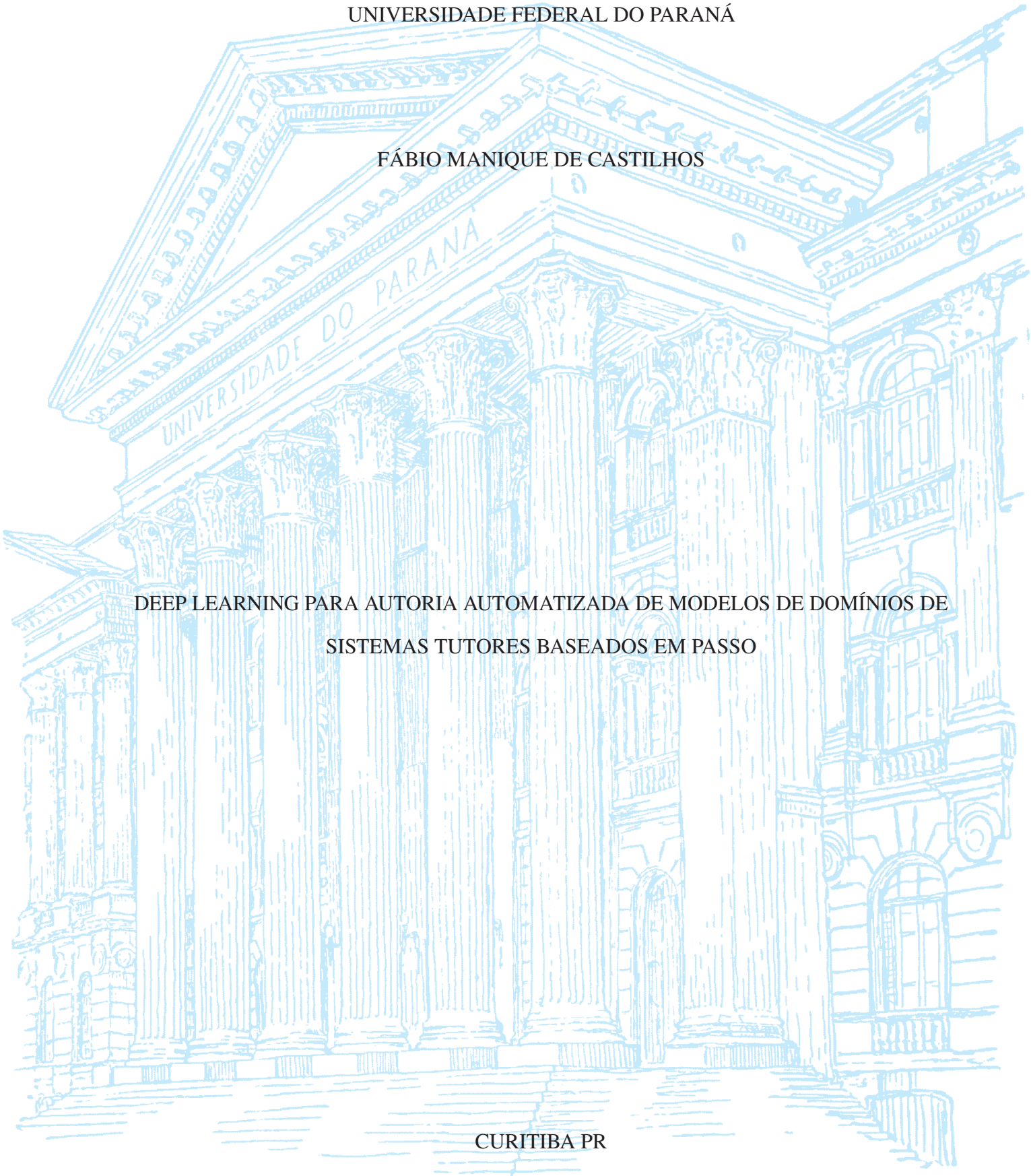
UNIVERSIDADE FEDERAL DO PARANÁ

FÁBIO MANIQUE DE CASTILHOS



DEEP LEARNING PARA AUTORIA AUTOMATIZADA DE MODELOS DE DOMÍNIOS DE SISTEMAS TUTORES BASEADOS EM PASSO

CURITIBA PR

2023

FÁBIO MANIQUE DE CASTILHOS

DEEP LEARNING PARA AUTORIA AUTOMATIZADA DE MODELOS DE DOMÍNIOS DE SISTEMAS TUTORES BASEADOS EM PASSO

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Patrícia A. Jaques Maillard.

CURITIBA PR

2023

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA CIÊNCIA E TECNOLOGIA

Castilhos, Fábio Manique de
Deep learning para autoria automatizada de modelos de domínios de sistemas tutores baseados em passo. / Fábio Manique de Castilhos. – Curitiba, 2023.
1 recurso on-line : PDF.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.

Orientadora: Patrícia Augustin Jaques Maillard.

1. Aprendizado profundo. 2. Transformadores. 3. Programa de computadores (STIs). I. Maillard, Patrícia Augustin Jaques. II. Universidade Federal do Paraná. Programa de Pós-Graduação em Informática. III. Título.

Bibliotecária: Roseny Rivelini Morciani CRB-9/1585

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **FÁBIO MANIQUE DE CASTILHOS** intitulada: **DEEP LEARNING PARA AUTORIA AUTOMATIZADA DE MODELOS DE DOMÍNIOS DE SISTEMAS TUTORES BASEADOS EM PASSO**, sob orientação da Profa. Dra. PATRICIA AUGUSTIN JAQUES MAILLARD, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 29 de Setembro de 2023.

Assinatura Eletrônica
02/10/2023 12:10:40.0
PATRICIA AUGUSTIN JAQUES MAILLARD
Presidente da Banca Examinadora

Assinatura Eletrônica
02/10/2023 17:23:17.0
RACHEL CARLOS DUQUE REIS
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
05/10/2023 19:47:52.0
CRISTIAN CECHINEL
Avaliador Externo (UNIVERSIDADE FEDERAL DE SANTA CATARINA)

AGRADECIMENTOS

Agradeço primeiramente a minha mãe e meu pai, que sempre acreditaram em mim. A minha namorada, que esteve do meu lado todo esse tempo. A minha orientadora Patricia, que me incentivou a começar o Mestrado e não me deixou desisir em nenhum momento. A minha irmã, meu cunhado e meus sobrinhos. A meus amigos, colegas de trabalho e todos que de alguma forma colaboraram para que este momento chegasse, um muito obrigado!

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 $95,84\%$ 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 95.84% accuracy rate in evaluating/correcting these equations.

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

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

AI	Artificial Intelligence
AMPS	Auxiliary Mathematics Problems and Solutions
BERT	Bidirectional Encoder Representations from Transformers
BWD	Backward generation
DINF	Departamento de Informática
FWD	Forward Generation
GRU	Gated Recorrent Unit
GPT	Generative Pre-training for Transformer
IBP	Integration by parts
LSTM	Long Short-Term Memory
MATH	Mathematics Aptitude Test of Heuristics
PAT2Math	Personal Affective Tutor to Math
PLN	Processamento de Linguagem Natural
PPGINF	Programa de Pós-Graduação em Informática
RNC	Rede Neural Convolucional
RNR	Rede Neural Recorrente
RMC	Relational Memory Core
Seq2seq	Sequence-to-Sequence
STI	Sistemas Tutores Inteligentes
SA	Step Analyzer
SG	Step Generator
SGD	Stochastic Gradient Descent
T5	Text-to-Text Transfer Transformer
TPMHA	Tensor Product Multiheaded Attention
UFPR	Universidade Federal do Paraná
XML	Extensible Markup Language

SUMÁRIO

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, que é composto por dois módulos: *step analyzer* (SA) e *step generator* (SG) (Jaques et al., 2013). O SG é responsável por gerar o próximo passo e por fornecer dicas ao aluno. O SA analisa e classifica os passos realizados pelo estudante, podendo retornar os componentes de conhecimento que o aluno utilizou e a descrição de um determinado erro (VanLehn, 2006).

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 torna-se uma alternativa para solucionar esses problemas e aprimorar o desenvolvimento dos sistemas tutores. Uma possibilidade de solução para a automatização desse processo é a utilização de *Deep Learning*, buscando uma implementação mais rápida e simples e a redução da complexidade de atualização, que permitiria um menor tempo de execução.

Deep Learning é amplamente utilizada para extrair valor da quantidade gigantesca de dados gerados diariamente, sendo utilizada nas mais diversas aplicações, como predição, reconhecimento de imagens e processamento de linguagem natural (PLN). A maior parte das tarefas de PLN são hoje realizadas através da arquitetura de *transformers*, que, desde 2017, possui o estado da arte nessa área, inclusive no PLN aplicado a questões matemáticas (Chollet, 2021).

A utilização de *Deep Learning* para a resolução de problemas matemáticos e simbólicos, apesar de restrita, tem sido foco de artigos, que publicaram importantes avanços na área. O trabalho de Wangperawong (2018), alcançou 84, 90% de acerto na resolução de frases matemáticas, com escopo restrito. Saxton, Grefenstette, Hill, and Kohli (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, chegaram a apenas 6, 9% de acerto. Já Lample and Charton (2019) e Charton, Hayat, and Lample (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.

Nesse contexto, diferente dos trabalhos citados, o objetivo deste trabalho, é a criação de um *step analyzer* 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 cerca de 115 mil expressões matemáticas. O modelo proposto busca atingir bons resultados de acurácia e evitar a dependência de ferramentas externas.

Esse 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. Após, verifica se o passo 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 trabalho visa auxiliar na ampliação do uso de *deep learning* em sistemas matemáticos, tomando como exemplo, a utilização no sistema tutor inteligente *PAT2Math*. Além disso, pretende-se realizar uma análise comparativa entre as diferentes arquiteturas e estratégias utilizadas, de forma a contribuir com o campo de pesquisa dos sistemas especialistas e sistemas tutores inteligentes, além de colaborar para a ampliar a utilização de ferramentas de ensino à distância, que ganharam tanta relevância no mundo, durante o período de pandemia, devido à necessidade rápida de adaptação.

O Capítulo 2 apresenta a fundamentação teórica e os conceitos relativos aos sistemas tutores inteligentes, à *Deep Learning* e às arquiteturas utilizadas. O Capítulo 3 apresenta os trabalhos relacionados. O Capítulo 4 descreve todo o processo de desenvolvimento, desde a coleta, análise e tratamento de dados até o pré-processamento e indicação dos modelos utilizados. No Capítulo 5 são apresentados os resultados obtidos com as seis versões criadas a partir das arquiteturas de redes neurais e *transformers* e o resultado final do trabalho, além de uma análise comparativa. Por fim, o Capítulo 6 finaliza o trabalho, incluindo as conclusões e limitações.

2 FUNDAMENTAÇÃO E CONCEITOS

Este capítulo descreve a fundamentação teórica relacionada aos Sistemas Tutores Inteligentes, seus módulos, funcionamento, sistemas especialistas e a apresentação do *PAT2Math*, sistema tutor inteligente utilizado no desenvolvimento deste trabalho. Além disso, descreve a utilização de *Deep Learning* e Processamento de Linguagem Natural na correção de expressões matemáticas, apresentando o conceito de dois componentes essenciais no desenvolvimento de modelos de aprendizado profundo: as Redes Neurais e os *Transformers*.

2.1 SISTEMAS TUTORES INTELIGENTES

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. Esses sistemas possuem módulos que representam o conhecimento especialista e o conhecimento dos estudantes, utilizando estratégias de tutoria, de forma a se adequarem às necessidades individuais do estudante (Woolf, 2007).

O primeiro tutor inteligente foi implementado na tese de doutorado de Carbonell (1970), que desenvolveu o SCHOLAR, um sistema que convidava estudantes a explorar características geográficas da América do Sul. Já o primeiro sistema tutor inteligente, baseado em um sistema especialista, foi o GUIDON, desenvolvido por Clancey (1979), que era voltado à área médica (Woolf, 2007).

Alguns resultados da efetividade dos sistemas tutores inteligentes são relatados por VanLehn (2011). O seu artigo, indica um aumento de 0,79 unidades, no desempenho dos estudantes com o auxílio de tutoria humana, enquanto os tutores inteligentes aumentam a pontuação dos alunos, em média, em 0,76 unidades, para sistemas baseados em passos. Tais resultados indicam que os sistemas tutores inteligentes podem alcançar um nível de aprendizado muito próximo ao obtido através de tutores humanos.

2.1.1 Módulos

Os tutores inteligentes exigem grandes quantidades de conhecimento codificado, armazenando informações sobre o domínio, sobre o aluno e sobre o método de ensino. Esses tipos de conhecimento são conceitualmente separados como módulos. A maioria dos tutores inteligentes passa de um módulo de aprendizagem para outro em um processo de integração que pode acontecer várias vezes antes que a resposta do tutor seja produzida (Woolf, 2007).

- **Módulo do Domínio:** representa o conhecimento especialista em um determinado domínio. Pode representar os fatos, procedimentos ou métodos que especialistas usam para realizar tarefas ou resolver problemas (Woolf, 2007).
- **Módulo do Aluno:** representa o aprendizado dos alunos sobre o domínio e descreve como raciocinar sobre o conhecimento deles. Contém tanto o conhecimento original e habilidades típicas do aluno sobre o domínio, quanto as informações sobre o desempenho, como possíveis equívocos, tempo gasto em problemas, dicas solicitadas, respostas corretas e estilo de aprendizado preferido (Woolf, 2007).
- **Módulo de Tutoria:** representa estratégias de ensino e inclui métodos para codificar o raciocínio sobre o *feedback*. Pode ser derivado da observação empírica de professores

orientados por teorias de aprendizagem, ou possibilitada pela tecnologia, portanto, apenas fracamente relacionada a um análogo humano (Woolf, 2007).

- **Módulo de Comunicação:** representa métodos de comunicação entre alunos e computadores (interfaces gráficas, agentes animados ou mecanismos de diálogo). Isso inclui gerenciar a comunicação, discutir o raciocínio do aluno, desenhar gráficos para ilustrar um ponto, mostrar ou detectar emoções e explicar como as conclusões foram alcançadas (Woolf, 2007).

2.1.2 Funcionamento

O funcionamento dos Sistemas Tutores Inteligentes pode ser baseado em passos (*Step-Based*) ou oferecer suporte apenas para a resposta final (*Answer-Based*). No caso dos tutores *step-based*, eles fornecem assistência para cada um dos passos do problema que está sendo resolvido pelo aluno, através de *feedbacks* e dicas. Já os demais tutores inteligentes oferecem *feedbacks* apenas para a resposta final, e fornecem dicas gerais para chegar até ela (VanLehn, 2011).

Os Sistemas Tutores Inteligentes possuem dois *loops*. O *outer loop* ou *loop* externo, executado uma vez para cada tarefa e o *inner loop* ou *loop* interno, executado uma vez para cada passo realizado pelo aluno na solução de uma tarefa.

O *Outer Loop* é responsável por decidir qual tarefa o aluno deverá fazer em seguida e apresentar a ele essa tarefa. Para isso, necessita de um vasto conjunto de tarefas que permita selecioná-las de forma inteligente (VanLehn, 2006). Existem quatro métodos comuns para a seleção de tarefas:

- O aluno seleciona a tarefa em um menu com todas as tarefas.
- O tutor atribui tarefas em uma sequência predeterminada.
- O tutor atribui tarefas a partir de um conjunto de tarefas de uma unidade até que o aluno domine o conhecimento ensinado pela unidade (*Mastery Learning*).
- O tutor acompanha os traços, como estilos de aprendizagem e componentes de conhecimento corretos e incorretos. Ele escolhe uma tarefa com base na correspondência entre as características da tarefa e do aluno (*Macroadaptative Learning*).

Enquanto o *Outer Loop* é relacionado às tarefas, o *Inner Loop* refere-se aos passos dentro de uma tarefa (VanLehn, 2006). De forma geral, os Sistemas Tutores Inteligentes fornecem os seguintes tipos de assistência na resolução de problemas:

- ***Feedback* Mínimo**: informa se o passo está correto ou incorreto.
- ***Feedback* específico de erro**: informa ao aluno a ocorrência de um erro e como corrigí-lo.
- **Dicas**: ajuda o estudante a prosseguir para o próximo passo ou chegar à resposta final. As dicas podem ser solicitadas manualmente pelo aluno, ou também fornecidas pelo sistema de forma automática.
- **Avaliação do conhecimento**: determina quais são os conhecimentos do aluno com base em sua utilização do sistema;
- **Revisão da solução**: informa ao aluno se a sua resposta final está certa ou errada.

2.1.3 PAT2Math

O PAT2Math é um Sistema Tutor Inteligente, baseado na *web*, que auxilia os alunos na resolução de equações de 1º grau. Ele é um sistema tutor baseado em passos (*step-based*), que permite que os alunos submetam cada passo da solução das equações apresentadas. O módulo de domínio é composto por um sistema especialista baseado em regras, responsável por corrigir as equações e verificar se os passos submetidos pelos alunos estão corretos (Jaques et al., 2013).

O sistema especialista, cuja arquitetura está disposta na Figura 2.1, é composto pelos módulos *step analyzer* (SA) e *step generator* (SG) (Jaques et al., 2013). Conforme estrutura descrita por VanLehn (2006), o SG é responsável por gerar o próximo passo e, também, por fornecer dicas ao aluno. O SA analisa e classifica os passos realizados pelo estudante. Se o passo está correto, o SA pode retornar os componentes de conhecimento que o aluno provavelmente usou para derivar o passo. Caso esteja incorreto, o SA pode retornar a descrição do erro cometido pelo estudante.

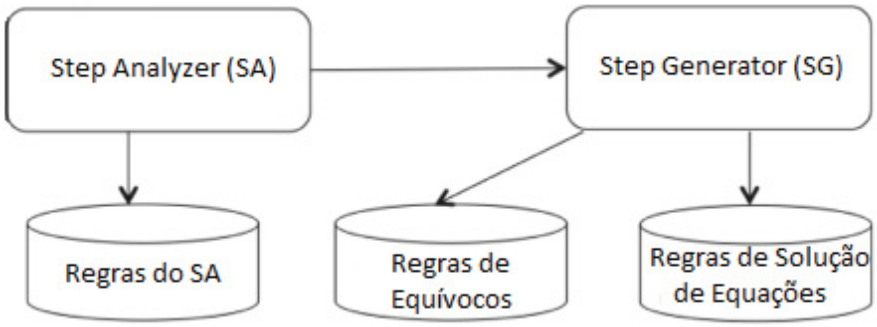


Figura 2.1: Arquitetura do sistema especialista do PAT2Math

Traduzido de Jaques et al. (2013)

A técnica presente no SA, que verifica qual conhecimento utilizado pelo aluno na resolução do passo, é conhecida como *Model Tracing* (Anderson, Corbett, Koedinger, & Pelletier, 1995). O *Model Tracing* assume que as etapas podem ser identificadas e explicitamente codificadas. Ele compara as ações dos alunos com a execução realizada pelo modelo de domínio, testando todas as regras possíveis até que um resultado equivalente seja obtido (Woolf, 2007). Os *designs* do *step analyzer* e do *step generator* variam consideravelmente de um tutor inteligente para outro.

