UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

LUCAS OLIVEIRA

APRENDIZADO PROFUNDO PARA PÓS-EDIÇÃO DE SAÍDAS DE SISTEMAS DE RECONHECIMENTO AUTOMÁTICO DE FALA

MEDIANEIRA

2022

LUCAS OLIVEIRA

APRENDIZADO PROFUNDO PARA PÓS-EDIÇÃO DE SAÍDAS DE SISTEMAS DE RECONHECIMENTO AUTOMÁTICO DE FALA

DEEP LEARNING FOR POST-EDITING OF AUTOMATIC SPEECH RECOGNITION SYSTEM OUTPUTS

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Jorge Aikes Junior

Coorientador: Prof. Dr. Arnaldo Candido Junior

MEDIANEIRA

2022

LUCAS OLIVEIRA

APRENDIZADO PROFUNDO PARA PÓS-EDIÇÃO DE SAÍDAS DE SISTEMAS DE RECONHECIMENTO AUTOMÁTICO DE FALA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 21/novembro/2022

Alan Gavioli
Professor Doutor
Universidade Tecnológica Federal do Paraná

Evando Carlos Pessini
Professor Doutor
Universidade Tecnológica Federal do Paraná

Jorge Aikes Junior
Professor Doutor
Universidade Tecnológica Federal do Paraná

MEDIANEIRA

2022

Dedico este trabalho à minha família pela contribuição na formação do meu caráter, aos meus amigos pelos momentos vividos, a minha namorada por sempre ser minha parceira e a minha mãe pelo melhor exemplo de força, dedicação e amor que tive.

AGRADECIMENTOS

Ao longo dos anos em que vivenciei o dia a dia dentro de uma universidade fui cercado por pessoas que sempre me ajudaram. Esse espaço é dedicado a lembrar dessas pessoas e deixa-las marcadas em minha história.

Agradeço primeiramente aos meus professores que, independente da situação, sempre se dedicaram em transmitir conhecimento da melhor forma possível. Sem essa dedicação eu não seria capaz de construir o conhecimento necessário para me tornar um profissional melhor.

Em especial, agradeço ao Prof. Dr. Arnaldo Candido Junior, pela sabedoria em me instruir dentro dos temas referentes ao meu trabalho de conclusão de curso, além de ter sido o primeiro a me apresentar os conceitos dos quais eu atuo no dia a dia como profissional.

Agradeço aos meus amigos e colegas de sala e universidade. São pessoas das quais eu sempre vou ter muito carinho e espero ter contribuído ao máximo do que pude durante o período em que compartilhamos experiências e aprendemos muito juntos.

Agradeço à minha família, por me auxiliar na mudança de estado por conta da universidade e nas necessidades que surgiram ao longo dos meus anos de estudo, sempre me ajudando a manter meu foco no que realmente era importante.

Em especial agradeço a minha mãe, Telma Resende de Oliveira, que infelizmente não acompanhou os meus anos dentro da universidade, mas que sempre lutou muito para que eu pudesse ter condições de realizar esse sonho. Sei que ela sempre esteve e está comigo e espero que essa conquista a encha de orgulho e certeza da incrível mulher e mãe que ela foi.

Por fim, agradeço também ao Centro de Excelência em Inteligência Artificial e ao Ministério da Educação por me auxiliar durante os anos de participação na Iniciação Científica, além de me fornecer espaço para compartilhar e aprender com profissionais qualificados.

RESUMO

Através do uso de Redes Neurais Artificiais é possível realizar o reconhecimento de voz de forma automática, possibilitando a conversão de falas em texto. Esse artifício está presente em diversos dispositivos, como celulares inteligentes e assistentes pessoais, além de ser amplamente utilizado. Redes Neurais Artificiais possuem uma quantidade de nós que estão organizados em camadas, as quais estão divididas entre camada de entrada, camada oculta e camada de saída. Esses nós representam os neurônios e realizam o processamento da informação. Neste trabalho foi realizada a construção de uma rede neural artificial que tinha como objetivo realizar o aprimoramento da saída de um sistema de reconhecimento automático de voz aplicado em áudios, o qual resultou como saída uma base de dados contendo frases do Português Brasileiro. Para cumprir o objetivo proposto Redes Neurais Recorrentes do tipo Long Short-Term Memory foram utilizadas, pois apresentam como diferencial o fator de possuírem memória, ou seja, de carregarem, durante o processo de treinamento, informações passadas em momentos anteriores e fazerem uso dessas informações para determinar novos valores. Experimentos foram realizados utilizando bases de dados que vieram de sistemas de reconhecimento automático de voz com o objetivo de aprimorar a rede para que ela possa cumprir o propósito de ser capaz de corrigir as frases da maneira correta. Em seguida foi realizada a construção da rede final, alterando a metologia utilizada até então e obtendo resultados melhores.

