelos baseados na arquitetura BERT (*Bidirectional Encoder Representations from Transformers*) e suas variantes, como o modelo *DistilBERT*. A tokenização é realizada nas colunas *initial equation* e *currentStep*, onde cada número, variável ou sinal

presente na equação, passa a representar um *token* (Sanh et al., 2020).

Após a divisão do texto (neste caso, da equação) em *tokens*, é necessário codificar cada um desses *tokens* para números inteiros. Há várias formas de realizar essa codificação, como por exemplo, a técnica de *One-Hot Encoding*, que transforma cada categoria única em um vetor binário, onde cada elemento no vetor representa uma categoria e é marcado como 1 se a observação pertence a essa categoria e 0 caso contrário. Já a técnica de *Word Embeddings* faz uma representação numérica para cada caractere ou palavra, mantendo as relações semânticas e sintáticas entre elas (Chollet, 2021).

Neste trabalho, os próprios tokenizadores utilizados já fazem o trabalho de *encoding*. O *DiltilBertTokenizerFast* utiliza uma técnica de *encoding* chamada *Word piece tokenization*, que divide o texto em subpalavras. O tokenizador também faz o preenchimento (*padding*), garantindo sequências de mesmo tamanho. Já na versão de redes neurais, o *Tokenizer* do pacote *keras* utiliza uma abordagem mais direta de *encoding*, chamada de *integer encoding*, convertendo as palavras em inteiros. O preenchimento é realizado através da função *pad sequences* (Sanh et al., 2020).

4.7 Divisão de dados

Os dados consolidados são embaralhados, para que não existam muitas instâncias similares em conjuntos iguais, e finalmente divididos entre o conjunto de treinamento, conjunto de validação e conjunto de testes. O conjunto de treinamento ficou com 70% do número total de dados, o conjunto de validação, com 20% e o conjunto de testes, com 10%.

O conjunto de treinamento é utilizado para treinar o modelo; o conjunto de validação contém dados não vistos pelo conjunto de treinamento e é utilizado para verificar a perda e acurácia em dados diferentes do conjunto original, permitindo evitar o *overfitting* e *underfitting*. Finalmente, o conjunto de testes é usado após a finalização do modelo e ajuste dos hiperparâmetos, para fazer a análise final e apurações dos resultados.

4.8 Modelo

Este trabalho utiliza duas arquiteturas principais para criação do modelo: rede Neural GRU e *transformers*. A rede GRU (*Gated Recurrent Unit*) utilizada é uma rede neural recorrente, similar às LSTMs. Ela é criada a partir de uma camada sequencial, terminando em uma camada densa, ativado por um *sigmoid*, que é uma função estatística utilizada na classificação binária (Chung et al., 2014).

Já para a arquitetura de *transformers*, foi utilizado o modelo *TF Distil Bert For Sequence Classification*. O *DistilBERT* é uma versão mais compacta do *BERT (Bidirectional Encoder Representations from Transformers)*, introduzido por Devlin et al. (2018) e fornecido pela biblioteca *Hugging Face*. Por sua vez, o *TF Distil Bert For Sequence Classification* é uma extensão do *DistilBERT* que é ajustada especificamente para tarefas de classificação de sequências, como análise de sentimentos e classificação de texto (Sanh et al., 2020).

A *Hugging Face* é plataforma de código aberto que se concentra em fornecer ferramentas e recursos avançados para o processamento de linguagem natural e aprendizado de máquina. Possui uma grande biblioteca *transformers*, que é referência no desenvolvimento e uso de modelos pré-

treinados para processamento de linguagem natural, como *BERT*, *GPT-2*, *RoBERTa*, entre outros.

Antes de iniciar o treinamento, é possível ajustar alguns parâmetros, chamados de hiperparâmetros, para diferenciá-los dos valores e pesos que são aprendidos pelas redes (Chollet, 2021). Abaixo são listados alguns dos principais hiperparâmetros utilizados:

Otimizador: O otimizador é um algoritmo que ajusta os pesos dos parâmetros do modelo durante o treinamento com o objetivo de minimizar a função de perda. Exemplos de otimizadores incluem o *SGD (Stochastic Gradient Descent)*, o *Adam* e o *RMSProp*. O otimizador *Adam (Adaptive Moment Estimation)*, usado neste trabalho, combina características de dois outros otimizadores, o *Momentum* e o *RMSProp*, e introduz adaptações que ajudam a acelerar a convergência e melhorar a eficiência do treinamento (Chollet, 2021).