O PAT2Math possui um editor de álgebra, que através de sua interface gráfica, exibe equações e permite que o estudante resolva os problemas apresentados, passo-a passo, emitindo *feedbacks* sobre o desempenho e a ocorrência de erros e permitindo a solicitação de dicas (Jaques et al., 2013). A Figura 2.2 apresenta a interface do PAT2Math, conforme segue: a) lista para seleção de equações; b) pontuação atribuída ao aluno de acordo com seu desempenho; c) equação exibida na tela, possibilita que o aluno digite os próximos passos para a resolução; d) *feedback* obtido para cada uma dos passos respondidos corretamente; e) barra de progresso do nível e botão para solicitação de dica; f) botão que disponibiliza a tabela de classificação dos estudantes.

A interface principal do PAT2Math (folha de caderno) representa o *loop* interno do STI, já que é nesse componente do PAT2Math que o estudante fornece os passos da equação e recebe dicas e *feedbacks* para esses passos. Além disso, no servidor, possui um sistema especialista que resolve equações, corrige os passos efetuados pelos alunos, avalia a sua evolução e atualiza os modelos relacionados ao conhecimento e erros, mantendo um histórico da resolução dos passos (Jaques et al., 2013; VanLehn, 2006). O *outer loop* do PAT2Math, representado pelo menu à

esquerda da interface gráfica, consiste de uma sequência de planos de tarefas a serem resolvidas pelo aluno.

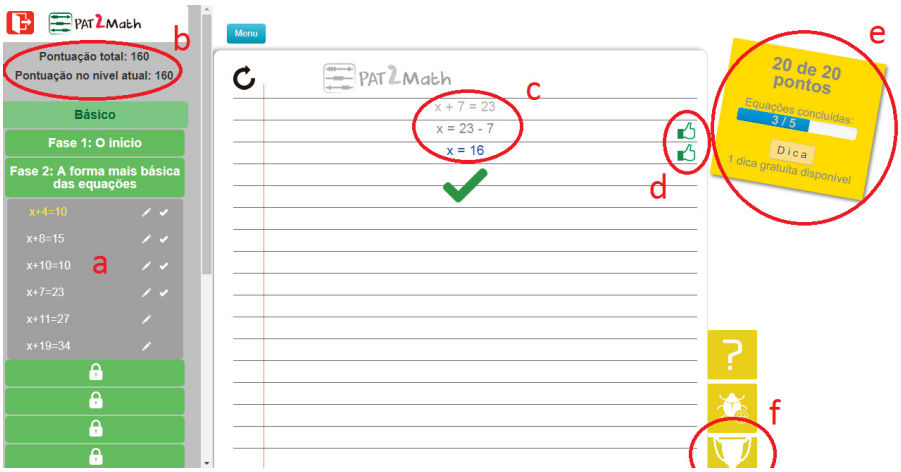


Figura 2.2: Interface do PAT2Math

Elaborado pelo autor

2.2 DEEP LEARNING E PROCESSAMENTO DE LINGUAGEM NATURAL

Os dados das interações dos estudantes junto ao sistema tutor *PAT2Math*, descrito na Seção 2.1 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.

2.2.1 Deep Learning

Deep Learning, ou aprendizado profundo, é um subcampo específico de *machine learning*, que por sua vez, é um subcampo da Inteligência Artificial. Um sistema de aprendizado profundo é treinado ao invés de programado e busca aprender representações/padrões a partir de dados, que eventualmente permitem ao sistema criar regras para automatizar determinada tarefa. A partir da entrada de dados e respostas (*labels*), o programa retorna as regras ou modelo, que podem ser aplicados a novos dados, diferente do que ocorre nos programas clássicos, onde são inseridos dados e regras, para obter respostas. A figura 2.3 exemplifica essa diferença (Chollet, 2021).

O nome *Deep Learning* refere-se à ideia de camadas sucessivas de redes neurais (Chollet, 2021). 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).

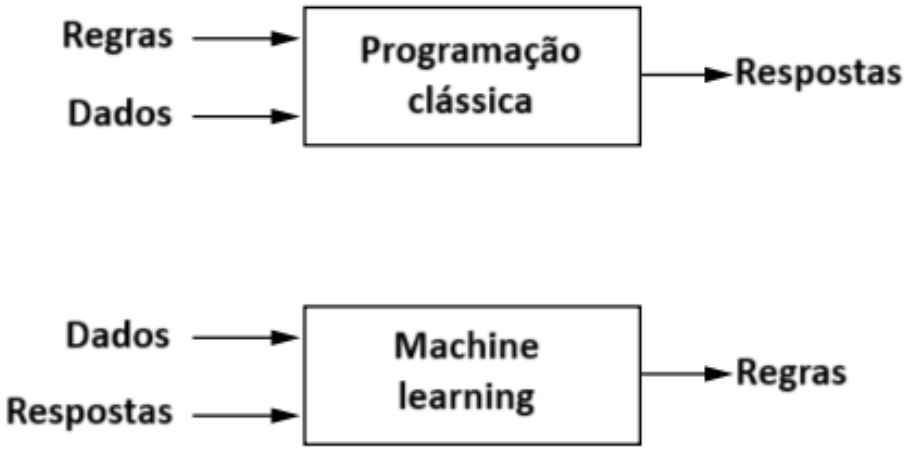


Figura 2.3: Diferença entre programas convencionais e deep learning

Traduzido de Chollet (2021)

2.2.2 Processamento de Linguagem Natural

O Processamento de Linguagem Natural (PLN) é uma subárea da inteligência artificial (IA), que trata do processamento de linguagens naturais, tais como o português ou o inglês. Esse processamento geralmente envolve a tradução da linguagem natural para dados/números, para que um computador possa interpretar e/ou gerar essas linguagens (Lane, Hapke, & Howard, 2019).

Em computação, as linguagens humanas são chamadas de naturais, para distingui-las das linguagens projetadas para máquinas, como *Assembly*, *XML* ou *Python*. Enquanto as linguagens de máquina são projetadas, a partir da criação de regras formais, com as linguagens naturais ocorre o oposto: o uso ocorre antes do surgimento das regras (Chollet, 2021).

O processamento de linguagem natural é utilizado nas mais diversas tarefas, como análise e classificação de texto, extração de informação, tradução automática e geração de texto. Além dessas, o PLN também é utilizado na resolução de tarefas na área matemática e simbólica.

2.3 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).

Alguns dos trabalhos mais antigos de Inteligência Artificial, que datam da década de 40, tiveram o objetivo de criar redes neurais artificiais. Um exemplo é o modelo matemático simples de neurônio desenvolvido por McCulloch and Pitts (1943). Seu artigo descreve como os neurônios devem funcionar e implementam uma rede neural simples, com um classificador linear (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, como processamento de linguagem natural.

A Figura 2.4 mostra a linha do tempo de diferentes mecanismos e arquiteturas utilizadas para o processamento de linguagem natural, que são conceituadas nas Subseções 2.3.1, 2.3.2, 2.3.3 e Seção 2.4.

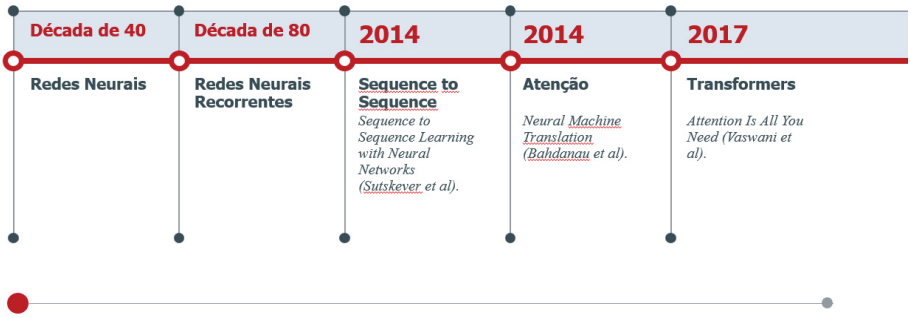


Figura 2.4: Linha do tempo da evolução no processamento de linguagem natural

Elaborado pelo autor

2.3.1 Redes Neurais Recorrentes

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 os exemplos da utilização das RNRs estão o reconhecimento de fala, a tradução automática de linguagens, a legendagem de imagens, dentre outros (Zhang, Lipton, Li, & Smola, 2020).

Apesar da sua utilidade, as redes neurais recorrentes mais simples possuem alguns problemas, como o *Vanishing Gradient Problem*. Nesse problema, a medida que mais camadas são adicionadas às redes neurais, os gradientes da função de perda se aproximam de zero, tornando a rede difícil de treinar. Isso significa que a rede tem dificuldade em memorizar palavras distantes na sequência e faz previsões com base apenas nas palavras mais recentes (Zhang et al., 2020).

Por esse motivo, existem alguns modelos mais poderosos como LSTM e GRU, que conseguem resolver essas limitações. 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 and Schmidhuber (1997) e foram refinadas e popularizadas nos anos seguintes. Essas redes funcionam bem em uma grande variedade de problemas e são amplamente utilizadas. As GRU (*Gated Recorrent Unit*), introduzidas por Chung, Gulcehre, Cho, and Bengio (2014), são redes similares às LSTM, mas possuem um único estado oculto, para manter as dependências de longo e curto prazo, ao mesmo tempo.

2.3.2 Sequence-to-Sequence (Seq2seq)

Introduzido em 2014 pelo *Google*, através do artigo *Sequence to Sequence Learning with Neural Networks*, o modelo *Sequence-to-Sequence* visa mapear uma entrada de comprimento fixo com uma saída de comprimento fixo em que o comprimento da entrada e da saída podem ser diferentes. É utilizado em diversas aplicações, como tradução, legendas, reconhecimento de fala e *chatbots*.

Esse modelo pode ser usado como uma solução para qualquer problema baseado em sequência, especialmente aqueles em que as entradas e saídas possuem tamanhos e categorias distintos (Sutskever, Vinyals, & Le, 2014).

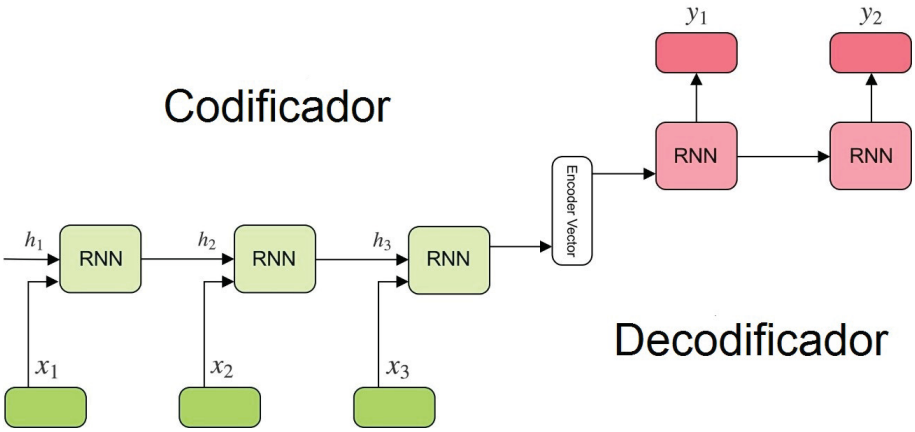


Figura 2.5: Arquitetura sequence-to-sequence

Traduzido de Kostadinov (2020)

Conforme mostrado na Figura 2.5, o modelo consiste em três partes: codificador (*encoder*), vetor intermediário e decodificador (*decoder*). O modelo transforma uma sequência em outra sequência utilizando uma rede neural recorrente ou, mais frequentemente, LSTM ou GRU. O codificador transforma cada item em um vetor oculto correspondente, contendo o item e seu contexto. O decodificador reverte o processo, transformando o vetor em um item de saída, usando a saída anterior como contexto de entrada (Kostadinov, 2020).

2.3.3 Atenção (Attention)

Na arquitetura do codificador-decodificador, a sequência completa de informações deve ser capturada por um único vetor. Isso gera problemas para manter as informações no início da sequência e codificar dependências de longo alcance. A ideia central da Atenção é focar nas partes mais relevantes da sequência de entrada para cada saída. Ao fornecer um caminho direto para as entradas, a atenção também ajuda a reduzir o *Vanishing Gradient Problem*, comentado na Seção 2.3.1.

Atenção foi proposta pela primeira vez por Bahdanau, Cho, and Bengio (2014) para tradução automática. O mecanismo 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: i) Atenção Codificador-Decodificador: Atenção entre a sequência de entrada e a sequência de saída; ii) *Self-Attention* na sequência de entrada: atende a todas as palavras na sequência de entrada; iii) *Self-Attention* na sequência de saída (Luong, Pham, & Manning, 2015).

A Figura 2.6 demonstra o mecanismo da atenção aplicada à tradução automática, tendo como entrada uma frase em português e como saída, a tradução, em francês. O codificador é apresentado em azul, contendo a frase de entrada. Os pesos, mostrados acima do codificador, definem a importância das palavras para a próxima palavra prevista.

O decodificador, em vermelho, recebe a saída completa do codificador, usa a rede recorrente para rastrear o que já foi gerado e consulta a atenção na saída do codificador, para produzir o vetor de contexto. Após, ele combina a saída da rede recorrente e o vetor de contexto, para gerar o vetor de atenção, em dourado, além de fazer previsões para o próximo *token*, com base no vetor de atenção (Luong et al., 2015).

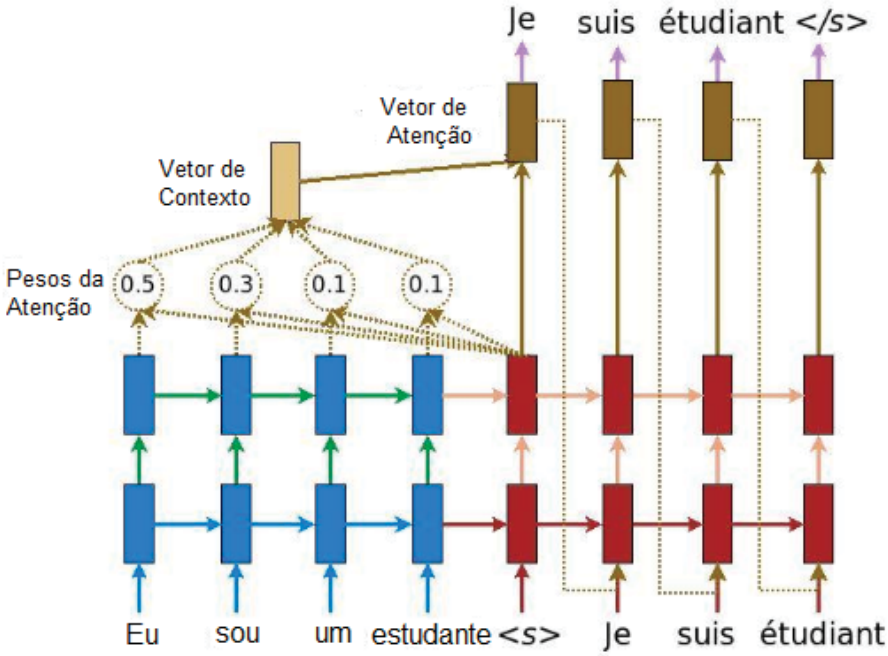


Figura 2.6: Mecanismo de atenção para o problema de tradução

Traduzido de TensorFlow (2020)

2.4 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 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., 2017).

Sua arquitetura provém diversas vantagens: 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., 2017).

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., 2017). A arquitetura completa do modelo pode ser verificada na Figura 2.7.

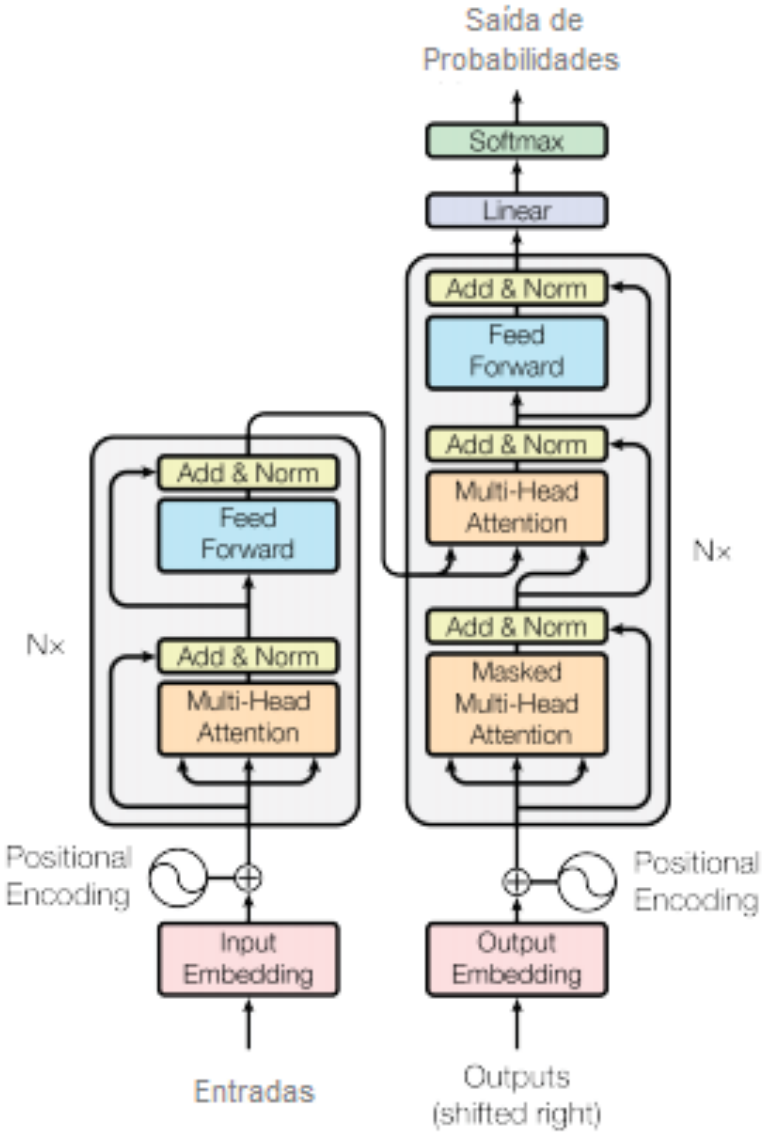


Figura 2.7: Arquitetura transformer

Traduzido de Vaswani et al. (2017)

Dentre os modelos mais recentes de *transformers*, pode-se citar GPT-2 (*Generative Pre-training for Transformer*) (Alammar, 2020) e GPT-3 (Brown et al., 2020), da *Open AI*, que atingiram resultados comparáveis a humanos na geração de texto. O T5, (*Text-to-Text Transfer Transformer*)(Raffel et al., 2019), criado pelo *Google*, que é um *transformer* multitarefa, que pode realizar várias tarefas distintas como tradução, resposta automática e classificação, tudo através do mesmo modelo. Finalmente, o BERT, (*Bidirectional Encoder Representations from*

Transformers) (Devlin, Chang, Lee, & Toutanova, 2018) é outro *transformer* famoso, usado para aprender representações de texto.