Palavras-chave: rede neural; reconhecimento de voz; memória.

ABSTRACT

Through the use of Artificial Neural Networks it is possible to perform voice recognition of automatically, allowing the conversion of speeches into text. This artifact is present on various devices, such as cell phones, smartphones and personal assistants, in addition to being widely used. Artificial Neural Networks have a number of nodes that are organized into layers, which are divided into input layer, hidden layer and output layer. These nodes represent neurons and intensify information processing. In this work, the construction of an artificial neural network that had with the aim of improving the output of an automatic recognition system of voice applied in audios, which resulted as output a database containing phrases of Brazilian Portuguese. To fulfill the proposed objective Current Neural Networks of the Long Short-Term Memory type were used, as they present as a differential the have memory, that is, to carry, during the training process, information past at previous times and make use of this information to determine new values. Experiments were performed using databases that came from systems of automatic voice recognition in order to improve the network so that it can fulfill the purpose of being able to correct the sentences in the correct way. Then the construction of the final network was carried out, changing the methodology used until then and obtaining better results.

Keywords: neural network; speech recognition; memory.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

No ano de 1930, pesquisadores dos Laboratórios Bell propuseram o primeiro modelo para análise e síntese de fala com o objetivo de aperfeiçoar sistemas de comunicação (JUANG; RABINER, 2005). Desde então o uso de sistemas de reconhecimento automático de fala tem se tornado comum e diversos sistemas que estão presentes na sociedade se aproveitam deste recurso, como as assistentes pessoais Siri, Alexa e a geração de legendas automáticas do Youtube. No ano de 2018 foi feita uma pesquisa que apontou que em 2022 a visualização de vídeos será responsável por 82% do tráfego da Internet (CISCO, 2018) acarretando no aumento da demanda no uso de sistemas de reconhecimento automático de fala (Automatic Speech Recognition - ASR).

Conforme este cenário aumenta de forma progressiva, os sistemas que realizam a conversão automática de fala para texto necessitam de melhorias. Por conta dessa demanda excessiva no uso de sistemas de reconhecimento de fala é que o aprimoramento das saídas geradas utilizando redes neurais recorrentes é desejável, tendo como objetivo reduzir a Taxa de Palavras Erradas que um modelo gera e tornando assim seu resultado mais útil para a resolução de problemas nos diversos campos possíveis em que essa solução é aplicável, como assistentes pessoais, geração de legendas automáticas e adaptação de vídeos para pessoas com deficiência auditiva. Quanto mais próximo estiver o resultado de sistemas de ASR da transcrição humana maiores serão as possibilidades de uso dessa ferramenta.

Uma das formas de melhorar os resultados obtidos por sistemas ASR são as Redes Neurais Recorrentes (Recurrent Neural Network - RNNs) que possuem a capacidade de realizar a previsão de caracteres dentro das frases utilizando-se de métodos matemáticos e estatísticos com base na entrada inserida na camada inicial, reformulando as frases geradas e corrigindo os erros encontrados (ZHANG *et al.*, 2021). RNNs também foram úteis para outras tarefas como monitorar condições em sistemas de usinas nucleares (ŞEKER; AYAZ; TÜRKCAN, 2003) e previsão de radiação solar (PANG; NIU; O'NEILL, 2020). A base de dados utilizada para treinar a RNN é o que determina seu resultado e traz sentido à diversidade no uso dessa ferramenta. Por exemplo, o sistema de Gris *et al.* (2021) usou uma base de dados para criar um sistema responsável por realizar o processo de transcrição de áudios em Português. Esse sistema comete erros que poderiam ser mitigados usando-se pós-processamento, aplicando corretor ortográfico para corrigir palavras com grafia inconsistente.

O modelo de Gris *et al.* (2021) foi utilizado para rotular uma base previamente analisada por humanos. Isso permitiu a obtenção de pares de frase correta-gerada com um total de 12 milhões de amostras. As amostras são definidas como a saída gerada pelo ASR e a frase escrita da maneira correta conforme as regras ortográficas. Por exemplo, uma frase gerada pelo sistema obtida a partir das amostras foi "dados não trouxeram uma necessidade muito latente de atendimento desse público e aqui tem a fa" e a versão de referência da mesma frase "dados não trouxeram uma necessidade muito latente de atendimento desse público e aqui tem a fase".