Taxa de Aprendizado (Learning Rate): A taxa de aprendizado controla o tamanho dos passos que o otimizador dá durante o treinamento. Uma taxa de aprendizado alta pode levar a oscilações e divergências, enquanto uma taxa de aprendizado muito baixa pode levar a convergência lenta (Chollet, 2021).

Função de Perda (Loss): A função de perda é uma medida que quantifica o quão bem o modelo está performando em relação à tarefa. Durante o treinamento, o otimizador ajusta os parâmetros do modelo para minimizar a função de perda (Chollet, 2021).

Nas versões que utilizam redes neurais, foi utilizada a função *Binary Crossentropy* (Entropia Cruzada Binária), que é projetada especificamente para classificação binária, nos casos em que cada exemplo pertence a uma de duas classes possíveis. Já no caso das versões com *transformers*, foi selecionada a função *Stochastic Categorical Crossentropy* (Entropia Cruzada Categórica Estocástica), que é uma função usada em problemas de classificação multi-classe, onde cada exemplo de treinamento pode pertencer a uma única classe dentre várias classes possíveis.

Métricas de Avaliação: As métricas de avaliação são usadas para medir o desempenho, seja durante o processo de treinamento e validação, seja para verificar o desempenho final do modelo, com dados novos, não vistos nas fases anteriores. Exemplos de métricas incluem acurácia, precisão, *recall*, *F1-score*, *kappa*, dentre outras. Este trabalho utiliza apenas a acurácia, que é suficientemente boa para verificar a eficiência dos modelos aqui dispostos nos conjuntos de dados.

Tamanho do Batch (Batch Size): O tamanho do lote determina quantos exemplos de treinamento são processados em cada iteração. Um tamanho de lote maior pode acelerar o treinamento, mas também pode exigir mais memória. Um tamanho de lote menor pode ajudar na convergência.

Na seção 5, são apresentadas cada uma das versões elaboradas para a criação do modelo final, destacando as diferenças entre cada uma, descrevendo e avaliando os resultados obtidos.

5 Avaliação e Resultados

Esta seção descreve mais detalhadamente cada uma das versões utilizadas para a criação do modelo para automatização do módulo de domínio de um sistema tutor inteligente baseado em passos. Durante o desenvolvimento do trabalho, foram utilizadas duas arquiteturas diferentes: redes neurais recorrentes GRU e *transformers*. Para as arquiteturas citadas, foram criadas seis versões principais, que foram sendo aprimoradas e são listadas nas subseções seguintes.

Para facilitar a compreensão, essas versões foram agrupadas na tabela 2, identificadas pela arquitetura, número de instâncias utilizadas e ajuste dos rótulos. Quanto à arquitetura, são três versões utilizando as redes neurais GRU e três utilizando *transformers*. Quanto ao número de instâncias, duas versões foram testadas com aproximadamente 45 mil instâncias, duas com 80 mil e as duas últimas versões de cada arquitetura utilizaram cerca de 115 mil instâncias. Essas diferentes quantidades de instâncias foram geradas por *data augmentation* para testar a robustez dos modelos. Finalmente, quanto aos rótulos, quatro versões utilizaram os rótulos originais, conforme anotados pelo sistema especialista do *PAT2Math* e em apenas duas versões, os rótulos foram corrigidos.

Tabela 2: Detalhes das seis versões de modelos desenvolvidas: arquitetura, número de instâncias e se houve correção de rótulos..

Versões	Arquitetura	Nr. Instâncias	Rótulos
Versão 1	GRU	∼45 mil	Originais
Versão 2	GRU	∼80 mil	Originais
Versão 3	GRU	∼115 mil	Corrigidos
Versão 4	Transformers	∼45 mil	Originais
Versão 5	Transformers	∼80 mil	Originais
Versão 6	Transformers	∼115 mil	Corrigidos

Os resultados obtidos por cada um dos modelos são comparados e avaliados, chegando ao modelo final, que atingiu o maior percentual de acurácia na correção dos passos digitados pelos alunos, com relação à equação proposta.

5.1 Redes Neurais

Das seis versões de código testadas neste trabalho, três delas possuem arquitetura de redes neurais, mais especificamente de GRUs. Essas versões foram treinadas e validadas utilizando 25 épocas e foram configuradas com os mesmos hiperparâmetros, como segue:

- **Otimizador:** Adam, com *learning rate* de $3e-5$;
- **Função de Perda:** *Binary Crossentropy*;
- **Tamanho do Batch:** 32;
- **Métrica:** Acurácia;

*5.1.1 **Versão 1***

A primeira versão do modelo com redes neurais, utilizou os dados do conjunto original do *PAT2Math*, acrescentando apenas as instâncias corretas necessárias para o balanceamento dos dados. Dessa forma, o modelo foi treinado com cerca de 44.854 instâncias, sendo 22.426 corretas e 22.428 incorretas.