Abaixo, as Subseções 2.4.1, 2.4.2, 2.4.3 e 2.4.4 descrevem alguns dos principais componentes da arquitetura *transformer* e a Subseção 2.4.5, descreve mais detalhes do modelo de transformer utilizado neste trabalho.

2.4.1 Positional encoding

A arquitetura do *transformer* usa camadas de atenção empilhadas no lugar de RNCs ou RNRs. Isso facilita o aprendizado de dependências de longo alcance, mas não contém informações internas sobre as posições relativas dos itens em uma sequência. Dessa forma, a codificação posicional (*positional encoding*) é adicionada para fornecer ao modelo algumas informações sobre a posição relativa das palavras na frase (Vaswani et al., 2017).

O vetor de codificação posicional é adicionado ao vetor de *embeddings*. Os *embeddings* representam um token em um espaço d-dimensional onde os tokens com significados semelhantes estarão mais próximos uns dos outros, mas os *embeddings* não codificam a posição relativa das palavras em uma frase. Portanto, após adicionar a codificação posicional, as palavras ficarão mais próximas umas das outras com base na semelhança de seu significado e sua posição na frase, no espaço d-dimensional (TensorFlow, 2020).

2.4.2 Masking

Durante o tratamento das entradas, são acrescentados às palavras tokens de *padding* ou preenchimento, visando sua padronização. Para garantir que o modelo não trate os tokens de *padding* como entrada, é necessário mascarar todos esses tokens no *batch* (lote) da sequência. A máscara indica onde o valor de *padding* 0 está presente e produz 1 nesses locais e 0, caso contrário.

A máscara de antecipação é usada para mascarar os tokens futuros em uma sequência. Em outras palavras, a máscara indica quais entradas não devem ser usadas. Isso significa que, para prever a terceira palavra, apenas a primeira e a segunda palavras serão usadas. Da mesma forma, para prever a quarta palavra, apenas a primeira, a segunda e a terceira palavras serão usadas e assim por diante (TensorFlow, 2020).

2.4.3 Scaled dot product attention

A partir de uma sequência de entrada, a atenção calcula os pesos das palavras, conforme sua importância, através de uma função de atenção. A função de atenção usada pelo *transformer* tem três entradas: Q (*querie*/consulta), K (*key*/chave), V (*value*/valor). A equação usada para calcular os pesos de atenção é:

$$Attention(Q, K, V) = softmax_k(\frac{QK^T}{\sqrt{d_k}})V$$

A atenção do produto escalar (*scaled dot product attention*) é dimensionada por um fator de raiz quadrada da profundidade. Isso é feito, porque para grandes valores de profundidade, o produto escalar cresce em magnitude, empurrando a função *softmax* onde ela tenha pequenos gradientes, resultando em um *softmax* muito difícil (TensorFlow, 2020).

Por exemplo, considerando que Q e K tenham média 0 e variância 1, a multiplicação de matrizes terá média 0 e variância dk. Portanto, a raiz quadrada de dk é usada para escalar porque o *MatMul* de Q e K deve ter uma média de 0 e variância de 1, e com isso, se obtém um *softmax* mais suave (TensorFlow, 2020).

A máscara é multiplicada por $-1e9$ (próximo ao infinito negativo). Isso é feito porque a máscara é somada com a multiplicação da matriz em escala de Q e K e é aplicada imediatamente antes de um *softmax*. O objetivo é zerar essas células, e grandes entradas negativas para *softmax* que estão perto de zero na saída (TensorFlow, 2020).

Como a normalização *softmax* é feita em K, seus valores decidem a importância dada a Q. A saída representa a multiplicação dos pesos de atenção e o vetor V (valor). Isso garante que as palavras mais importantes sejam mantidas como estão e que as palavras irrelevantes sejam eliminadas (TensorFlow, 2020). A Figura 2.8 demonstra o esquema utilizado.

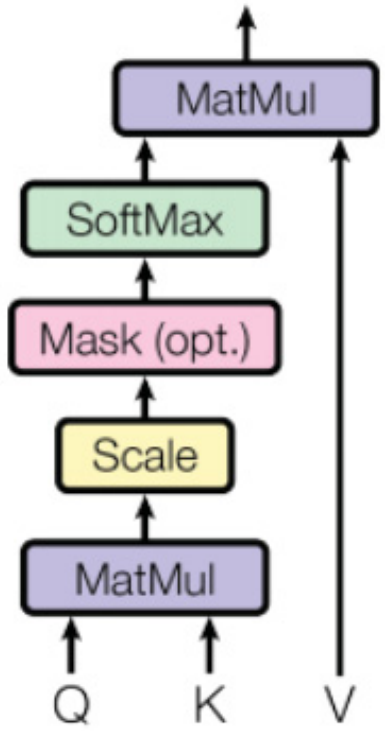


Figura 2.8: Scaled dot product attention

Retirado de TensorFlow (2020)

2.4.4 Multi-head attention

A *Multi-head attention* consiste em quatro partes, conforme mostrado na Figura 2.9:

- Camadas lineares e divisão em *heads*/cabeças.
- *Scaled dot-product attention*.
- Concatenação de *heads*/cabeças.
- Camada linear final

Em vez de uma única *head*/cabeça de atenção, Q, K e V são divididos em várias cabeças, permitindo que o modelo atenda conjuntamente a informações em diferentes posições de diferentes espaços representacionais. Após a divisão, cada cabeça tem uma dimensionalidade reduzida, então a computação pode ser paralelizada e o custo total é o mesmo que a atenção de uma única cabeça, com dimensionalidade total (DeepLearning.ai, 2020).

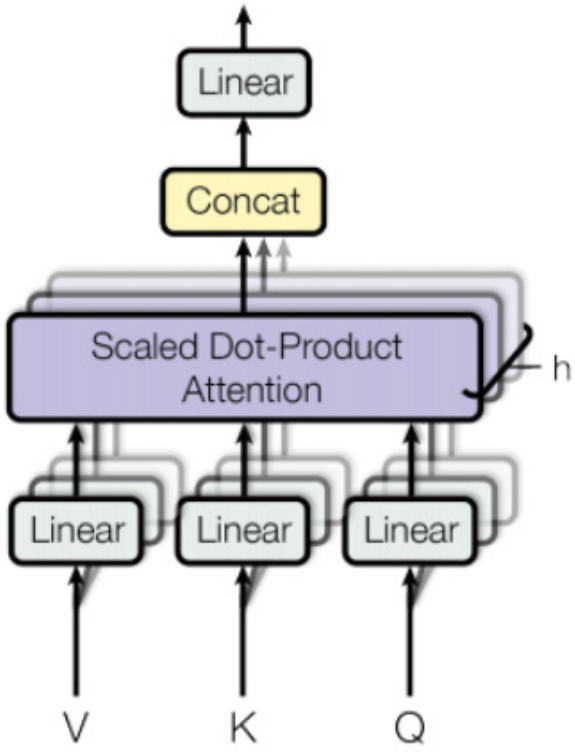


Figura 2.9: Multi-head attention

Retirado de TensorFlow (2020)

2.4.5 Bidirectional Encoder Representations from Transformers (BERT)

O *Bidirectional Encoder Representations from Transformers* (BERT) é um modelo de linguagem pré-treinado baseado na arquitetura *Transformer*. Ele foi desenvolvido pelo *Google AI Language*, em 2018, e desde então é aplicada em uma grande variedade de tarefas relacionadas ao processamento de linguagem natural (Devlin et al., 2018).

O BERT passa por um processo de pré-treinamento em grandes quantidades de texto, no qual o modelo aprende a prever palavras faltantes em sentenças, de forma bidirecional, ou seja, considerando tanto o contexto anterior quanto o posterior a cada palavra. Isso permite que o BERT desenvolva uma compreensão profunda e contextual da linguagem (Devlin et al., 2018).

Dentre as vantagens do BERT, está a capacidade de transferir o conhecimento adquirido durante o pré-treinamento para tarefas específicas com relativamente poucos dados rotulados, o que é importante em situações onde a coleta de grandes conjuntos de dados rotulados é dispendiosa ou impraticável (Devlin et al., 2018).

O BERT possui diversas variações, tais como: RoBERTa, desenvolvido pelo *Facebook AI*; XLNet; DistilBERT, que é uma versão compacta do BERT, desenvolvida pela *Hugging Face*; BioBERT; dentre outras.

2.5 CONCLUSÃO

Este capítulo descreveu a fundamentação teórica do trabalho aqui desenvolvido. A Seção 2.1 apresentou o conceito dos sistemas tutores inteligentes, seus módulos, funcionamento e o *PAT2Math*. Em seguida, na Seção 2.2, foram apresentadas as definições das áreas relacionadas à solução proposta. Foram descritos os conceitos de *Deep Learning* e processamento de linguagem

natural. Por fim, nas Seções 2.3 e 2.4, foram apresentadas as arquiteturas de Redes Neurais e *Transformers*, finalizando com a apresentação do modelo BERT.

Os conceitos apresentados neste capítulo, serão retomados nos capítulos seguintes, durante a descrição dos trabalhos relacionados, método, avaliação e resultados, nos Capítulos 3, 4 e 5 respectivamente.

3 TRABALHOS RELACIONADOS

Este capítulo descreve os trabalhos relacionados, nos quais a *Deep Learning* é utilizada para a resolução de problemas matemáticos, de diferentes tipos e complexidades.

A pesquisa foi realizada através do *Google Scholar*, filtrando os últimos cinco anos e selecionando trabalhos com as palavras-chave ***Deep Learning*** e ***Mathematics***, selecionando os artigos mais próximos ao objeto desta pesquisa, no caso, a utilização de aprendizagem profunda para a resolução de problemas matemáticos. Os trabalhos estão listados abaixo, em ordem cronológica.

3.1 ANALYSING MATHEMATICAL REASONING ABILITIES OF NEURAL MODELS (SAXTON, HILL, 2019)

No artigo de Saxton et al. (2019), são apresentados conjuntos de problemas matemáticos envolvendo perguntas e respostas sequenciais em um formato de texto livre. Esse conjunto é avaliado através de redes neurais, por duas classes de arquiteturas *Sequence-to-Sequence*, visando resolver os problemas matemáticos e generalizar seus conhecimentos.

O trabalho apresenta um conjunto de dados gerado sinteticamente, contendo questões envolvendo um ou mais objetos matemáticos de diversos grupos. No total, são dois milhões de pares pergunta-resposta, nas seguintes categorias: álgebra (equações lineares, raízes polinomiais, sequências); aritmética (operações em pares e expressões mistas); cálculo (diferenciação); comparação (números mais próximos, comparações de pares, classificação); medição (conversão, trabalhando com tempo); números (conversão de base, restos, divisores e múltiplos comuns, primalidade, valor posicional, números de arredondamento); polinômios (adição, simplificação, composição, avaliação, expansão) e probabilidade (amostragem sem reposição).

O conjunto de dados está disponível em http://github.com/deepmind/mathematics_dataset. Um exemplo de questão desse conjunto, seria descrita como: $Calculate - 841880142.544 + 411127$ e a resposta, como: -841469015.544.

Além disso, são disponibilizados dois conjuntos de testes distintos: os testes de interpolação, um para cada tipo de questão ocorrida no conjunto de treinamento; e testes de extrapolação, que medem a generalização ao longo de vários níveis de dificuldade, para além do que é visto durante a etapa de treinamento.

Para avaliação, os autores examinaram dois modelos distintos para solução de problemas *Sequence-to-Sequence*: Arquiteturas neurais recorrentes e a Arquitetura Atencional/Transformers, conforme descrita por Vaswani et al. (2017).

Na primeira arquitetura, foram analisados os modelos de LSTM simples, LSTM de atenção, de Bahdanau et al. (2014), que tem sido predominante em sistemas neurais de tradução automática, e *Relational Memory Core* (RMC), descrita por Santoro, Hill, Barrett, Morcos, and Lillicrap (2018), que possui vários slots de memória que interagem por meio de atenção. Já a segunda arquitetura utilizada foi o modelo *Transformer*, descrito por Vaswani et al. (2017), que possui ótima performance em tradução automática. Os parâmetros do *transfomer* foram ajustados para um *d-model* de 512, 8 cabeças/*heads* atencionais e *DFF* de 2048.

Na Figura 3.1 são apresentados os percentuais de acurácia na utilização dos modelos descritos anteriormente. O *Transformer* teve desempenho significativamente melhor do que os modelos recorrentes, tanto nos testes de interpolação, quanto nos testes de extrapolação.

	Parameters	Interpolation	Extrapolation
Simple **LSTM**	18M	0.57	0.41
Simple **RMC**	38M	0.53	0.38
Attentional **LSTM**, LSTM encoder	24M	0.57	0.38
Attentional **LSTM**, bidir LSTM encoder	26M	0.58	0.42
Attentional **RMC**, bidir LSTM encoder	39M	0.54	0.43
Transformer	30M	**0.76**	**0.50**

Figura 3.1: Comparação entre as diferentes arquiteturas

Retirado de Saxton et al. (2019)

Por fim, o modelo *Transformer*, que obteve os melhores resultados, foi testado em um conjunto de 40 perguntas selecionadas a partir de exames de matemática para alunos britânicos de 16 anos. Nessas questões, o modelo acertou 14 das 40 questões, o que é equivalente a um aluno com grade E.

Em suma, os autores criaram um conjunto de dados com questões matemáticas em formato textual, no qual a arquitetura de *transformers* obteve um desempenho moderado, mas ainda superior ao das redes neurais.

3.2 ATTENDING TO MATHEMATICAL LANGUAGE WITH TRANSFORMERS (WANGPERAWONG, 2019)

No artigo de Wangperawong (2018), o autor treina modelos de redes neurais para efetuarem a leitura de frases matemáticas e realizarem a avaliação dos resultados. 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.

No estudo, todos os alvos considerados, são números decimais, representados no nível de caractere. Cerca de 12 milhões de exemplos foram gerados e divididos randomicamente entre conjuntos de treinamento e de teste, na proporção aproximada de $9:1$. Cada entrada é lida e codificada para o nível de caractere. Da mesma forma, cada saída é decodificada do nível de caractere. A Figura 3.2 mostra o esquema de codificação utilizado nos valores de entrada e no resultado obtido.

Exemplo de Entrada

x	$=$	8	5	,	y	$=$	$-$	5	2	3	,	x	$\times$	y
g	r	f	d	n	p	r	w	d	q	e	n	g	k	p

Exemplo de Saída

$-$	4	4	4	5	5
w	m	m	m	d	d

Figura 3.2: Codificação de entrada e saída

Traduzido de Wangperawong (2018)

Os pares de entrada e saída são primeiramente usados para treinar um modelo de *transformer* sem recorrência, conforme descrito por Vaswani et al. (2017). Os mesmos hiperparâmetros da versão padrão dos *transformers* foram utilizados, exceto por alguns detalhes. 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, Gouws, Vinyals, Uszkoreit, and Kaiser (2018), que pode ser computacionalmente universal, se tiver memória suficiente. Esse *transformer* alcançou o estado da arte na tradução automática, superando LSTMs e o *transformer* padrão, com os mesmos hiperparâmetros, 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%.

Os resultados descritos no artigo, também foram reproduzidos no *Github*, através do *link* https://github.com/artitw/tensor2tensor.

3.3 DEEP LEARNING FOR SYMBOLIC MATHEMATICS (LAMPLE, CHARTON; 2020)

O trabalho de Lample and Charton (2019), pesquisadores do *Facebook*, 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.

As expressões matemáticas podem ser representadas como árvores, com operadores e funções como nós internos, operandos como filhos e números, constantes e variáveis como folhas. No entanto, os modelos de árvore para árvore são mais lentos e complexos do que modelos *Sequence-to-Sequence*.

Para a utilização de modelos *Seq2seq* para gerar árvores é necessário mapear as árvores para sequências. Para esse efeito, o artigo usa a notação prefixada, escrevendo cada nó antes de seus filhos, listados da esquerda para a direita. Por exemplo, a expressão aritmética $2 + 3 * (5 + 2)$ é representada como a sequência [+ 2 * 3 + 5 2].

Para criar os dados de treinamento, foram gerados conjuntos de expressões matemáticas aleatórias, baseados em três diferentes abordagens:

- ***Forward Generation (FWD)*** – Abordagem direta. Gera aleatoriamente uma função simbólica f e, então, usa uma ferramenta externa para computar a integral simbólica F.
- ***Backward generation (BWD)*** – Utiliza o fato de que é fácil derivar, mas difícil integrar uma função. Dessa forma, gera aleatoriamente uma função F e, então, automaticamente deriva e recebe a função f.
- ***Integration by parts (IBP)*** - Este método faz uso de uma propriedade matemática de integrais sobre funções que são multiplicadas juntas.

Para os experimentos, os autores treinaram um modelo *Seq2seq* para prever as soluções de determinados problemas, ou seja, para prever uma primitiva dada uma função, ou prever uma solução dada uma equação diferencial. O modelo utilizado no artigo é o modelo de *Transformers*, descrito por Vaswani et al. (2017), com 8 cabeças de atenção, 6 camadas e uma dimensionalidade de 512. Nos experimentos, o uso de modelos maiores não melhorou o desempenho. Os modelos foram treinados com o otimizador Adam. Na inferência, as expressões são geradas por *beam search*, com larguras de 1, 10 e 50.

O algoritmo *Beam Search* seleciona várias alternativas para uma sequência de entrada em cada etapa, com base na probabilidade condicional. A cada intervalo de tempo, o algoritmo

seleciona um determinado número de alternativas que possuem maior probabilidade de serem a solução correta (Khandelwal, 2020).

Após cada época de treinamento, foi avaliada a capacidade do modelo de prever as soluções de determinadas equações. As hipóteses podem ser verificadas, simplesmente comparando as expressões geradas com suas soluções de referência. Dada a facilidade dessa verificação, todas as hipóteses do *beam* foram consideradas, e não apenas aquelas com maior pontuação. Foi considerado que o modelo resolveu com sucesso a equação de entrada se, ao menos, uma delas estiver correta. Assim, resultados com tamanho de *beam* 10 indicam que pelo menos uma das 10 hipóteses no *beam* estava correta.