1.1 Objetivo geral e específicos

O objetivo geral deste trabalho consiste no aprimoramento das saídas obtidas de sistemas que realizam o reconhecimento automático de fala. Com base no objetivo geral mencionado anteriormente é possível identificar os seguintes objetivos específicos:

- Ajuste na base de dados fornecida pelo sistema ASR (GRIS *et al.*, 2021) organizando as frases geradas em Português Brasileiro pelo modelo de reconhecimento automático de fala;
- Implementação de uma Rede Neural Recorrente (RNN) para previsão de caracteres e treinamento do modelo gerado utilizando a base de dados adaptada;
- Comparação entre o resultado gerado pelo modelo treinado e a entrada original dos dados.

1.2 Justificativa

Dentre as tecnologias que vêm ganhando notoriedade estão as redes neurais que possuem grande capacidade de resolução de diversos problemas por conta da sua versatilidade, como detecção de padrões de fluxo multifásico (AL-NASER; ELSHAFEI; AL-SARKHI, 2016), reconhecimento da escrita coreana Hangul manuscrita (KIM; XIE, 2015) e super-resolução de imagem única (DONG *et al.*, 2015). Existem diversas formas de se utilizar redes neurais e uma delas acontece no campo do reconhecimento de voz.

Segundo a CISCO (2018), 3 trilhões de minutos de conteúdo de vídeo cruzarão a internet no mundo todo a cada mês até 2022, tendo como consequência a demanda excessiva no que diz respeito a reconhecimento de voz automático para geração de legendas. Por conta do aumento crescente no uso de métodos que utilizam a voz como meio de comunicação é que o aprimoramento dos resultados obtidos por sistemas de reconhecimento automático de fala é necessário e deve ser aprimorado constantemente.

Ao aprimorar o resultado gerado pelos sistemas ASR será possível implementá-los de forma expansiva para melhorar as soluções já existentes, como a geração de legendas automáticas para pessoas que possuem deficiência auditiva, uso de assistentes digitais automatizadas que realizem o reconhecimento de voz para orientação dos usuários e também no setor de entretenimento que faz uso constante de voz para comunicação, além de impactar positivamente a área de inteligência artificial devido ao fato do resultado obtido deste estudo ficar disponível para análises e aplicações futuras. Por exemplo, a análise de sentimentos em áudios permite aplicar esse tipo de pesquisa em chamadas de emergência e definir quando existem emoções na voz que sinalizam perigo com base na análise realizada. O aprimoramento de sistemas ASR acarreta em vantagens para empresas que realizam o atendimento ao consumidor por meio dessa

ferramenta, pois otimiza esse processo, diminui custos e reduz os procedimentos necessários durante o atendimento.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos e fundamentos de Aprendizado de Máquina (Seção 2.1) e Redes Neurais Artificiais (Artificial Neural Networks - ANNs) (Seção 2.2). Em seguida, tem-se como foco abordar os principais modelos de Redes Neurais que compõem o cenário de Inteligência Artificial (Artificial Intelligence - AI) e suas funcionalidades. Por fim, são apresentados alguns modelos de ASR e a metodologia utilizada para correção da base de dados fornecida.

2.1 Aprendizado de Máquina

Segundo Goodfellow, Bengio e Courville (2016), um algoritmo de Aprendizado de Máquina é um algoritmo que está apto a aprender com os dados. É possível acrescentar outras definições para este conceito, como o campo de Aprendizado de Máquina ser capaz de fornecer aos computadores a habilidade de aprender sem ser explicitamente programado (SAMUEL, 1959).

De acordo com Bonaccorso (2017), “aprender pode ser descrito como a habilidade de mudar de acordo com estímulos externos e lembrar da maioria das experiências anteriores". O objetivo principal da aplicação de algoritmos de Aprendizado de Máquina está em estudar, projetar e melhorar modelos matemáticos que podem ser treinados com dados para cumprir determinada tarefa, que será resultante dos dados inseridos na entrada, sem possuir o conhecimento completo de todos os elementos que influenciam essa ação.

Algoritmos de Aprendizado de Máquina podem ser classificados como supervisionados e não supervisionados com base no tipo de experiência que é fornecida durante o processo de aprendizado (GOODFELLOW; BENGIO; COURVILLE, 2016). Algoritmos não supervisionados trabalham com bases de dados que possuem muitos recursos e aprendem propriedades úteis dessa estrutura, descobrindo padrões e regularidades que devem ser exibidas nas saídas geradas. Algoritmos supervisionados trabalham com bases de dados também contendo recursos, porém cada instância é associada a um rótulo ou alvo.