O destaque desta versão, é a sua simplicidade e rapidez na execução do treinamento. O modelo foi treinado em apenas 2 minutos e 37 segundos. Quanto aos resultados, enquanto no conjunto de treinamento, a acurácia chegou a 89%, no conjunto de validação, a acurácia atingiu **84,32%** e a função de *loss* chegou a $0,3433$. A escolha pelo melhor modelo é realizada pelo

callback Model Checkpoint, sendo selecionado de acordo com o menor valor da função de perda no conjunto de validação.

Esse modelo demonstra que com arquiteturas simples e uma base de dados pequena, é possível atingir bons resultados. O resultado final deste modelo foi de **83%** de acurácia na correção de equações do conjunto de testes. Foram 3.725 instâncias preditas corretamente e 761 preditas incorretamente.

5.1.2 ***Versão 2***

A segunda versão desta arquitetura utilizou um número bem superior de dados, totalizando 78.307 instâncias, sendo que em 39.078 o passo está correto e em 39.229 o passo está incorreto. Desta vez, através da função de *data augmentation*, foram gerados tanto instâncias corretas quanto incorretas.

O treinamento dessa versão de rede também foi executado com rapidez, sendo concluído em apenas cinco minutos e 25 segundos. No conjunto de treinamento, houve um aumento de cerca de três pontos percentuais na acurácia, chegando a $92,32\%$. No conjunto de validação, a acurácia chegou a **87,33%**, também um aumento de três pontos percentuais. A função de perda ficou em $0,2911$.

O resultado apurado para o modelo, no conjunto de testes, foi de **87,33%**, cerca de quatro pontos percentuais acima da versão anterior. Foram 6.839 rótulos classificados corretamente e 992 incorretamente.

5.1.3 ***Versão 3***

A versão final, das que utilizam arquitetura de redes neurais, contou com ajustes no algoritmo de *data augmentation*, aumento no número de instâncias para 114.930 (em torno de 50% de cada classe) e ajuste dos rótulos incorretos constantes nos dados originais, que geravam o que é chamado de ruído.

O treinamento foi executado em somente sete minutos e 28 segundos e a acurácia no conjunto de validação chegou a **89,48%**, com $0,2304$ como resultado de perda. O resultado é cerca de dois pontos percentuais superior ao modelo anterior e pode ser conferido nos gráficos das figuras 2 e 3.

O resultado final deste modelo e da arquitetura de redes neurais foi de **90,02%** no conjunto de testes. São 10.346 rótulos classificados corretamente e 1.147 classificados de forma incorreta, como pode ser visto na matriz de confusão da figura 4. Em azul escuro, constam os quantitativos dos rótulos preditos corretamente e em azul claro, os rótulos preditos incorretamente, divididos pelo *label* 0 e 1.

5.2 Tranformers

De forma muito similar às redes neurais, são apresentadas três versões utilizando *transformers*. As versões foram testadas e validadas com 25 épocas e seguem a configuração abaixo listada:

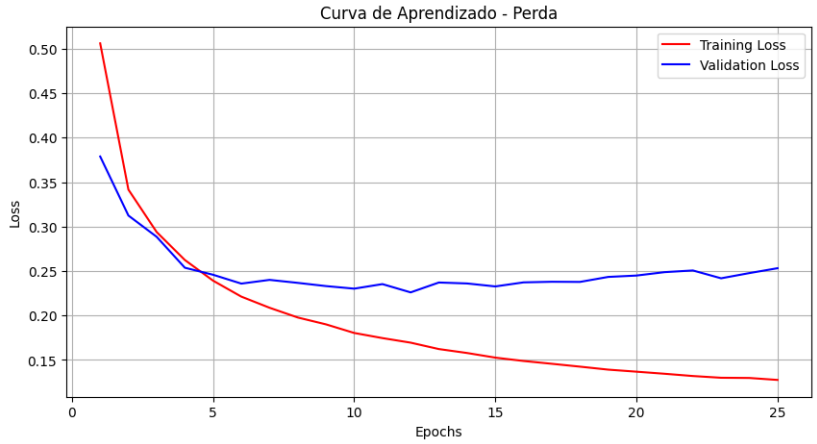

Figura 2: Versão 3 - Curva de Aprendizado - Perda.