Na Figura 3.3 são apresentados os resultados obtidos na avaliação do modelo, utilizando os dados de treinamento gerados de apenas uma forma (FWD, BWD ou IBP) e avaliando o modelo nos dados gerados pelo mesmo método. Os resultados atingidos ficaram próximos de 100, 0% para integração e chegaram a 94, 0% e 91, 2% para equações diferenciais de primeira e segunda ordem, respectivamente.

	Integration (FWD)	Integration (BWD)	Integration (IBP)	ODE (order 1)	ODE (order 2)
Beam size 1	93.6	98.4	96.8	77.6	43.0
Beam size 10	95.6	99.4	99.2	90.5	73.0
Beam size 50	96.2	99.7	99.5	94.0	81.2

Figura 3.3: Comparação de resultados

Retirado de Lample and Charton (2019)

Posteriormente, os autores compararam os resultados do modelo ao resultado obtido pelos softwares comerciais *Mathematica*, *Maple* e *Matlab*. Nesse caso, apenas o gerador BWD foi avaliado. Para avaliação do software *Mathematica*, foi definido um tempo máximo de 30 segundos para que o sistema apresentasse a solução, caso contrário, os autores consideraram que o software não era capaz de realizar o cálculo. Os resultados dessa comparação constam na Figura 3.4. O modelo *Seq2seq* claramente supera os *frameworks* matemáticos em equações geradas pela abordagem BWD.

	Integration (BWD)	ODE (order 1)	ODE (order 2)
Mathematica (30s)	84.0	77.2	61.6
Matlab	65.2	-	-
Maple	67.4	-	-
Beam size 1	98.4	81.2	40.8
Beam size 10	99.6	94.0	73.2
Beam size 50	99.6	97.0	81.0

Figura 3.4: Comparação com frameworks comerciais

Retirado de Lample and Charton (2019)

Resumindo, os autores demonstraram que a abordagem utilizada pode funcionar até mesmo para tarefas que requerem uma precisão absoluta. Os resultados obtidos foram surpreendentemente bons e superaram *frameworks* matemáticos populares. A implementação do trabalho está disponível em https://github.com/facebookresearch/SymbolicMathematics.

3.4 ENHANCING THE TRANSFORMER WITH EXPLICIT RELATIONAL ENCODING FOR MATH PROBLEM SOLVING (SCHLAG, SMOLENSKY, FERNANDEZ, JOJIC, SCHMIDHUBER, GAO; 2020)

O trabalho de (Schlag et al., 2019) propõe 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)*, que alcança melhores resultados na resolução de questões matemáticas em texto-livre, no conjunto de dados proposto por (Saxton et al., 2019).

A ideia principal da nova estrutura é capturar relações de função de preenchimento, incorporando um produto *Hadamard* de cada representação de vetor de valor (após a atenção) com um vetor de relação, para cada cabeça de atenção em cada camada.

Da mesma forma que em Saxton et al. (2019), o modelo foi testado com dois conjuntos distintos: testes de interpolação e testes de extrapolação.

	Weights	**Steps**	**Train**	**Interpolation**		**Extrapolation**	
				acc	>95%	acc	>95%
LSTM with thinking steps (Saxton et al.)	18M	500k	-	57.00%	6	41.00%	1
Transformer (Saxton et al.)	30M	500k	-	76.00%	13	50.00%	1
Transformer (ours)	44.2M	1000k	86.60%	79.54%	16	53.28%	2
TP-Transformer (ours)	49.1M	1000k	89.01%	81.92%	18	54.67%	3
TP-Transformer B (ours)	43.0M	1000k	87.53%	80.52%	16	52.04%	1
TP-Transformer C (ours)	30.0M	1000k	86.33%	79.02%	14	54.71%	1

Figura 3.5: Resultados obtidos com o TP-Transformer

Retirado de Schlag et al. (2019)

A Figura 3.5 mostra os resultados obtidos no trabalho. Além dos resultados apresentados em Saxton et al. (2019), são descritos os resultados obtidos com a nova estrutura de *TP-Transformer* e a mesma estrutura com alteração de seus hiperparâmetros, definidas como B e C.

Desta forma, o modelo atingiu 81, 92% de acurácia nos testes de interpolação e 54, 67% de acurácia nos testes de extrapolação, superando os resultados originais. O repositório com o código completo pode ser encontrado em https://github.com/ischlag/TP-Transformer.

3.5 LEARNING ADVANCED MATHEMATICAL COMPUTATIONS FROM EXAMPLES (CHARTON, HAYAT, LAMPLE; 2021)

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. Todos os três problemas foram amplamente pesquisados e possuem muitas aplicações fora da matemática pura.

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 foi treinado com o otimizador *Adam*, com um aprendizado na taxa de 10^{-4}. Os modelos qualitativos (prevendo estabilidade, controlabilidade e existência de soluções) foram treinados por cerca de 12 horas, mas precisões próximas aos valores ótimos foram alcançadas após cerca de 6 horas.

Apesar dos problemas requererem uma combinação de técnicas avançadas, simbólicas e numéricas, que parecem improváveis de serem aprendidas com exemplos, eles foram resolvidos com precisão muito alta, mesmo o modelo sendo totalmente inconsciente da teoria subjacente. O modelo final atingiu mais de 95% de precisão em todas as tarefas qualitativas e entre 65 e 85% em cálculos numéricos.

3.6 MEASURING MATHEMATICAL PROBLEM SOLVING WITH THE MATH DATASET (HENDRYCKS, BURNS, KADAVATH, ARORA, BASART, TANG, SONG, STEINHARDT; 2021)

O artigo de Hendrycks et al. (2021) apresenta um novo 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.

Os problemas foram extraídos de competições de matemática, do ensino médio e classificados em níveis de dificuldade, que varia entre 1 e 5. A Figura 3.6 ilustra um exemplo do conjunto de dados.

MATH Dataset (Ours)

Problem: Tom has a red marble, a green marble, a blue marble, and three identical yellow marbles. How many different groups of two marbles can Tom choose?

Solution: There are two cases here: either Tom chooses two yellow marbles (1 result), or he chooses two marbles of different colors ($\binom{4}{2} = 6$ results). The total number of distinct pairs of marbles Tom can choose is $1 + 6 = \boxed{7}$.

Figura 3.6: Conjunto de dados MATH

Retirado de Hendrycks et al. (2021)

Além do conjunto de dados MATH, o trabalho também apresenta um 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*.

Como as respostas do conjunto *MATH* devem ser geradas, o artigo utiliza os modelos de linguagem autoregressiva *GPT-2* e *GPT-3*, que são modelos decodificadores pré-treinados em textos em linguagem natural. Porém, antes disso, o modelo é treinado através do conjunto *AMPS*.

Os resultados alcançados com ambos os modelos são apresentados na Figura 3.7. Esses resultados são bastante baixos, devido ao nível de dificuldade imposto pelo conjunto de dados *MATH*, sendo que o melhor resultado é obtido pelo modelo *GPT-2*, que atinge 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.

Em suma, os autores concluem que enquanto a maioria das outras tarefas baseadas em texto são resolvidas por enormes *Transformers*, o conjunto *MATH* é notavelmente diferente. A precisão aumenta muito lentamente e, dessa forma, indica a necessidade de novos avanços conceituais e algorítmicos para atingir um forte desempenho. O código e o conjunto de dados *MATH* podem ser encontrados em github.com/hendrycks/math/.

Model	Prealgebra	Algebra	Number Theory	Counting & Probability	Geometry	Intermediate Algebra	Precalculus	Average
GPT-2 (0.1B)	5.2	5.1	5.0	2.8	5.7	6.5	7.3	5.4 (+0%)
GPT-2 (0.3B)	6.7	6.6	5.5	3.8	6.9	6.0	7.1	6.2 (+15%)
GPT-2 (0.7B)	6.9	6.1	5.5	5.1	8.2	5.8	7.7	6.4 (+19%)
GPT-2 (1.5B)	8.3	6.2	4.8	5.4	8.7	6.1	8.8	6.9 (+28%)
GPT-3 (2.7B)	2.8	2.9	3.9	3.6	2.1	2.5	2.6	2.9 (−46%)
GPT-3 (175B)	7.7	6.0	4.4	4.7	3.1	4.4	4.0	5.2 (−4%)

Figura 3.7: Resultados MATH

Retirado de Hendrycks et al. (2021)

3.7 CONCLUSÃO

Este capítulo ilustrou os principais resultados obtidos na área de Processamento de Linguagem Natural aplicado a expressões matemáticas, obtidas dentre os trabalhados relacionados.

A Tabela 3.1 mostra um resumo das informações apresentadas. Estão listados, além do nome do artigo e dos autores, o ano de publicação, o problema a ser resolvido, a estrutura utilizada e os resultados informados.

Numa primeira análise, verifica-se que os trabalhos relacionados procuram resolver problemas com complexidade bastante distinta. Dessa forma, os resultados atingidos servem apenas como indícios da efetividade da aplicação de determinado modelo para a resolução do problema a que se propõe e não para a realização de uma comparação direta entre os resultados atingidos.

Um exemplo de trabalho que atinge uma boa performance, mas num escopo limitado, é o artigo de Wangperawong (2018), que alcança 84, 90% de acerto na resolução de expressões matemáticas.

Na resolução de problemas matemáticos, temos os artigos baseados no conjunto de dados de Saxton et al. (2019), *Analysing Mathematical Reasoning Abilities of Neural Networks* e *Enhancing the Transformer with Explicit Relational*, que atingem 76 e 82% de acurácia, respectivamente. Tratando também de problemas matemáticos, temos ainda o artigo *Measuring Mathematical Problem Solving with the MATH Dataset*, que chega a apenas 6, 9% de acerto, mas num conjunto de dados bem mais complexo.

Por fim, temos os artigos Lample and Charton (2019) e Charton et al. (2020), que conseguem atingir uma ótima performance em questões com maior nível de complexidade, sendo 99, 7 e 95% de acurácia, respectivamente.

Em praticamente todos os trabalhos aqui listados, os resultados finais indicam que os melhores desempenhos são verificados nos modelos que utilizam *transformers*, indiferente das configurações utilizadas.

Este trabalho visa atingir percentuais o mais próximos possíveis ao Estado da Arte apresentado nos resultados da Tabela 3.1, considerando margem para as particularidades e diferenças com relação aos trabalhos relacionados. Serão utilizadas versões baseadas na arquiteturas de redes neurais GRU e *transformers*, visando a criação de um *step analyzer* e a automatização do módulo de domínio de um sistema tutor inteligente baseado em passos, evitando a dependência de ferramentas externas.

Tabela 3.1: Tabela comparativa dos trabalhos relacionados

Artigo	Ano	Autores	Problema	Estrutura	Resultados
Analysing Mathematical Reasoning Abilities of Neural Models	2019	Saxton, Hill	Problemas Matemáticos em texto livre	LSTM, LSTM com Atenção, RMC e Transformer	76% Interpolação e 50% Extrapolação
Attending to Mathematical Language with Transformers	2019	Wangperawong	Expressões Matemáticas	Transformer, Transformer Universal e Transformer Universal Adaptativo	84,90%
Deep Learning for Symbolic Mathematics	2020	Lample, Charton	Equações Diferenciais e Integração	Transformer	99,7% Integração e até 94% Equações Diferenciais
Enhancing the Transformer With Explicit Relational	2020	Schlag, Smolensky, Fernandez, Jojic, Schmidhuber, Gao	Problemas Matemáticos em texto livre	TP-Transformer	81,92% Interpolação e 54,67% Extrapolação
Learning advanced mathematical computations from examples	2021	Charton, Hayat, Lample	Propriedades Qualitativas e Quantitativas de Objetos Matemáticos	Transformer	95% Qualitativas e 85% Cálculos
Measuring Mathematical Problem Solving With the MATH Dataset	2021	Hendrycks, Burns, Kadavath, Arora, Basart, Tang, Song, Steinhardt	Problemas Matemáticos	GPT-2 e GPT-3	6,9%

Elaborada pelo autor

4 MÉTODO

O objetivo principal deste trabalho é a criação de um *step analyzer* e a automatização do módulo de domínio de um sistema tutor inteligente baseado em passos, a partir de um modelo, que utiliza *Deep Learning* para realizar a correção de equações de primeiro grau. 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.

Essas atualizações nos dados permitem ainda alterações nos objetos avaliados e a expansão do *step analyzer*, de forma que, além de classificar os passos realizados pelos estudantes, ele possa indicar os componentes de conhecimento utilizados pelo estudante ao resolver a equação ou retornar a descrição de qual erro foi cometido.

O modelo aqui proposto 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. Feito isso, verifica se o passo 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.

A Figura 4.1 mostra as fases de desenvolvimento do trabalho e criação do modelo, iniciando pela coleta e análise de dados. Posteriormente, são mostrados o tratamento de dados e o pré-processamento. Em seguida, a seleção da arquitetura utilizada, treinamento, validação e ajustes de parâmetros. Por fim, os testes em dados não vistos pelo modelo, a apuração e avaliação dos resultados obtidos.

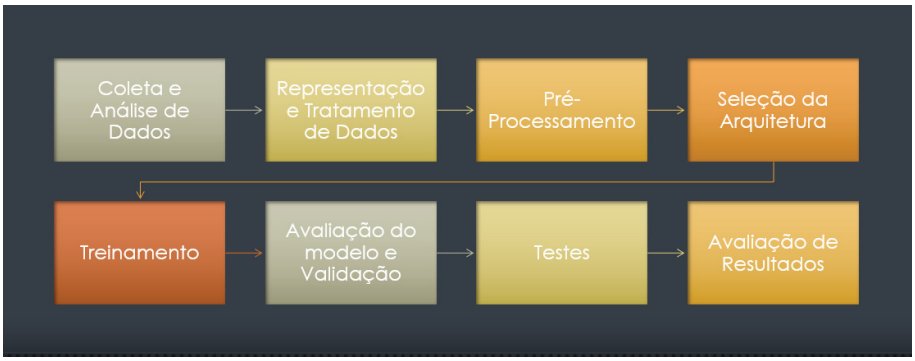


Figura 4.1: Fases de desenvolvimento do trabalho

Elaborado pelo autor

Este capítulo descreve todo o processo de desenvolvimento do trabalho, até a seleção da arquitetura. As demais fases são tratadas no Capítulo 5. Ao todo, foram criadas seis versões principais, sendo três versões utilizando a arquitetura de redes neurais e três versões com *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 deste 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*. 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 pelo 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 várias 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 4.2. 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 4.2: Cinco primeiras linhas do conjunto de dados atualizado

Elaborado pelo autor

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 Seção 4.4.

4.3 REPRESENTAÇÃO DE DADOS

Os dados inicialmente retirados no *log* do *PAT2Math*, contém equações na forma infixa, que é o formato mais tradicionalmente utilizado para representar equações, com os operadores inseridos entre os operandos.

Entretanto, a notação infixa exige a utilização de parênteses, para que não exista ambiguidade na ordem da resolução das equações. Por esse motivo, quando equações são utilizadas em sistemas computacionais, geralmente são usados outros tipos de notação, que conseguem lidar com a questão da ambiguidade, eliminando a necessidade de parênteses e reduzindo o número de termos da equação.

As duas notações mais comuns desse tipo, são a notação prefixada, também conhecida como notação polonesa, proposta por Łukasiewicz, em 1924, em que os operadores sao inseridos

antes dos operandos e a notação posfixada, também chamada de notação polonesa reversa, proposta nos anos 50, no qual os operadores são descritos após os operandos (McIlroy, 2023).

A conversão das equações iniciais, em notação infixa, para as equações resultantes, em notação posfixada, foi realizada por um código baseado no algoritmo *shunting yard*, de Dijkstra, que recebeu esse nome pela semelhança com trilhos ferroviários. Por exemplo, a equação $x + 7 = 13 + 1$, descrita na notação infixa, é convertida para $x7+ = 131+$, na notação pósfixada (McIlroy, 2023)

Este trabalho utiliza as equações representadas tanto na notação infixa quanto na notação posfixada. Nas versões iniciais, ambos os sistemas atingiram resultados similares. Dessa forma, como o modelo conseguiu trabalhar bem com ambas, as versões finais priorizaram a notação infixa, que facilita a visualização e apresentação das equações. Os resultados dos modelos podem ser consultados no Capítulo 5.

4.4 TRATAMENTO DE DADOS

Esta seção 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, que continham os mesmos valores nas duas colunas: *initial equation* e *currentStep*. A exclusão reduziu o drasticamente o número de instâncias, passando das cerca de 251 mil, para um pouco menos de 35 mil interações.

Essa exclusão é bastante importante, porque se não for realizada, pode acarretar em 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* e suas funções *sympify* e *solve*; 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 ainda tem 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 que estavam classificados incorretamente, 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 correto. Dessas instâncias, 20.919 estão rotuladas como incorretas e 9.360 estão rotuladas como corretas, remetendo a um novo problema: o desbalanceamento de dados, tratado na Seção 4.5.

4.5 PRÉ-PROCESSAMENTO

Mesmo após o tratamento inicial dos dados, antes da utilização em algoritmos de *Deep Learning*, é necessária a realização do pré-processamento. Essa seção inclui o balanceamento/aumento de dados, a tokenização, codificação e ajuste do conjunto de dados.

4.5.1 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 multiclasse, 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* e *Área Sob a Curva da Característica de Operação do Receptor*, geralmente definida como *AUC-ROC*, que são mais robustas que a acurácia. Outra forma de lidar com o desbalanceamento é a utilização da validação cruzada. No caso deste trabalho, além da utilização das métricas *F1 Score* e *AUC-ROC*, 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).

Este trabalho usa duas diferentes funções de *data augmentation*:

- **Data Augmentation na coluna *initial equation***: a primeira função parte das instâncias rotuladas como corretas e altera a coluna *initial equation*, criando novas equações rotuladas como corretas ou incorretas. No caso da geração de equações corretas, a função inclui uma multiplicação, divisão, adição ou um subconjunto dessas operações, em ambos os lados da equação. Depois, também de forma aleatória, pode inverter os

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

- **Data Augmentation na coluna *current step***: a segunda função é aplicada a uma quantidade significativamente menor de dados (cerca de 5% da função anterior), das instâncias rotuladas como corretas, desta vez, para alterar a coluna *current step*. No caso da geração de equações corretas, as equações presentes na coluna *initial equation* são copiadas para *current step*, podendo ter sua ordem invertida. Já na geração de equações incorretas, a coluna *current step* recebe formações inválidas aleatórias.

A partir dessas funções, foram criadas mais 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.

4.5.2 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 (subpalavra), 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 número ou símbolo 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, Debut, Chaumond, & Wolf, 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 *DistilBERTTokenizerFast* utiliza uma técnica de *encoding* chamada *Word piece tokenization*. 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).