Soluções resultantes da aplicação do Aprendizado de Máquina tem sido muito utilizadas em diversos cenários, como previsão de informações, classificação e agrupamento. Algoritmos que realizam a classificação e a regressão de dados fazem parte do grupo de supervisionados. A classificação possui um rótulo discreto a ser predito e busca determinar a qual classe a instância estudada faz parte. A regressão utiliza um rótulo contínuo a ser predito e utiliza-se de dados numéricos para estipular o valor de um campo escolhido (FACELI, 2011). Algoritmos que realizam o agrupamento, associação e a sumarização das informações fazem parte do grupo de não supervisionados. O agrupamento divide os dados em grupos com base em sua semelhança, a sumarização busca descrever de forma simples e compacta um conjunto de dados

e a associação procura padrões frequentes de associações entre os atributos do conjunto de dados.

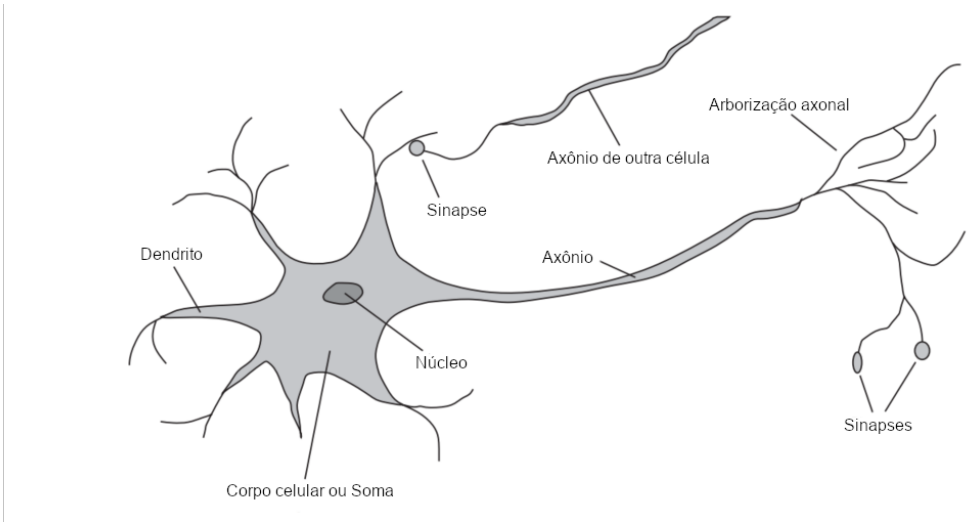
2.2 Redes Neurais Artificiais

Redes Neurais Artificiais são um tipo de algoritmo de Aprendizado de Máquina (GOODFELLOW; BENGIO; COURVILLE, 2016) e foram criadas com base nas redes neurais existentes no cérebro humano. O cérebro é um computador altamente complexo que tem a capacidade de processar informações paralelamente e organizar seus neurônios para realizar tarefas muito mais rápido que o melhor computador existente (HAYKIN, 2010). Neurônios são definidos como células nervosas que estão no cérebro e reagem a estímulos sensoriais, sejam elétricos, químicos ou ópticos (RUSSELL; NORVIG, 2002). É por meio de cadeias que o cérebro consegue realizar suas funções, emitindo estímulos que são passados entre os neurônios. As ANNs consistem em redes de neurônios artificiais inspiradas no funcionamento do cérebro e, por consequência, conseguem a habilidade de aprender. Através dos estímulos corretos, é possível ensinar uma máquina a cumprir uma função, desde que os estímulos aplicados estejam de acordo com a saída esperada.

2.2.1 Neurônio Biológico

O neurônio consiste em um corpo celular chamado de soma e possui um núcleo, extensões menores situadas nas extremidades do seu corpo celular chamadas de dendritos, que são responsáveis por receber estímulos de outros neurônios, e uma extensão mais longa que possui o nome de axônio, responsável por propagar a saída do soma para outros neurônios. A Figura 1 traz a representação de um neurônio biológico.