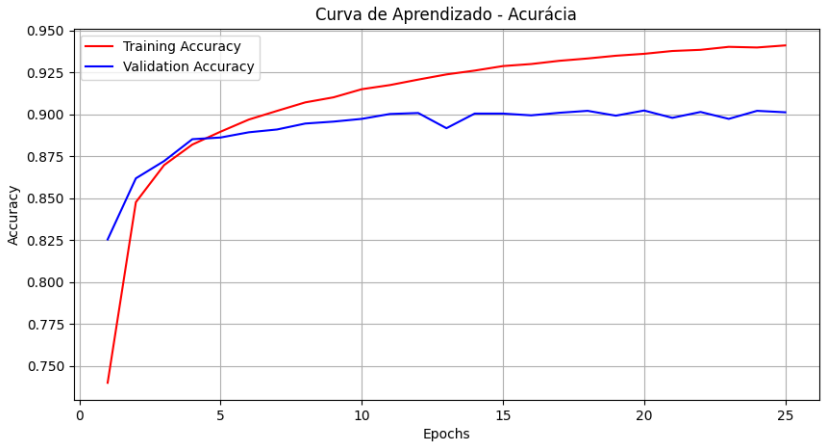

Figura 3: Versão 3 - Curva de Aprendizado - Acurácia.

- **Otimizador:** Adam, com *learning rate* de $3e-5$;
- **Função de Perda:** *Sparse Categorical Crossentropy*;
- **Tamanho do Batch:** 32;
- **Métrica:** Acurácia;

*5.2.1 **Versão 4***

A primeira versão com *transformers* neste trabalho, também foi testada usando cerca de 40 mil instâncias (na verdade, 44.851). Diferente das demais versões, esta versão foi utilizada tanto com equações em notação infixa quanto com equações em notação pósfixada. Como o resultado foi um pouco inferior utilizando a notação pósfixada, a decisão foi por manter apenas a notação infixa nas versões posteriores, que tornam a descrição das equações mais claras e simples.

A primeira diferença óbvia é o tempo de treinamento. As versões que estão utilizando a arquitetura *transformers* demoram bem mais tempo no treinamento do modelo. Para fins de

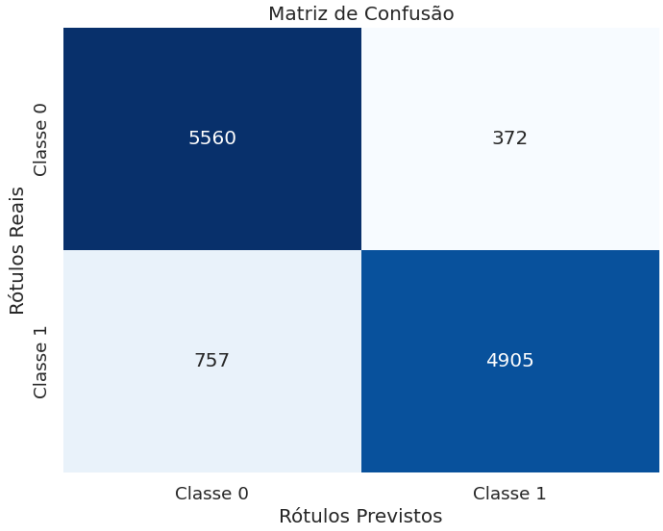

Figura 4: Versão 3 - Matriz de confusão.

comparação, cada época deste modelo leva cerca de três minutos e meio para ser treinada, metade do tempo de treinamento total da última versão com redes neurais.

Esta versão levou cerca de duas horas para conclusão do treinamento. No conjunto de validação, o modelo atingiu uma acurácia de **85,6%** e a função de perda chegou a $0,2873$, valores bem inferiores ao modelo completo com a arquitetura de redes neurais.

Apesar do resultado na validação, no resultado final, o modelo atingiu **88,07%** de acurácia no conjunto de testes, um resultado dois pontos percentuais menor do que o melhor resultado obtido com redes neurais. Foram 3.951 rótulos classificados corretamente e 535 classificados incorretamente.

5.2.2 ***Versão 5***

A segunda versão da arquitetura *transformers* já utiliza um número maior de instâncias, que foram geradas pela função *data augmentation*. No total, são 77.215 instâncias usadas para criação do modelo.