A Figura 4.3 demonstra o processo desde a conversão da notação da equação, até o processo de tokenização e *encoding*, realizadas após as exclusões dos dados.

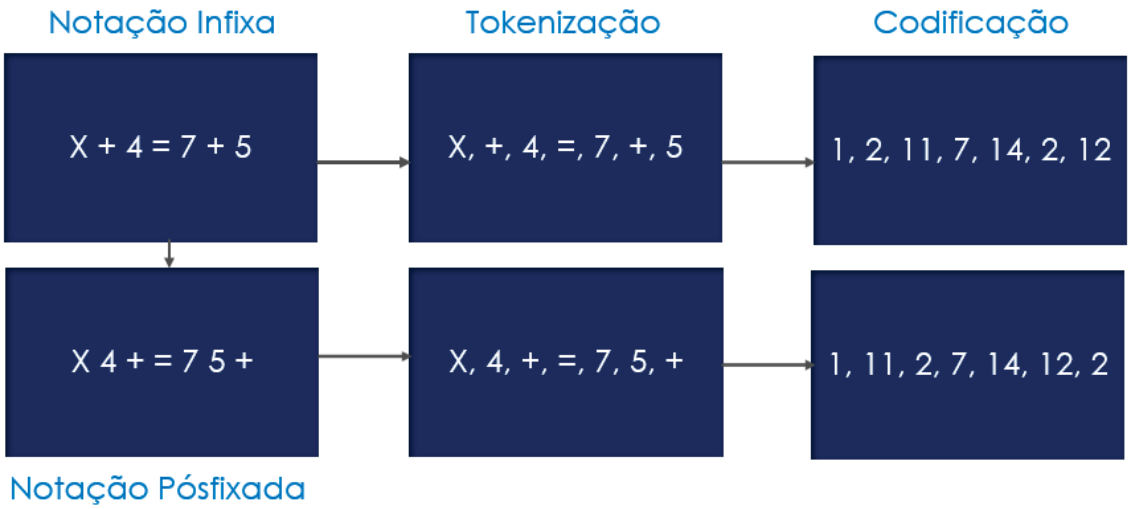


Figura 4.3: Processo de conversão entre notações, tokenização e codificação

Elaborado pelo autor

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

Tabela 4.1: Diferença entre entradas separadas (initial equation e current Step) e entrada única (input sequence)

Elaborado pelo autor

4.5.3 Entrada Única/Duas entradas

Para utilização nas redes neurais e *transformers*, o tutorial do tokenizador *DistilBERT* (HuggingFace, 2023), utilizado neste trabalho, sugere que os dados sejam modificados, unindo as colunas de entrada e incluindo um separador central ([SEP]). Assim, as primeiras versões deste trabalho, realizaram a união das colunas *initial equation* e *current step* e atribuíram o resultado à coluna *input sequence*, passando o conjunto de dados a ter somente duas colunas, a entrada (*input sequence*) e o *label* (*step is correct*). A coluna *step is correct*, também é modificada, de forma que possua somente os números inteiros 0 ou 1, ao invés de números de ponto flutuante.

A Tabela 4.1 exemplifica como seria essa modificação. 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).

Entretanto, ao passar duas entradas distintas, o próprio tokenizador já implementa essa modificação internamente, sem que isso afete os resutados finais do modelo. Dessa forma, este trabalho utiliza as duas entradas iniciais (*initial equation* e *currentStep*), sem realizar as modificações propostas, mantendo o formato original, com exceção de *step is correct*, que é mantido apenas com números inteiros.

4.5.4 Divisão de dados

Os dados consolidados são embaralhados, para que não exista duplicidade de instâncias 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 a apuração dos resultados.

4.6 ARQUITETURA E HIPERPARÂMETROS

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, ativada 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*, *RoBERTa*, entre outros.

4.6.1 Hiperparâmetros

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)*, 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 multiclasse, 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 três métricas diferentes: acurácia, para medir o percentual de instâncias classificadas corretamente; *F1 Score*, que faz uma média harmônica entre as métricas *precision* e *recall*, sendo especialmente útil para medição de desempenho em dados desbalanceados; *AUC-ROC* (Área Sob a Curva da Característica de Operação do Receptor), que avalia o desempenho de modelos de classificação binária.

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.

No Capítulo 5, são apresentadas cada uma das versões elaboradas para a criação do modelo final, destacando as diferenças entre cada uma delas, descrevendo e avaliando os resultados obtidos.

4.7 CONCLUSÃO

Este capítulo apresentou todo o desenvolvimento realizado no trabalho, desde a coleta dos dados no *PAT2Math*, na Seção 4.1, até a análise, representação e tratamento desses dados, apresentadas nas Seções 4.2, 4.3 e 4.4, respectivamente. Na Seção 4.5, são descritas as etapas de pré-processamento, com a correção de rótulos, inclusão de instâncias, através da função *data augmentation*, tokenização, codificação, ajustes e separação dos conjuntos de dados. Por fim, na Seção 4.6 são apresentados os modelos e as arquiteturas utilizadas, além das suas respectivas configurações.

5 AVALIAÇÃO E RESULTADOS

Este capítulo descreve 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 e *transformers*. Para as arquiteturas citadas, foram criadas seis versões principais, que foram sendo aprimoradas e são listadas nas Seções 5.1 e 5.2. O *link* para os códigos completos é disponibilizado no Apêndice A.

Para facilitar a compreensão, essas versões foram agrupadas na Tabela 5.1, 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. 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.

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. Na Seção 6.1, do Capítulo 6, também são listadas as limitações e dificuldades encontradas.

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;
- **Métricas:** Acurácia, *F1 Score* e *AUC-ROC*.

	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

Tabela 5.1: Versões utilizadas para criação do modelo final

Elaborado pelo autor

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 44.863 instâncias, sendo 22.436 corretas e 22.427 incorretas.

O destaque desta versão, é a sua simplicidade e rapidez na execução do treinamento. O modelo foi treinado em apenas 2 minutos e 57 segundos. Quanto aos resultados, enquanto no conjunto de treinamento, a acurácia chegou a $88,61\%$, no conjunto de validação, a acurácia atingiu **84,47%** e a função de *loss* chegou a $0,3367$. 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. Os gráficos apresentados nas Figuras 5.1 e 5.2 mostram, respectivamente, a variação da função de perda e da acurácia, tanto no conjunto de treinamento quanto no conjunto de validação. O menor valor da função de perda, e consequentemente o melhor modelo dessa versão, é obtido na 14ª época. Após, o valor da perda começa a subir suavemente.

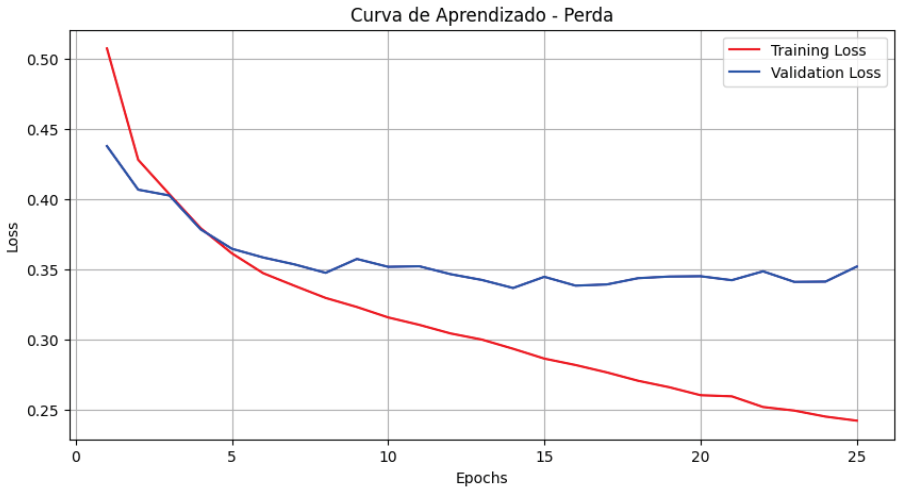


Figura 5.1: Versão 1 - Curva de aprendizado - Perda

Elaborado pelo autor

O resultado final deste modelo, conforme apurado junto ao conjunto de testes, foi de **85,24%** de acurácia na correção de equações do conjunto de testes, atingindo um *F1 Score* de 83,55%. Foram 3.825 instâncias preditas corretamente e 662 preditas incorretamente. Na Figura 5.3, a matriz de confusão ilustra essas informações. No eixo vertical são mostrados os rótulos reais e no eixo horizontal constam os rótulos preditos. O tom de azul fica mais escuro, de acordo com o número de instâncias contidas em cada quadro. São 2.143 verdadeiros negativos, 1.682 verdadeiros positivos, 513 falsos negativos e 149 falsos positivos.

Por fim, a Figura 5.4 mostra a área sob a cruva (ROC). A curva ROC é uma representação gráfica que mostra a taxa de verdadeiros positivos (sensibilidade) em relação à taxa de falsos positivos (especificidade) para diferentes valores de limite de decisão do modelo. Quanto mais próximo de 1, melhor é o desempenho do modelo para distinguir entre as classes. Na versão 1, o valor chegou a 0,92.

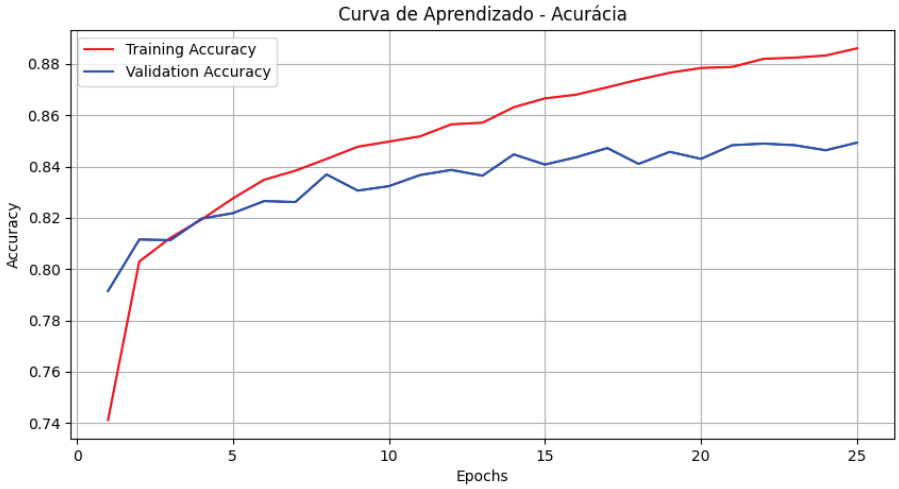


Figura 5.2: Versão 1 - Curva de aprendizado - Acurácia

Elaborado pelo autor

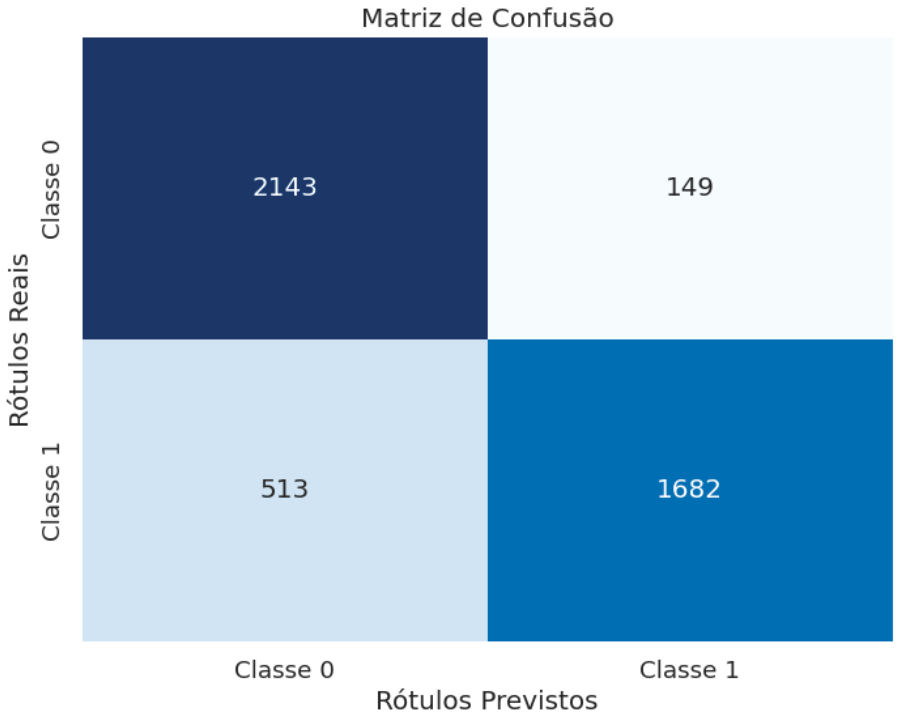


Figura 5.3: Versão 1 - Matriz de confusão

Elaborado pelo autor

5.1.2 **Versão 2**

A segunda versão desta arquitetura utilizou um número bem superior de dados, totalizando 78.378 instâncias, sendo que em 39.152 o passo está correto e em 39.226 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 quatro minutos e 56 segundos. No conjunto de treinamento, houve um aumento de

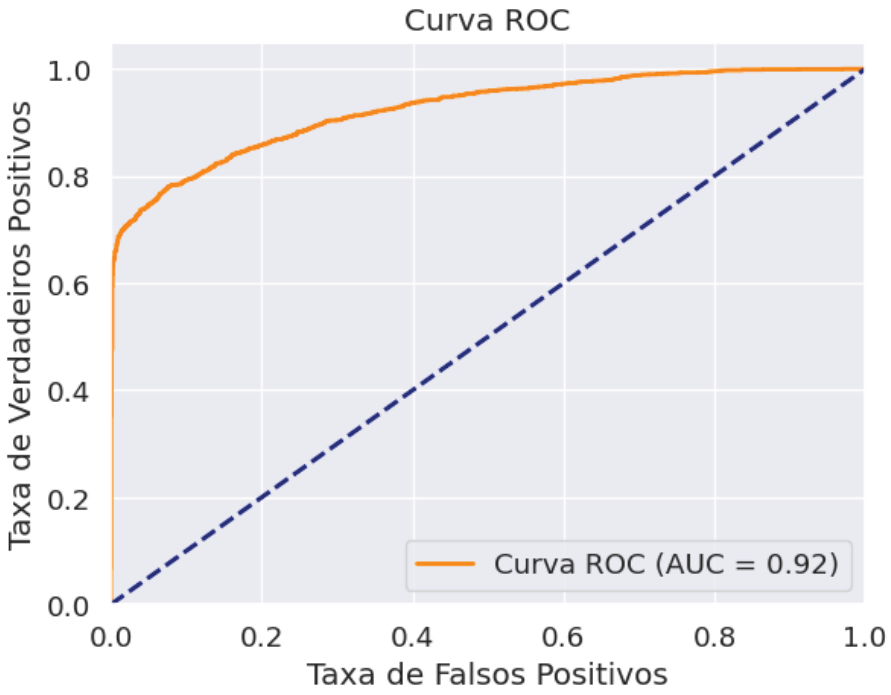


Figura 5.4: Versão 1 - AUC-ROC

Elaborado pelo autor

quatro pontos percentuais na acurácia, chegando a 92, 53%. No conjunto de validação, a acurácia chegou a **87,70%**, um aumento de cerca de três pontos percentuais. A função de perda ficou em 0, 2778. Os gráficos apresentados nas Figuras 5.5 e 5.6 mostram a variação desses valores.

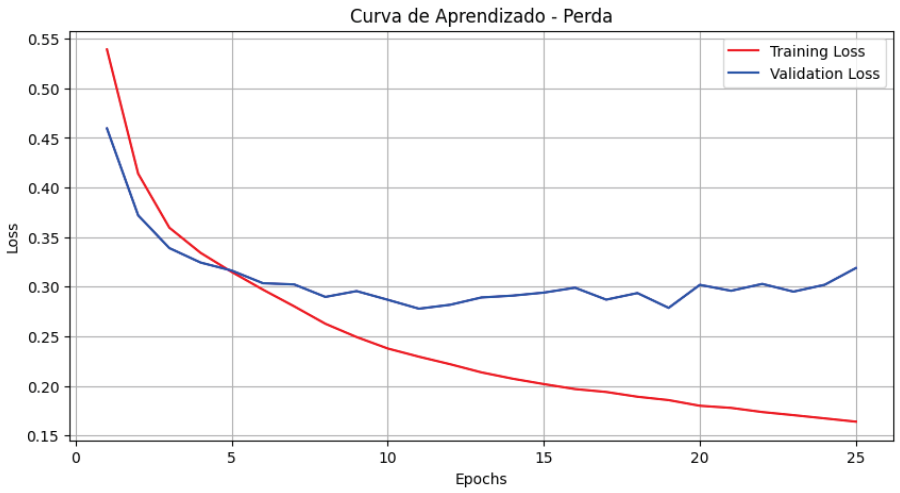


Figura 5.5: Versão 2 - Curva de aprendizado - Perda

Elaborado pelo autor

Nesta versão, o menor valor na função de perda foi obtido já na 11ª época, subindo gradativamente, na sequência. Mesmo que a perda no conjunto de treinamento continue reduzindo, o aumento da função de perda no conjunto de validação indica *overfitting*.

O resultado apurado para o modelo, no conjunto de testes, foi de **88,38%**, três pontos percentuais acima da versão anterior, com um *F1 Score* de 88,42%. Foram 6.927 rótulos

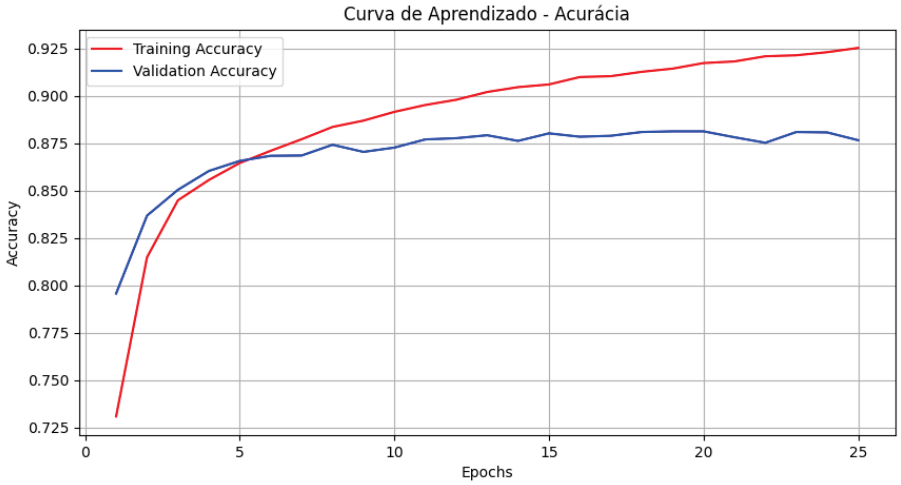


Figura 5.6: Versão 2 - Curva de aprendizado - Acurácia

Elaborado pelo autor

classificados corretamente e 911 incorretamente. A matriz de confusão desta versão, pode ser visualizada na Figura 5.7, que mostra um desempenho equilibrado na predição das duas classes.