O treinamento demorou cerca de duas horas e 20 minutos para ser completado. Nessa versão, a acurácia chegou a **92,42%** e a função de *loss* a $0,1528$ no conjunto de validação. Os valores já são bem superiores ao de todas as demais versões.

No conjunto de testes, o modelo chegou a um percentual de **93,5%** de acurácia. Mais de três pontos percentuais acima do melhor modelo anterior. Foram 7.221 instâncias classificadas corretamente e 501 de forma incorreta.

5.2.3 ***Versão 6***

Esta é a última versão da arquitetura *transformers* e é o modelo que atingiu a melhor performance dentre todas, sendo a escolhida como modelo e resultado final deste trabalho. Esta versão utilizou

um total de 114.884 instâncias, aproximadamente a mesma quantidade da versão 3. Em ambos os casos, a expansão da aplicação da função de *data augmentation* acarretou número maior de instâncias. Além disso, nesta versão houve a correção dos rótulos anotados de forma incorreta pelo sistema tutor inteligente.

O treinamento dessa versão foi o mais demorado, levando cerca de 5 horas para a conclusão. Após o treinamento, o modelo gerado chegou a um percentual de **94,79%** de acurácia no conjunto de validação e um valor de $0,1193$ na função de perda. O valor é mais de 2% superior ao segundo colocado e as variações podem ser visualizadas nos gráficos das figuras 5 e 6.

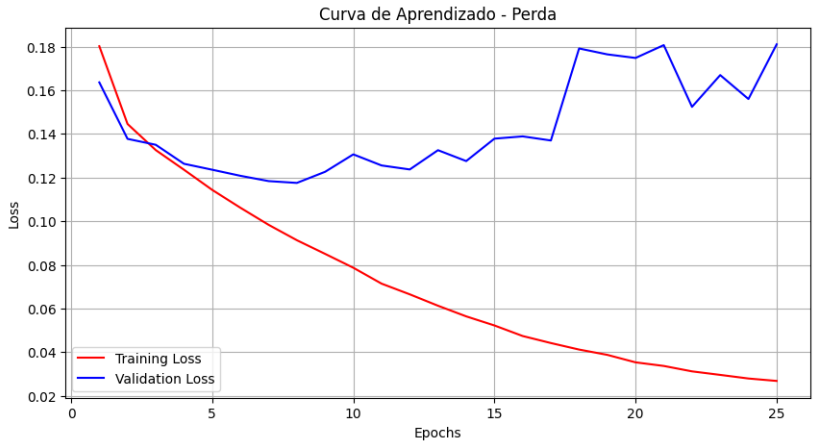

Figura 5: Versão 6 - Curva de Aprendizado - Perda.

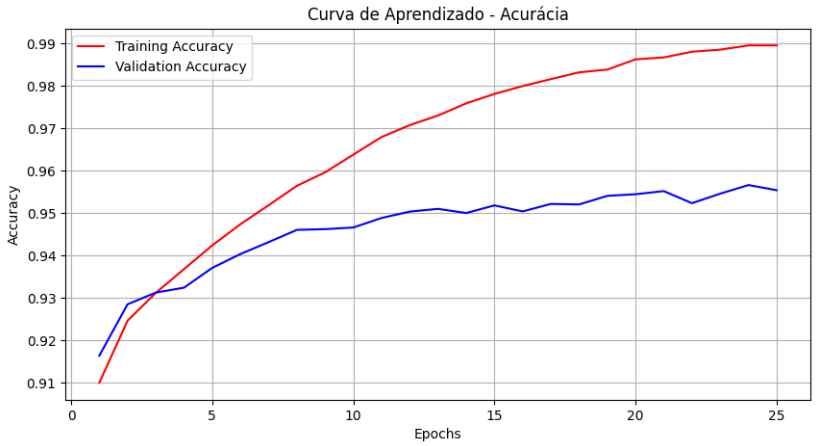

Figura 6: Versão 3 - Curva de Aprendizado - Acurácia.

No conjunto de teste, o modelo atingiu **95,52%** de acurácia, sendo esse o resultado final do modelo e o resultado final deste trabalho. No total foram 10.976 instâncias classificadas corretamente, contra 514 classificadas incorretamente. O gráfico da figura 7 e a matriz de confusão da figura 8 mostram esses números.

Outra análise interessante que pode ser realizada é da tabela disposta na figura 9, que mostra os rótulos classificados e a probabilidade com que o sistema decide por uma ou outra classe. A

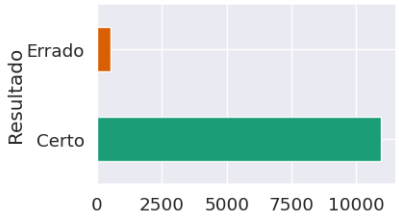

Figura 7: Versão 6 - Quantidade de rótulos classificados.

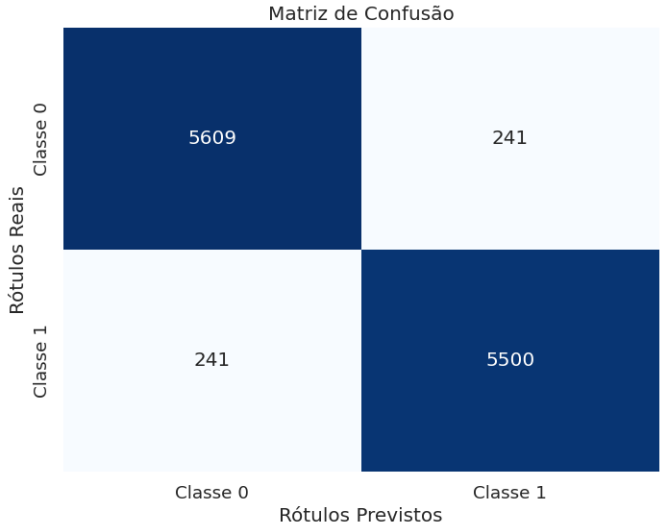

Figura 8: Versão 6 - Matriz de confusão.

primeira e a segunda coluna, indicam respectivamente a equação proposta e o passo atual digitado. Na terceira coluna consta a probabilidade com que o sistema seleciona uma classe (acima de 50% o rótulo é classificado como correto e abaixo de 50%, o rótulo é classificado como incorreto). A quarta e quinta colunas apresentam o rótulo predito e o rótulo real. Por fim, a coluna resultado informa se o sistema acertou ou errou a predição.

A coluna resultado foi ordenada justamente para que ficassem dispostos apenas rótulos classificados incorretamente. Assim, é possível checar o funcionamento do modelo, de forma a aprimorá-lo a partir de uma maior generalização dos dados. Eventualmente, também é possível verificar alguns casos de rótulos anotados incorretamente.

5.3 Avaliação

A tabela 3 mostra as seis versões utilizadas, para fins de comparação. A primeira coluna informa a arquitetura utilizada, se rede neural GRU ou *transformers*. A segunda coluna mostra o número total de instâncias utilizadas na criação do modelo. A terceira coluna indica se os rótulos que estavam originalmente incorretos no banco de dados do sistema tutor, foram corrigidos. As colunas quatro, cinco e seis, informam respectivamente, o tempo total para treinamento do modelo e as acurácias no conjunto de validação e conjunto de testes. O resultado final de cada modelo é dado pelo conjunto de testes.

Equação Inicial	Passo Atual	Probabilidade (%)	Rótulo Predito	Rótulo Verdadeiro	Resultado
(80) / (50) = (x) / (20)	800 = 25x	0.005406988202594221	0	1	Errado
4x + 6 + 5x - 5 = 16	4x + 5x = 16 - 6	58.88481140136719	1	0	Errado
12x - 3x - 2x = 12 + 3	7x = 15	23.251953721046448	0	1	Errado
(3x + 6) / (8) = (2x + 10) / (6)	x = 40 - 18	32.88757801055908	0	1	Errado
- x = - (216) / (- 1)	- x = 216	25.98384916782379	0	1	Errado
x = - (8) / (12)	x = - 4 / 6	0.1791434625307083	0	1	Errado
(2 * (x + 1)) / (3) = 3x + (6) / (5)	10x - 15x = 18 - 10	51.86859369277954	1	0	Errado
16x - 8 = 30x - 12 - 10	x = - 1	94.38527226448059	1	0	Errado
- x + 100 = - 200	- x = 200 - 100	64.08470869064331	1	0	Errado
12x + 24 = 168	(67) / (6) = x	68.86509656906128	1	0	Errado
(12x + 12) - (24 - 24x) = (- 36 + 36x) - (- 48x + 60)	12x + 12 - 24 + 24x = - 36 + 36x + 48x - 60	0.02443837292958051	0	1	Errado
x = (90) / (- 15)	x = (30) / (- 3)	99.38021898269653	1	0	Errado

Figura 9: Tabela mostrando as equações propostas, os passos atuais digitados, as probabilidades de classificação, os rótulos preditos e reais, e o resultado da predição do modelo..