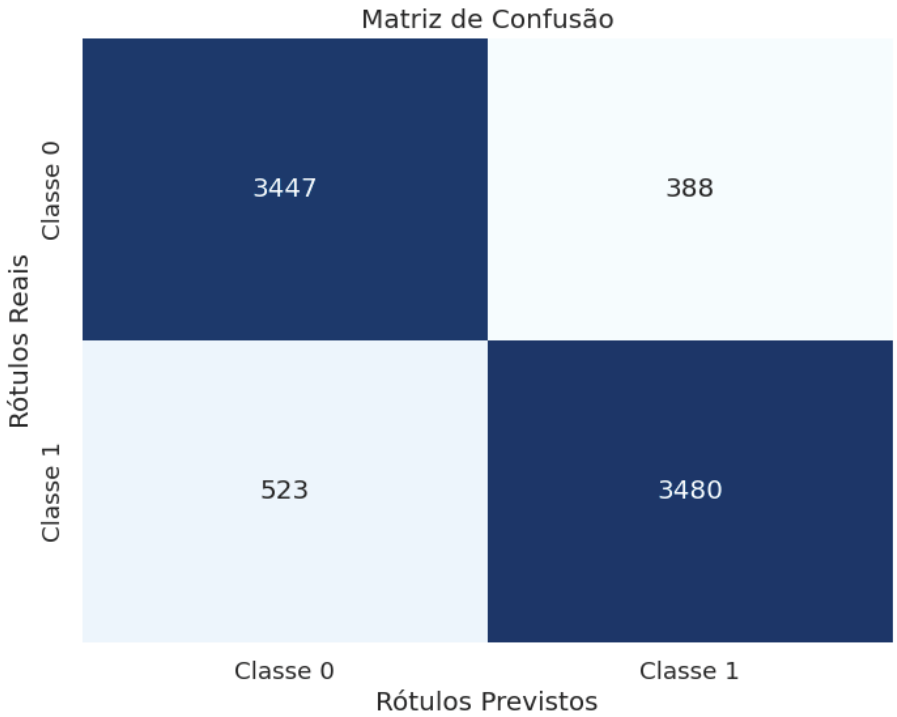


Figura 5.7: Versão 2 - Matriz de confusão

Elaborado pelo autor

Por fim, a Figura 5.8 mostra a área sob a curva, que atingiu 0,95. A linha laranja mais próxima do canto superior esquerdo indica um melhor desempenho na comparação com a versão anterior.

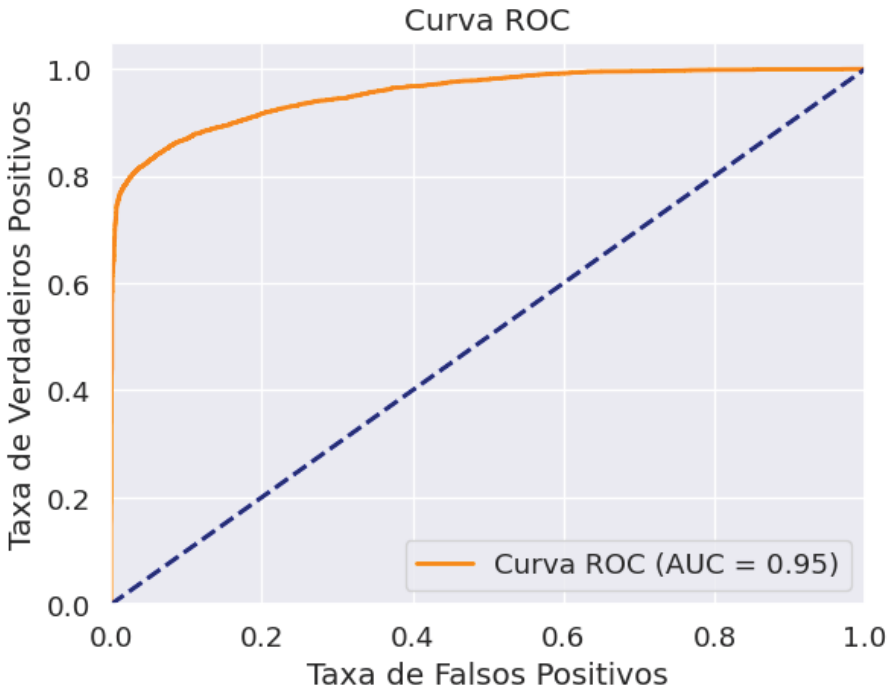


Figura 5.8: Versão 2 - AUC-ROC

Elaborado pelo autor

5.1.3 **Versão 3**

A versão final, das que utilizam arquitetura de redes neurais, contou com ajustes no algoritmo de *data augmentation* (que passou a ser aplicada em ambas as colunas de entrada), aumento no número de instâncias para 115.931 (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 47 segundos e a acurácia no conjunto de validação chegou a **90,08%**, com 0, 226 como resultado de perda (atingido na 12ª época). O resultado é cerca de dois pontos percentuais superior ao modelo anterior e pode ser conferido nos gráficos apresentados nas Figuras 5.9 e 5.10.

O resultado final deste modelo e da arquitetura de redes neurais foi de **90,26%** de acurácia no conjunto de testes, chegando a um *F1 Score* de 89,67%. São 10.465 rótulos classificados corretamente e 1.129 classificados de forma incorreta, como pode ser visto na matriz de confusão da Figura 5.11.

A matriz mostra um desempenho superior na predição de classes negativas. São 5.560 verdadeiros negativos, 4.905 verdadeiros positivos, 757 falsos negativos e 372 falsos positivos. Na sequência, a Figura 5.12 mostra também a área sob a curva, métrica que atingiu um resultado de 0,97.

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:

- **Otimizador:** Adam, com *learning rate* de $3e-5$;

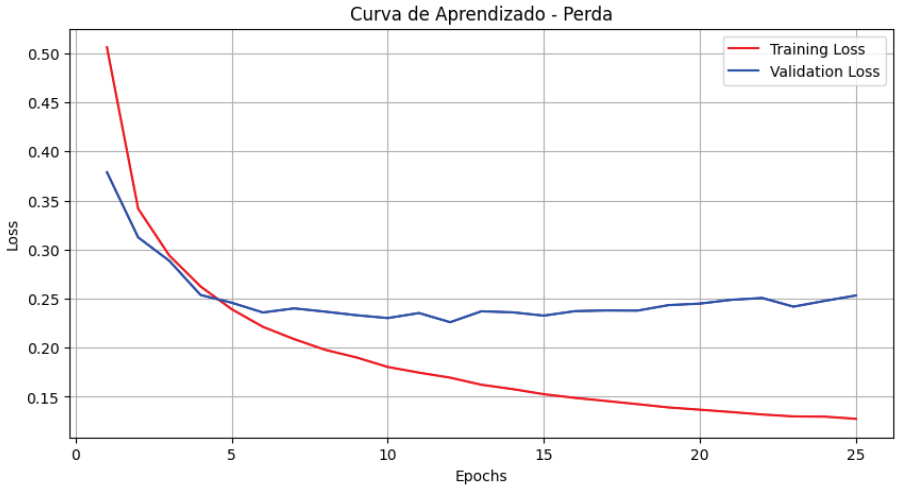


Figura 5.9: Versão 3 - Curva de aprendizado - Perda

Elaborado pelo autor

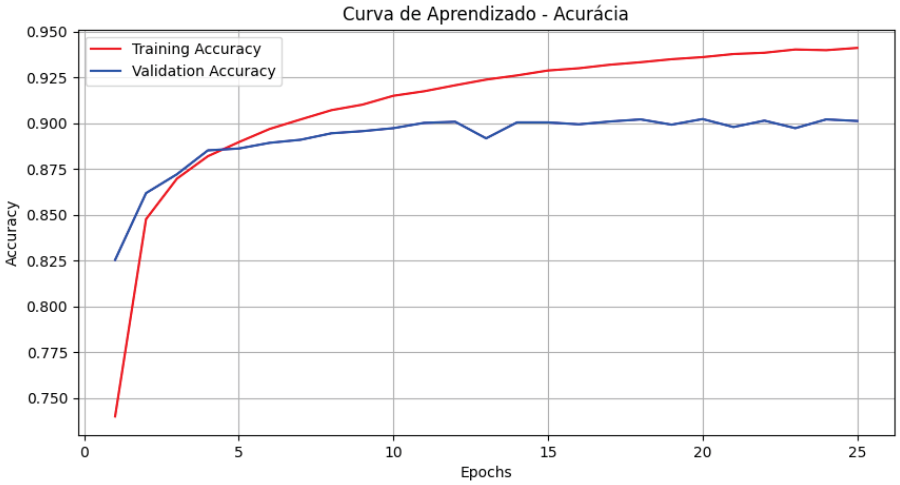


Figura 5.10: Versão 3 - Curva de aprendizado - Acurácia

Elaborado pelo autor

- **Função de Perda:** *Sparse Categorical Crossentropy*;
- **Tamanho do Batch:** 32;
- **Métricas:** Acurácia, *F1 Score* e *AUC-ROC*.

5.2.1 Versão 4

A primeira versão com *transformers* apresentada neste trabalho, também foi testada usando cerca de 45 mil instâncias (na verdade, 44.863). Diferente das demais versões, esta versão foi utilizada tanto com equações em notação infixa quanto com equações em notação posfixada. Como o resultado foi um pouco inferior utilizando a notação posfixada, a decisão foi por manter apenas a notação infixa nas versões posteriores, que tornam a descrição das equações mais claras e simples.

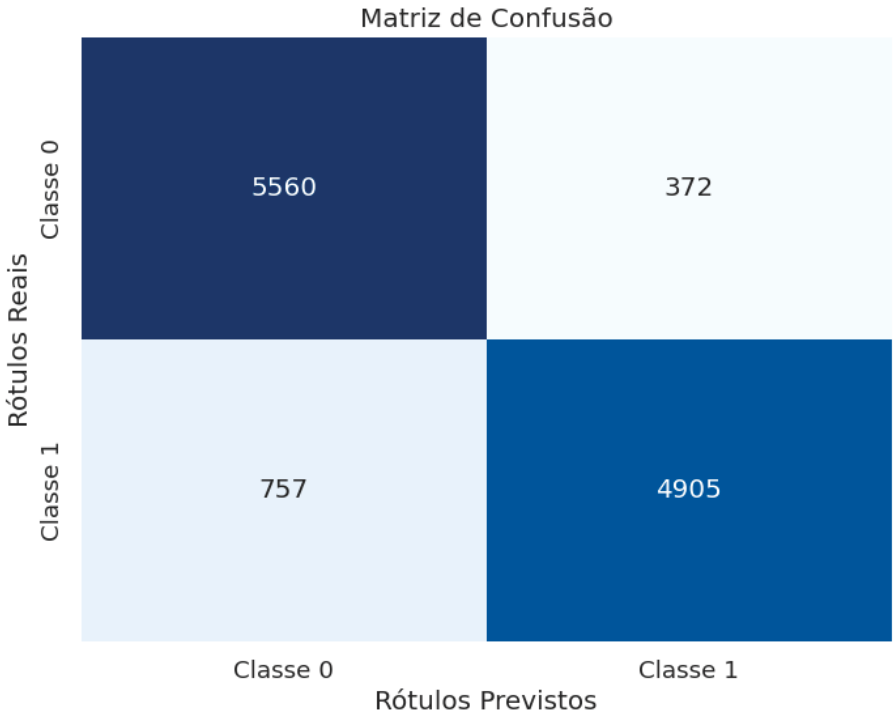


Figura 5.11: Versão 3 - Matriz de confusão

Elaborado pelo autor

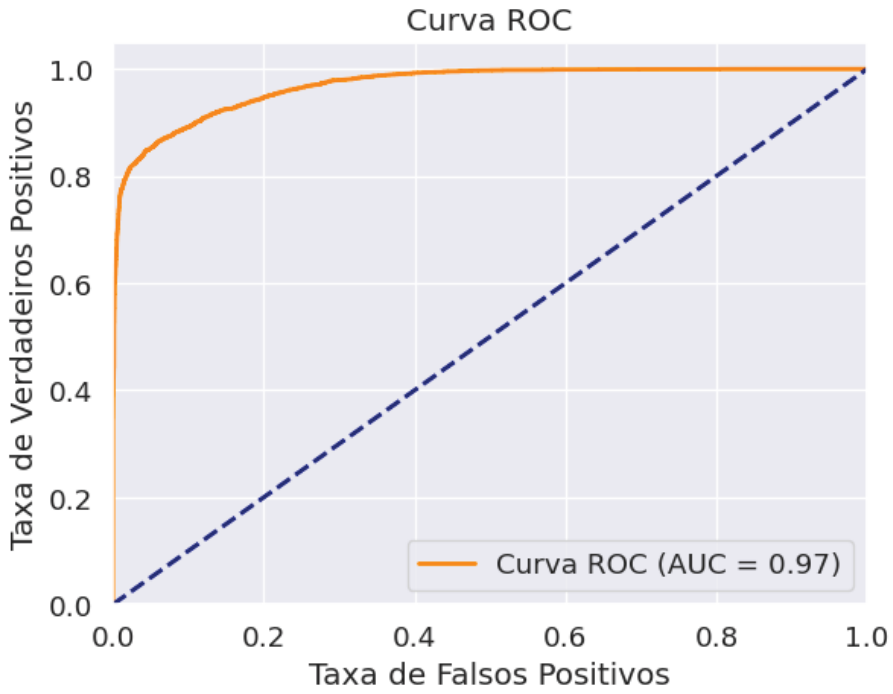


Figura 5.12: Versão 3 - AUC-ROC

Elaborado pelo autor

A primeira diferença óbvia entre esta versão e as versões anteriores é o tempo de treinamento. As versões que estão utilizando a arquitetura *transformers* demoram bem mais

tempo no treinamento do modelo. Para fins de comparação, cada época deste modelo leva cerca de três minutos e meio para ser treinada, quase metade do tempo de treinamento total da última versão com redes neurais.

Esta versão levou cerca de duas horas e trinta minutos para conclusão do treinamento. No conjunto de validação, o modelo atingiu uma acurácia de **86,79%** e a função de perda chegou a $0,2781$, valores bem inferiores ao modelo completo com a arquitetura de redes neurais. Os gráficos apresentados nas Figuras 5.13 e 5.14 mostram, respectivamente, a evolução do modelo durante o treinamento e validação.

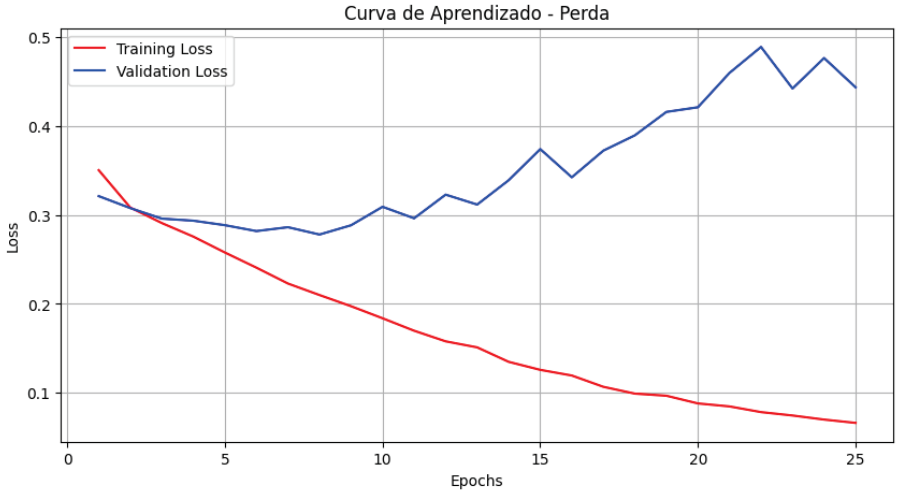


Figura 5.13: Versão 4 - Curva de aprendizado - Perda

Elaborado pelo autor

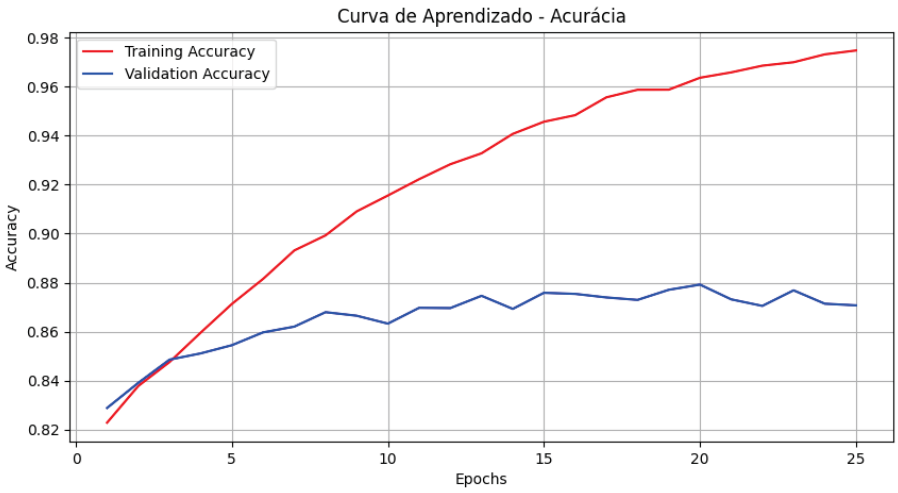


Figura 5.14: Versão 4 - Curva de aprendizado - Acurácia

Elaborado pelo autor

Os gráficos mostram outra diferença importante: o menor valor de perda desta versão, foi obtido logo na oitava época, um pouco mais cedo do que nas versões que utilizaram arquitetura de redes neurais.

Como resultado final, o modelo atingiu **87,12%** de acurácia no conjunto de testes, um resultado de mais de três pontos percentuais menor do que o melhor resultado obtido com redes

neurais. O *F1 Score* chegou a 87,55%. Foram 3.909 rótulos classificados corretamente e 578 classificados incorretamente. Esses resultados podem ser visualizados na matriz de confusão da Figura 5.15.

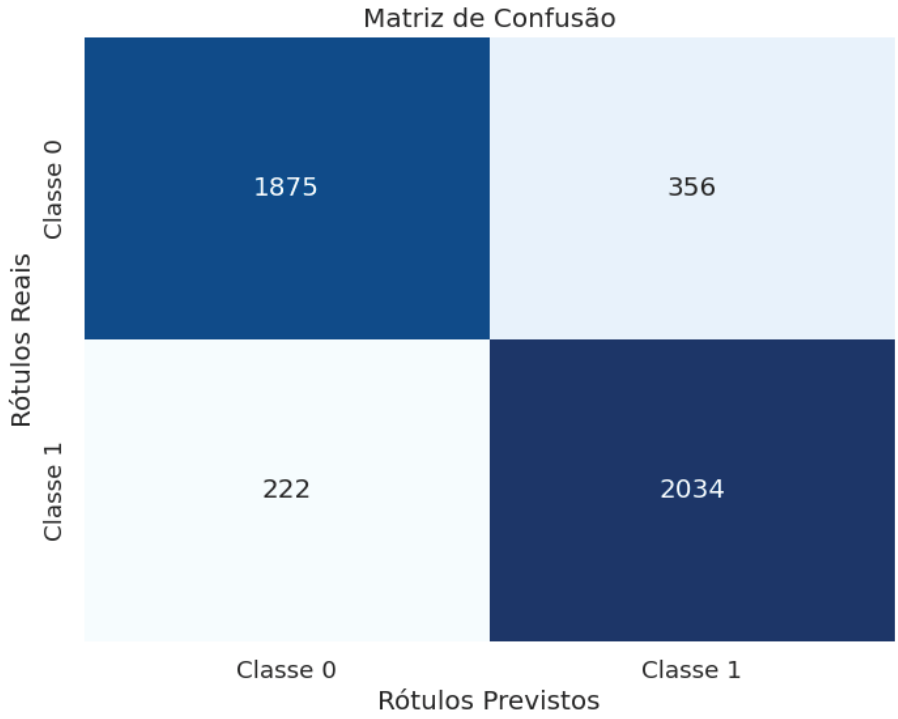


Figura 5.15: Versão 4 - Matriz de confusão

Elaborado pelo autor

A matriz apresentou um restrospecto melhor na predição de rótulos positivos, sendo 2.034 verdadeiros positivos, 1.875 verdadeiros negativos, 356 falsos positivos e 222 falsos negativos. Finalizando, a área sob a curva chegou a um valor de 0,96, resultado que pode ser visto na Figura 5.16.

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 77.286 instâncias usadas para criação do modelo.

O treinamento demorou cerca de três horas e 56 minutos para ser completado. Nessa versão, a acurácia chegou a **92,60%** e a função de *loss* a um valor de 0, 1535 (obtido na 11ª época) no conjunto de validação. Os valores já são bem superiores ao de todas as demais versões. Os gráficos apresentados nas Figuras 5.17 e 5.18 mostram, respectivamente, a evolução da perda e da acurácia, nos conjuntos de treinamento e validação.

No conjunto de testes, o modelo chegou a um percentual de **93,25%** de acurácia. Quase três pontos percentuais acima do melhor modelo anterior. O *F1 Score* chegou a 93,18%. Foram 7.208 instâncias classificadas corretamente e 521 de forma incorreta. A matriz de confusão do modelo ilustra esses dados na Figura 5.19.

São 3.647 verdadeiros negativos, 3.561 verdadeiros positivos, 308 falsos negativos e 213 falsos positivos. Em seguida, a Figura 5.20, mostra a área sob a curva, que chegou a 0,99.

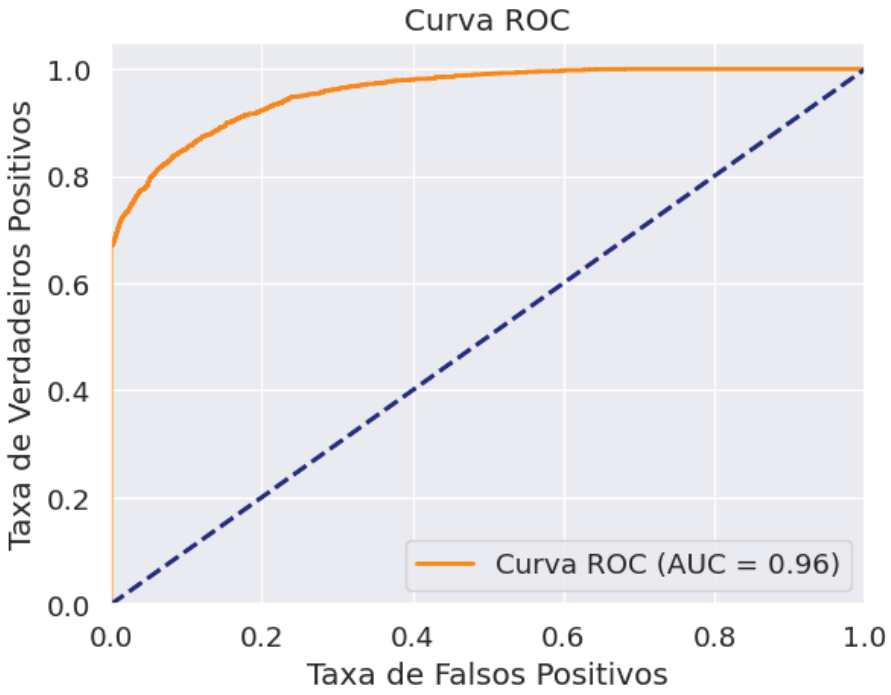


Figura 5.16: Versão 4 - AUC-ROC

Elaborado pelo autor

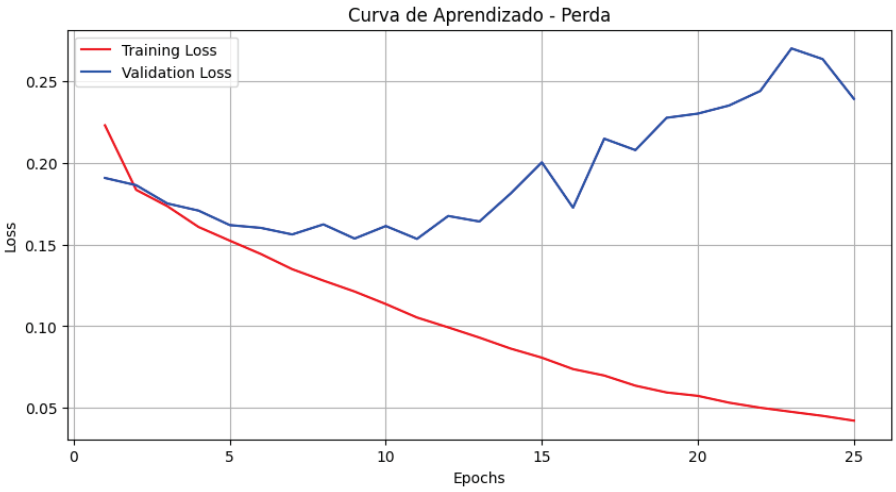


Figura 5.17: Versão 5 - Curva de aprendizado - Perda

Elaborado pelo autor

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

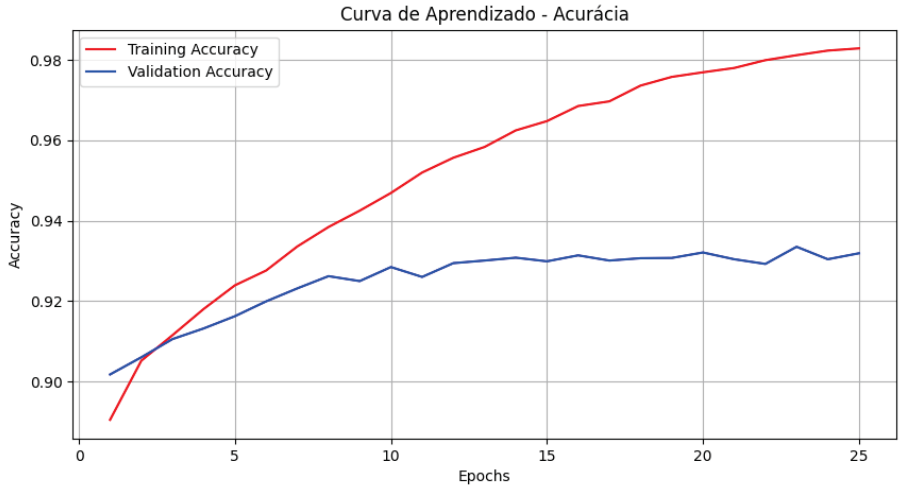


Figura 5.18: Versão 5 - Curva de aprendizado - Acurácia

Elaborado pelo autor

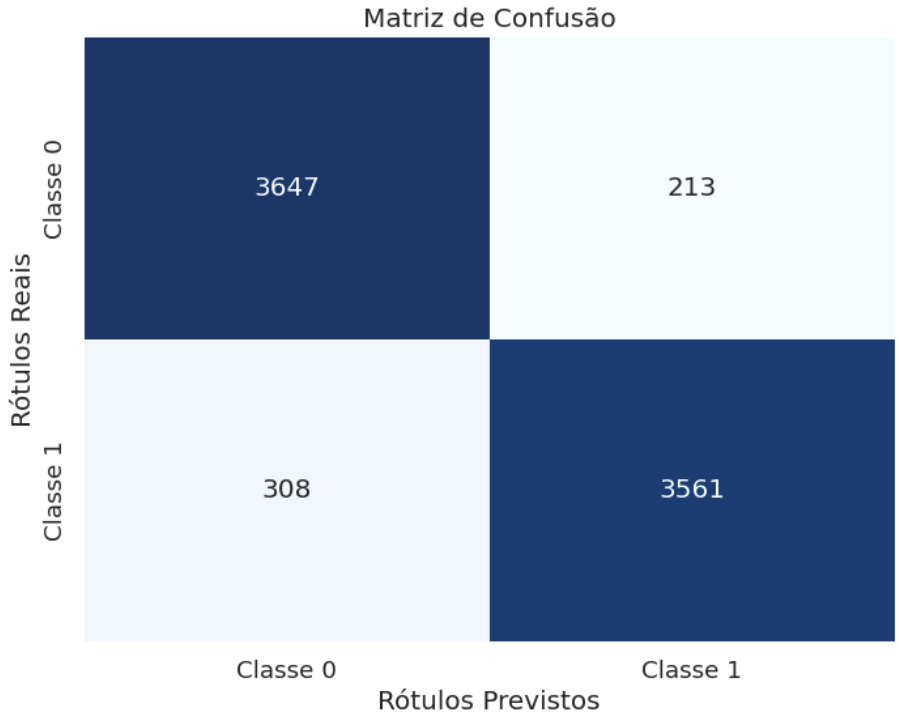


Figura 5.19: Versão 5 - Matriz de confusão

Elaborado pelo autor

O treinamento dessa versão foi o mais demorado, levando cerca de 5 horas e 40 minutos para a conclusão. Após o treinamento, o modelo gerado chegou a um percentual de **94,60%** de acurácia no conjunto de validação e um valor de 0, 1175 na função de perda (atingido na 8ª época). O valor é 2% superior ao segundo colocado e as variações podem ser visualizadas nos gráficos mostrados nas Figuras 5.21 e 5.22.

No conjunto de teste, o modelo atingiu **95,84%** de acurácia, sendo esse o resultado final do modelo e o resultado final deste trabalho. O *F1 Score* chegou a 95,80%. No total foram

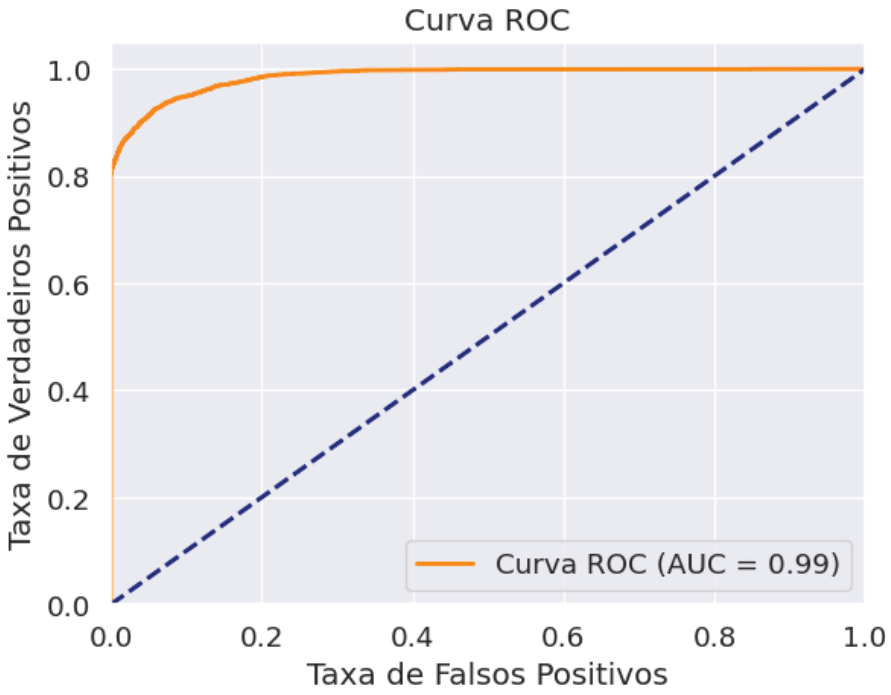


Figura 5.20: Versão 5 - AUC-ROC

Elaborado pelo autor

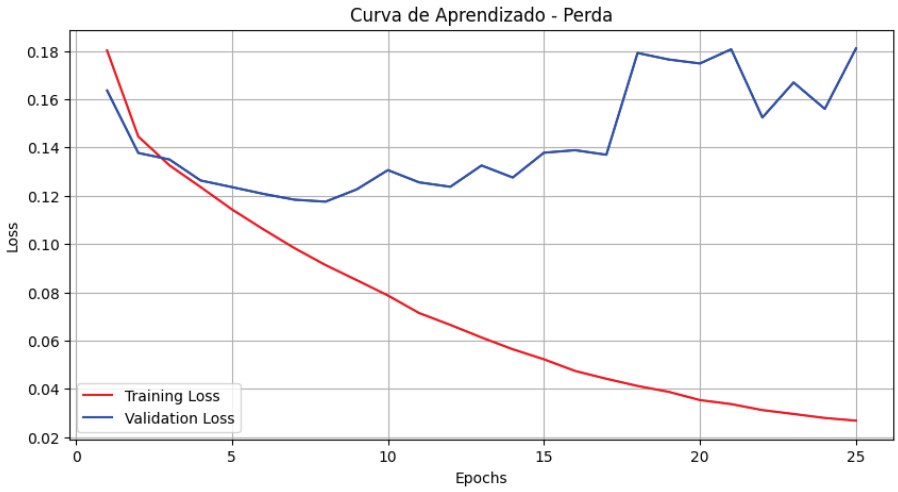


Figura 5.21: Versão 6 - Curva de aprendizado - Perda

Elaborado pelo autor

11.109 instâncias classificadas corretamente, contra 482 classificadas incorretamente. A Figura 5.23 mostra a matriz de confusão.

Esta versão possui o resultado mais equilibrado na predição de rótulos negativos e positivos. São 5.609 verdadeiros negativos, 5.500 verdadeiros positivos, 241 falsos negativos e 241 falsos positivos. Em seguida, a Figura 5.24 mostra a área sob a curva, que chegou ao valor de 0,99, o valor mais alto atingido, que deixa a linha laranja mais próxima do canto superior esquerdo.

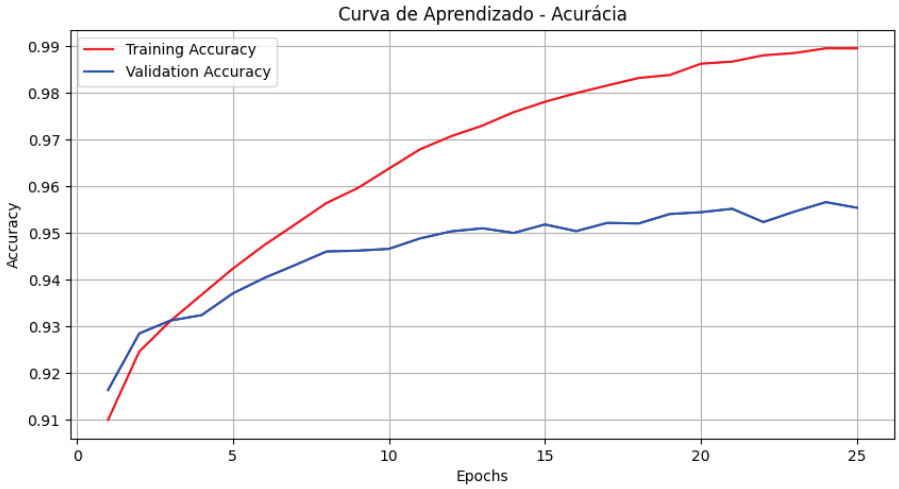


Figura 5.22: Versão 6 - Curva de aprendizado - Acurácia

Elaborado pelo autor

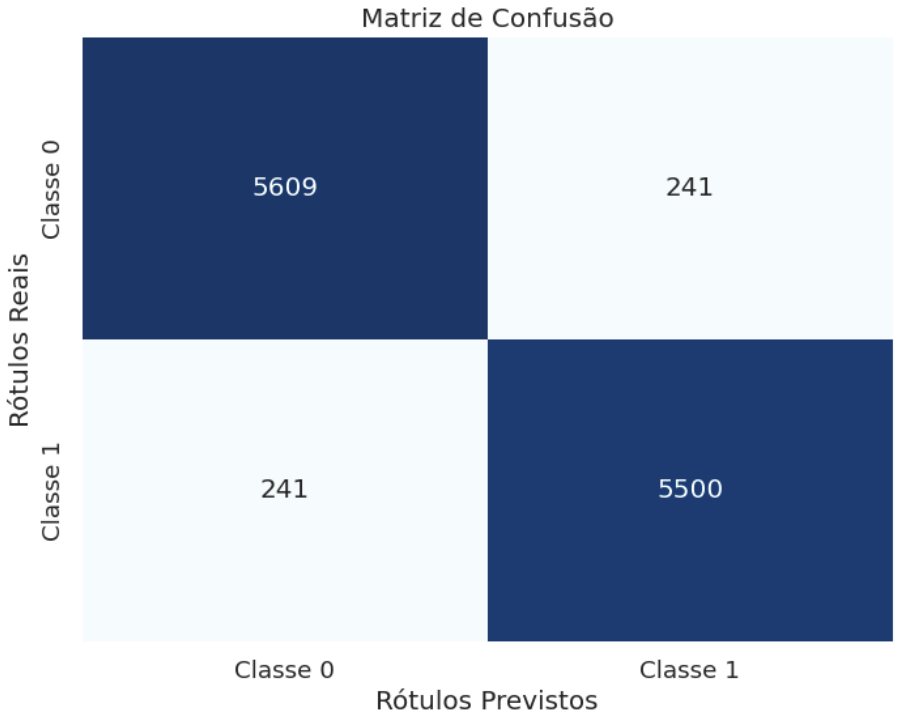


Figura 5.23: Versão 6 - Matriz de confusão

Elaborado pelo autor

Outra análise que pode ser realizada é a da Tabela 5.25, que mostra os rótulos classificados e a probabilidade com que o sistema decide por uma ou outra classe. A 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.