Tabela 3: Comparação detalhada das seis versões dos modelos, incluindo a arquitetura, número de instâncias, correção de rótulos, tempo de treinamento, acurácias de validação e teste..

Versões	Arquitetura	Nr. Instâncias	Correção de Rótulos	Tempo de treinamento	Acurácia Validação	Acurácia Teste
Versão 1	GRU	44.854	Não	2m37s	84,32%	83%
Versão 2	GRU	78.307	Não	5m25s	87,33%	87,33%
Versão 3	GRU	114.930	Sim	7m28s	89,48%	90,02%
Versão 4	Transformer	44.851	Não	2h	85,6%	88,07%
Versão 5	Transformer	77.215	Não	2h20	92,42%	93,5%
Versão 6	Transformer	114.884	Sim	5h	94,79%	95,52%

A versão com melhor desempenho foi a versão 6, que utilizou *transformers* e um dos maiores conjuntos de dados. Essa versão também teve os rótulo corrigidos. A **versão 6** atingiu um resultado final de **95,52%** de acurácia no conjunto de testes.

Dentre as conclusões retiradas, com base na tabela 3, nota-se que as versões implementadas com redes neurais obtiveram um tempo de treinamento muito inferior, entretanto, ao mesmo tempo, tiveram resultados inferiores às versões que utilizaram *transformers*.

Igualmente, as versões que trabalharam com mais dados, que tiveram suas instâncias aumentadas por *data augmentation*, conseguiram generalizar melhor e atingiram melhores resultados do que suas versões com a mesma arquitetura. O mesmo pode ser dito com relação à correção dos rótulos, visto que essas versões com rótulos corrigidos não precisaram lidar com o ruído nos dados, que informavam *labels* incorretos.

6 Conclusão

Este trabalho apresentou o desenvolvimento de um modelo para automatização do módulo de domínio de um sistema tutor inteligente baseado em passos, realizando a correção de equações de primeiro grau, através de um modelo de *Deep Learning* e Processamento de Linguagem Natural.

Primeiramente, foram descritas a fundamentação teórica e os conceitos relacionados a essa tarefa. Depois foram listados os trabalhos relacionados na mesma área. Em seguida, este trabalho detalhou todo o desenvolvimento das versões e da criação do modelo, iniciando pela coleta e análise de dados, passando pelo pré-processamento e concluindo na descrição das arquiteturas. Por fim, foram apresentados os resultados de cada um das versões criadas, seguidas de uma tabela

comparativa e da análise dos seus desempenhos.

A partir dessa análise, o modelo com melhor desempenho, alcançou cerca de 95,5% de acurácia no conjunto de testes, utilizando uma arquitetura de *transformers*. Mesmo as versões que utilizaram redes neurais também apresentaram bons resultados.

O resultado final foi bastante robusto e indica que o modelo conclui com êxito o objetivo principal deste trabalho, garantindo a possibilidade de automatização do módulo de domínio de um sistema tutor inteligente baseado em passos, através da avaliação/correção de equações e dos passos digitados pelos estudantes.

Também é possível antever que, a partir de alguns ajustes pontuais e, principalmente, da ampliação do conjunto de dados disponíveis, que permitiriam uma maior generalização, o modelo tende a atingir resultados ainda melhores, tornando-se um modelo completamente autônomo. Uma futura integração do modelo de *deep learning* ao sistema tutor inteligente, garantiria não somente uma constante fonte de novos dados, mas também o constante aprendizado e auto-aperfeiçoamento do modelo.

Este trabalho também buscou contribuir com a pesquisa na área de Processamento de Linguagem Natural aplicada à matemática, com o aprimoramento dos sistema tutores inteligentes e de ferramentas que possibilitem a ampliação do aprendizado a distância, que ganhou notória relevância, durante a pandemia da *COVID-19*.

Agradecimentos

Este estudo foi parcialmente financiado pelas seguintes agências brasileiras de financiamento de pesquisa: FAPERGS (processo 17/2551-0001203-8) e CNPq (processo 306005/2020-4).

Referências

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*. [GS Search].

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901. [GS Search].

Charton, F., Hayat, A., & Lample, G. (2020). Learning advanced mathematical computations from examples. *arXiv preprint arXiv:2006.06462*. [GS Search].

Chawla, N. V. (2005). Data Mining for Imbalanced Datasets: An Overview. *Data Mining and Knowledge Discovery Handbook*, 853–867. [GS Search].

Chollet, F. (2021). *Deep Learning with Python, Second Edition*. Manning. https://books.google.com.br/books?id=mjVKEAAAQBAJ [GS Search].