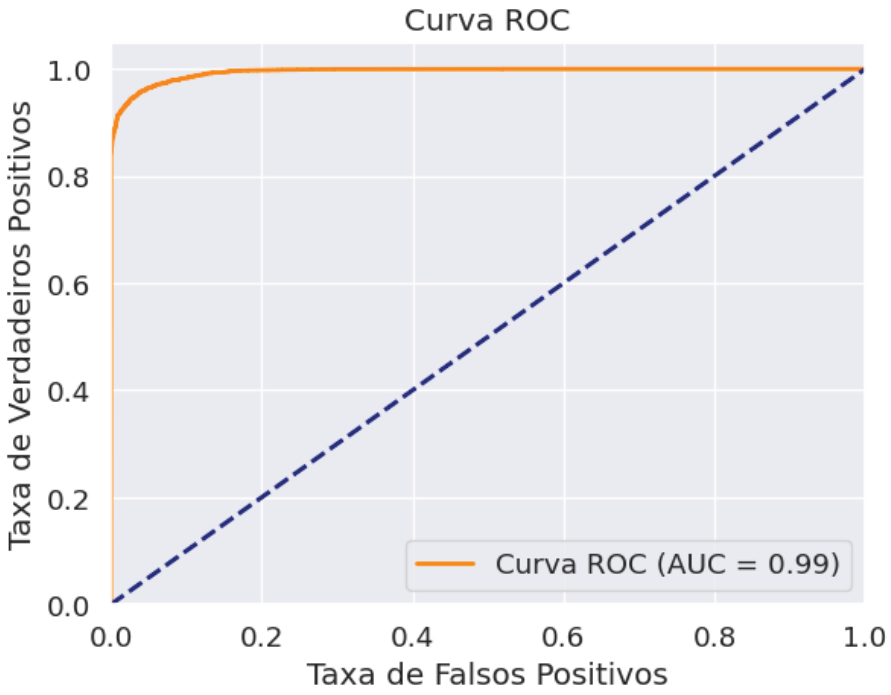


Figura 5.24: Versão 6 - AUC-ROC.

Elaborado pelo autor

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.

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	0.1791434362530708	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 5.25: Versão 6 - Resultado e probabilidade da predição

Elaborado pelo autor

5.3 AVALIAÇÃO

A Tabela 5.2 mostra as seis versões utilizadas, para fins de comparação. A primeira coluna informa o nome da versão. A segunda coluna indica a arquitetura utilizada, se rede neural GRU ou *transformers*. A terceira coluna mostra o número total de instâncias utilizadas na criação do modelo. A quarta coluna indica se os rótulos que estavam originalmente incorretos no banco de dados do sistema tutor, foram corrigidos. As colunas cinco, seis e sete, informam

Versão	Arquitetura	Nr. Inst.	Cor. Rótulos	Tempo Trein.	Acurácia Val.	Acurácia Teste	F1 Score	AUC ROC
Vers 1	GRU	44.863	Não	2m57s	84,47%	85,24%	83,55%	0,92
Vers 2	GRU	78.378	Não	4m56s	87,70%	88,38%	88,42%	0,95
Vers 3	GRU	115.931	Sim	7m47s	90,08%	90,26%	89,67%	0,97
Vers 4	Transformers	44.863	Não	2h30m	86,79%	87,12%	87,55%	0,96
Vers 5	Transformers	77.286	Não	3h56m	92,60%	93,25%	93,18%	0,99
Vers 6	Transformers	115.898	Sim	5h40m	94,60%	95,84%	95,80%	0,99

Tabela 5.2: Comparação entre os resultados obtidos por cada versão

Elaborado pelo autor

respectivamente, o tempo total para treinamento do modelo e as acurácias no conjunto de validação e conjunto de testes. Por fim, as colunas oito e nove, mostram o valor das métricas *F1 Score* e AUC/ROC, respectivamente. O resultado final de cada modelo é dado pela acurácia, no conjunto de testes.

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 **95,84%** de acurácia no conjunto de testes.

Dentre as conclusões retiradas, com base na Tabela 5.2, nota-se que as versões implementadas com redes neurais obtiveram um tempo de treinamento muito inferior, entretanto, ao mesmo tempo, tiveram resultados inferiores ao das versões que utilizaram transformers.

Igualmente, as versões que trabalharam com mais dados, neste caso, 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.

Assim como observado nos estudos do Capítulo 3, a implementação da arquitetura de *Transformers* demonstrou um desempenho significativamente superior em comparação com as redes neurais GRU. Notavelmente, a versão 6, que utilizou a arquitetura de *Transformers*, apresentou uma acurácia de aproximadamente 5,5% acima da acurácia obtida pela versão 3, que, até então, era a mais eficaz entre as implementações que utilizavam as redes neurais GRU.

Os gráficos das Figuras 5.26 e 5.27 demonstram respectivamente a variação da acurácia e da perda no conjunto de validação. Enquanto a acurácia das seis versões sobe quase a ritmo constante, durante o processo de treinamento, a perda cai inicialmente e após algumas épocas estabiliza e/ou passa a aumentar. Nas versões que utilizam redes neurais, a perda para de reduzir entre 11 e 14 épocas. No caso das versões que utilizam *transformers*, é possível verificar que isso ocorre mais cedo, entre 8 e 11 épocas.

Após os modelos atingirem o valor mínimo de perda, o treinamento realizado posteriormente apenas gera *overfitting*, ou seja, há um sobretreinamento do conjunto de validação que melhora o desempenho nesse conjunto de dados, mas não necessariamente irá desempenhar da mesma forma em dados não vistos anteriormente. Assim, é possível reduzir bastante o tempo de treinamento, para cerca de 11 épocas nas versões que utilizam *transformers*. Essa alteração permitiria selecionar os melhores modelos da arquitetura *transformers*, com tempos de treinamento reduzidos para cerca de 44% dos tempos totais informados na Tabela 5.2.

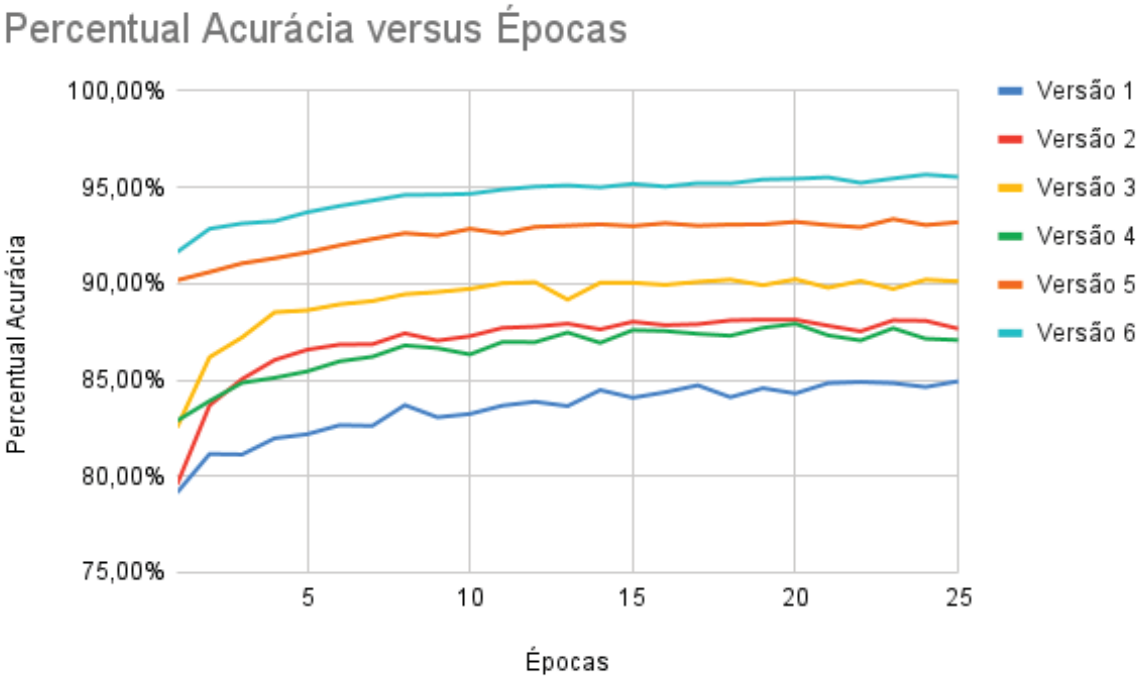


Figura 5.26: Evolução da acurácia das seis versões, no conjunto de validação

Elaborado pelo autor

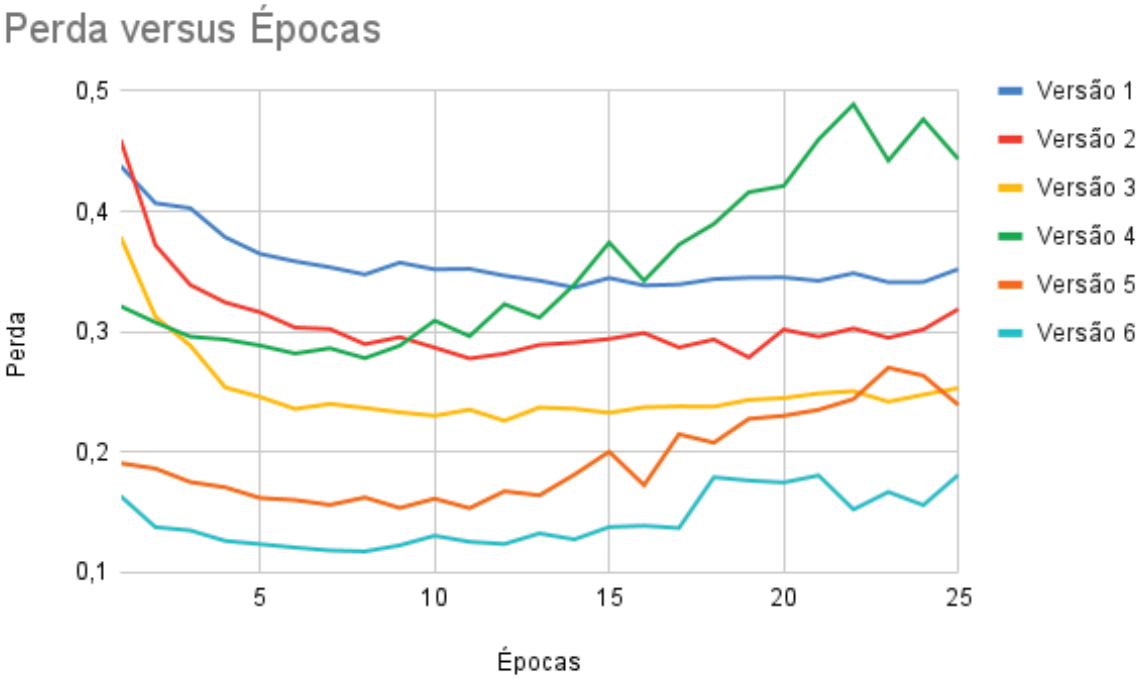


Figura 5.27: Evolução da perda das seis versões, no conjunto de validação

Elaborado pelo autor

5.4 CONCLUSÃO

Esta seção apresentou os resultados obtidos por seis versões diferentes, de duas arquiteturas distintas: redes neurais GRU e *transformers*. O final do capítulo, foi ilustrado com tabelas e gráficos comparativos, que indicam que a versão 6, que utiliza *transformers*, obteve o melhor resultado.

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.

6 CONCLUSÃO

Este trabalho apresentou o desenvolvimento de um modelo para automatização do módulo de domínio de um sistema tutor inteligente baseado em passos e criação de um *step analyzer*, para realizar 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 tabelas e gráficos comparativos e da análise dos seus desempenhos.

A partir dessa análise, o modelo com melhor desempenho, alcançou 95, 84% 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.

O *step analyzer* criado, também pode ser expandido, de forma a predizer os componentes de conhecimento utilizados e descrever os erros cometidos pelos estudantes durante a resolução das equações.

Este trabalho 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 à distância, que ganhou notória relevância, durante a pandemia da *COVID-19* (OPAS, 2023).

6.1 LIMITAÇÕES

Durante o desenvolvimento do trabalho, muitas dificuldades e limitações apareceram, sendo que a maior parte dessas dificuldades foram contornáveis. Nesse contexto, houveram duas principais limitações/dificuldades que podem ser destacadas:

- **Escassez de dados**: Foram utilizados dados disponíveis no *log* do sistema tutor inteligente *PAT2Math*, referentes ao ano de 2018. Pela própria natureza dos dados, mesmo com uma grande quantidade inicial de interações disponíveis, muitas delas não continham os dados necessários ao trabalho ou ainda continham dados repetidos. Essa limitação foi parcialmente tratada através da técnica de *data augmentation*, que permitiu ampliar o número de instâncias disponíveis, aumentar a generalização do modelo e, consequentemente, melhorar os resultados finais obtidos. De toda forma, com uma maior quantidade de dados, o modelo certamente atingiria resultados superiores.

- **Rótulos Incorretos**: Durante o desenvolvimento, foi identificado que quase 7% dos rótulos estavam com valores incorretos. Esse ruído nos dados impacta diretamente nos resultados finais obtidos. Essa questão também foi parcialmente solucionada, através do ajuste desses rótulos, com algumas poucas exceções. Dessa forma, essa dificuldade teve impacto muito baixo nos resultados finais.

REFERÊNCIAS

Alammar, J. (2020). *The illustrated gpt-2.* Retrieved 08.12.2020, from http://jalammar.github.io/illustrated-gpt2/

Anderson, J. R., Corbett, A. T., Koedinger, K., & Pelletier, R. (1995). Cognitive tutors: Lessons learned. *The Journal of the Learning Sciences*, *4*, 167-207.

Bahdanau, D., Cho, K., & Bengio, Y. (2014, 09). Neural machine translation by jointly learning to align and translate. *ArXiv*, *1409*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Carbonell, J. R. (1970). *Mixed-initiative man-computer instructional dialogues* (Unpublished doctoral dissertation). Massachusetts Institute of Technology.

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

Chawla, N. V. (2005). Data mining for imbalanced datasets: An overview. *Data Mining and Knowledge Discovery Handbook*, 853-867.

Chollet, F. (2021). *Deep learning with python, second edition*. Manning. Retrieved from https://books.google.com.br/books?id=mjVKEAAAQBAJ

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

Clancey, W. J. (1979). Tutoring rules for guiding a case method dialogue. *International Journal of Man-Machine Studies*, *11*(1), 25-49. Retrieved from https://www.sciencedirect.com/science/article/pii/S0020737379800048 doi: https://doi.org/10.1016/S0020-7373(79)80004-8

Crabtree, A., M. Nehme. (2023). *What is data analysis? an expert guide with examples.* Retrieved 10.09.2023, from https://www.datacamp.com/blog/what-is-data-analysis-expert-guide

DeepLearning.ai. (2020). *Natural language processing with attention models.* Retrieved 08.12.2020, from 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*.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, *abs/1810.04805*. Retrieved from http://arxiv.org/abs/1810.04805

Escovedo, T., & Koshiyama, A. (2020). *Introdução a data science: Algoritmos de machine learning e métodos de análise*. Casa do Código. Retrieved from https://books.google.com.br/books?id=cL7TDwAAQBAJ

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., ... Steinhardt, J. (2021). Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Hochreiter, S., & Schmidhuber, J. (1997, 12). Long short-term memory. *Neural computation*, *9*, 1735-80. doi: 10.1162/neco.1997.9.8.1735

HuggingFace. (2023). *Distilbert.* Retrieved 14.10.2023, from https://huggingface.co/docs/transformers/model_doc/distilbert

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. doi: 10.1016/j.eswa.2013.04.004

Khandelwal, R. (2020). *An intuitive explanation of beam search.* Retrieved 20.12.2020, from https://towardsdatascience.com/an-intuitive-explanation-of-beam-search-9b1d744e7a0f

Kostadinov, S. (2020). *Understanding encoder-decoder sequence to sequence model.* Retrieved 30.10.2020, from https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

Lample, G., & Charton, F. (2019). Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*.

Lane, H., Hapke, H., & Howard, C. (2019). *Natural language processing in action: Understanding, analyzing, and generating text with python*. Manning Publications. Retrieved from https://books.google.com.br/books?id=UyHgswEACAAJ

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

McCulloch, W., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, *7*, 115-133.

McIlroy, M. (2023). *Reverse polish notation.* Retrieved from https://mathworld.wolfram.com/ReversePolishNotation.html (From MathWorld–A Wolfram Web Resource, created by Eric W. Weisstein)

Nagori, V., & Trivedi, B. (2012, October). 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. Retrieved from https://ideas.repec.org/a/aml/intbrm/v3y2012i5p249-254.html

OPAS. (2023). *Histórico da pandemia de covid-19.* Retrieved 02.10.2023, from https://www.paho.org/pt/covid19/historico-da-pandemia-covid-19#

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, *abs/1910.10683*. Retrieved from http://arxiv.org/abs/1910.10683

Russell, S., & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Prentice Hall.

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*.

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

Saxton, D., Grefenstette, E., Hill, F., & Kohli, P. (2019). Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*.

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*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 27* (pp. 3104–3112). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

TensorFlow. (2020). *Transformer model for language understanding*. Retrieved 08.12.2020, from https://www.tensorflow.org/tutorials/text/transformer

Trask, A. W. (2019). *Grokking deep learning*. Simon and Schuster.

VanLehn, K. (2006, August). The behavior of tutoring systems. *Int. J. Artif. Intell. Ed.*, *16*(3), 227–265. Retrieved from http://dl.acm.org/citation.cfm?id=1435351.1435353

VanLehn, K. (2011). The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*, *46*(4), 197-221. Retrieved from http://www.tandfonline.com/doi/abs/10.1080/00461520.2011.611369 doi: 10.1080/00461520.2011.611369

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In I. Guyon et al. (Eds.), *Advances in neural information processing systems 30* (pp. 5998–6008). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

Wangperawong, A. (2018, December). Attending to Mathematical Language with Transformers. *arXiv e-prints*, arXiv:1812.02825.

Woolf, B. P. (2007). *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2020). *Dive into deep learning*. (https://d2l.ai)

APÊNDICE A – CÓDIGOS

Os códigos utilizados nas seis versões do Capítulo 5 estão disponíveis através do *link* `https://github.com/fabiocastilhoss/StepAnalyzerPAT2Math.git`. Os códigos foram executados por meio do *Google Colaboratory* e são disponibilizados no *GitHub*, com a extensão .ipynb.