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. [GS Search].

Crabtree, A., M. Nehme. (2023). *What is Data Analysis? An Expert Guide With Examples*. https://www.datacamp.com/blog/what-is-data-analysis-expert-guide

DeepLearning.ai. (2020). Natural Language Processing with Attention Models. https://www.coursera.org/learn/attention-models-in-nlp/home

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., & Kaiser, L. (2018). Universal transformers. *arXiv preprint arXiv:1807.03819*. [GS Search].

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, *abs/1810.04805*. http://arxiv.org/abs/1810.04805 [GS Search].

Escovedo, T., & Koshiyama, A. (2020). *Introdução a Data Science: Algoritmos de Machine Learning e métodos de análise*. Casa do Código. https://books.google.com.br/books?id=cL7TDwAAQBAJ [GS Search].

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., & Steinhardt, J. (2021). Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv preprint arXiv:2103.03874*. [GS Search].

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-term Memory. *Neural computation*, *9*, 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735 [GS Search].

Jaques, P. A., Seffrin, H., Rubi, G., de Morais, F., Ghilardi, C., Bittencourt, I. I., & Isotani, S. (2013). Rule-based expert systems to support step-by-step guidance in algebraic problem solving: The case of the tutor PAT2Math. *Expert Systems With Applications*, *40*(14), 5456–5465. https://doi.org/10.1016/j.eswa.2013.04.004 [GS Search].

Lample, G., & Charton, F. (2019). Deep Learning for Symbolic Mathematics. *arXiv preprint arXiv:1912.01412*. [GS Search].

Lane, H., Hapke, H., & Howard, C. (2019). *Natural Language Processing in Action: Understanding, analyzing, and generating text with Python*. Manning Publications. https://books.google.com.br/books?id=UyHgswEACAAJ [GS Search].

Luong, M.-T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*. [GS Search].

Nagori, V., & Trivedi, B. (2012). Which type of Expert System – Rule Base, Fuzzy or Neural is Most Suited for Evaluating Motivational Strategies on Human Resources :- An Analytical Case Study. *International Journal of Business Research and Management (IJBRM)*, *3*(5), 249–254. https://ideas.repec.org/a/aml/intbrm/v3y2012i5p249-254.html [GS Search].

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9. [GS Search].

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2019). Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *CoRR*, *abs/1910.10683*. http://arxiv.org/abs/1910.10683 [GS Search].

Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach* (3ª ed.). Prentice Hall. [GS Search].

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*. [GS Search].

Santoro, A., Hill, F., Barrett, D. G. T., Morcos, A. S., & Lillicrap, T. (2018). Measuring abstract reasoning in neural networks. *ArXiv*, *abs/1807.04225*. [GS Search].

Saxton, D., Grefenstette, E., Hill, F., & Kohli, P. (2019). Analysing Mathematical Reasoning Abilities of Neural Models. *arXiv preprint arXiv:1904.01557*. [GS Search].

Schlag, I., Smolensky, P., Fernandez, R., Jojic, N., Schmidhuber, J., & Gao, J. (2019). Enhancing the transformer with explicit relational encoding for math problem solving. *arXiv preprint arXiv:1910.06611*. [GS Search].

TensorFlow. (2020). Transformer model for language understanding. https://www.tensorflow.org/tutorials/text/transformer

Trask, A. W. (2019). *Grokking deep learning*. Simon; Schuster. [GS Search].

VanLehn, K. (2011). The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. *Educational Psychologist*, *46*(4), 197–221. https://doi.org/10.1080/00461520.2011.611369 [GS Search].

VanLehn, K. (2006). The Behavior of Tutoring Systems. *Int. J. Artif. Intell. Ed.*, *16*(3), 227–265. http://dl.acm.org/citation.cfm?id=1435351.1435353

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (s.d.). Attention is All you Need. Em I. G. et al. (Ed.), *Advances in Neural Information Processing Systems 30*.

Wangperawong, A. (2019). Attending to Mathematical Language with Transformers. *arXiv e-prints*, **jourarticle** arXiv:1812.02825, arXiv:1812.02825. [GS Search].

Woolf, B. P. (2007). *Building Intelligent Interactive Tutors: Student-centered Strategies for Revolutionizing e-Learning*. Morgan Kaufmann Publishers Inc. [GS Search].

Appendix 1

O código final deste trabalho, referente a versão 6, está disponível no *GitHub* e pode ser aberto no *Google Colaboratory*, através deste link.