Figura 1 – Neurônio Biológico



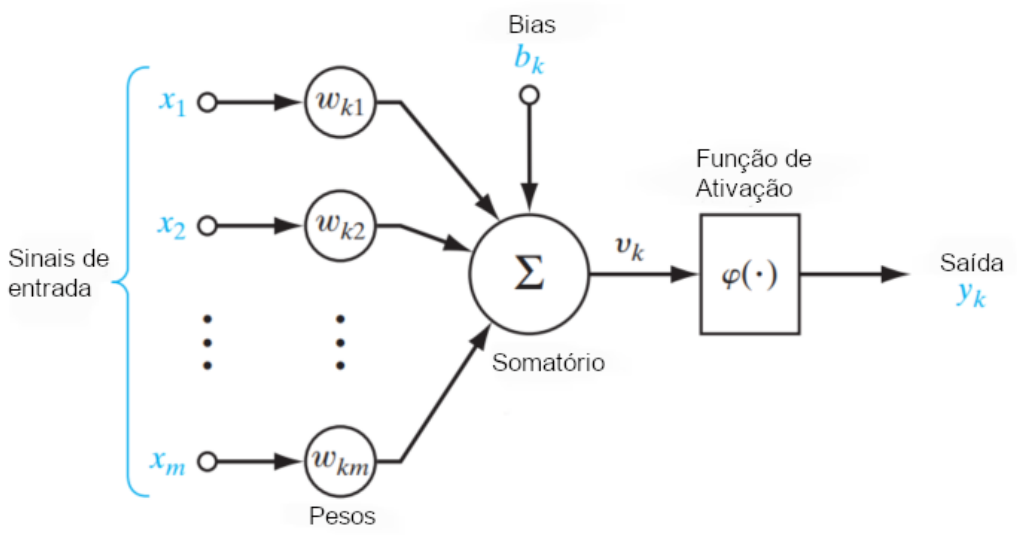
Fonte: Adaptado de Russell e Norvig (2002).

A comunicação entre os neurônios ocorre por meio de sinapses, que são sinais propagados de um neurônio para o outro por meio de reações eletroquímicas. Por meio desses sinais é que o cérebro controla sua atividade a curto prazo e também permite mudanças a longo prazo na conexão dos neurônios. É por meio desse sistema que o cérebro consegue aprender (RUSSELL; NORVIG, 2002).

2.2.2 Neurônio Artificial

Neurônios artificiais possuem uma estrutura organizada em sequência e, em sua estrutura inicial, são formados por um conjunto de valores que são inseridos em sua camada inicial, e multiplicados por um valor associado a cada entrada chamado de peso. Esse conjunto pode ser comparado às sinapses que acontecem no cérebro, os sinais de entrada podem ser comparados aos estímulos oriundos de axônios de outros neurônios. Os resultados da multiplicação são somados e armazenados em uma variável que será utilizada para a próxima etapa. Após o somatório uma função de ativação é aplicada, limitando a saída do neurônio (HAYKIN, 2010). A Figura 2 demonstra a estrutura do neurônio artificial.

Figura 2 – Representação do Neurônio Artificial



Fonte: Adaptado de Haykin (2010).

Na Figura 2 do neurônio k existe a representação de um valor externo, chamado de *bias*. Esse parâmetro pode ser negativo ou positivo e altera o valor resultante do somatório, incrementando ou decrementando sua saída (HAYKIN, 2010). É possível descrever o neurônio representado na Figura 2 com base nas equações matemáticas Equação 1 e Equação 2, adaptadas de Haykin (2010):

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{1}$$

A Equação 1 realiza o somatório dos valores x_j inseridos na camada de entrada multiplicados pelo peso atribuído w_{kj}, percorrendo as entradas e pesos do primeiro valor até o último com base no valor de m, sendo o número de atributos inseridos. Esse processo constitui uma combinação linear e guarda o resultado em u_k.

$$y_k = \varphi(u_k + b_k) \tag{2}$$

A Equação 2 aplica a função de ativação φ no resultado da soma entre a saída da Equação 1 u_k e o valor do *bias* b_k. Por fim a saída do neurônio é armazenada em y_k.

2.3 Funções de Ativação

Desenvolvido entre 1950 e 1960 por (ROSENBLATT, 1961), o Perceptron é um algoritmo que possui a estrutura mencionada na Seção 2.2.2 . É considerado um dos mais importantes algoritmos dentro da história do Aprendizado de Máquina, pois foi utilizado como base para pesquisas que desenvolveram outros algoritmos. O Perceptron utiliza-se da função de ativação como fase final do processamento da informação para determinar a saída do neurônio. Nesta seção serão apresentadas as principais funções de ativação.

2.3.1 Função de Limiar

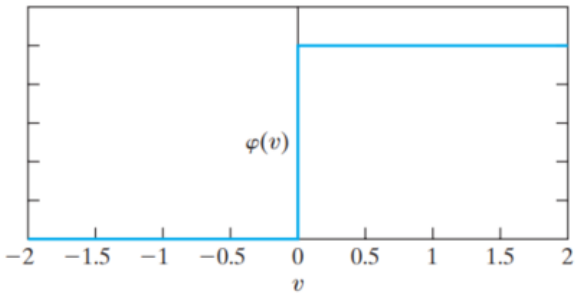
A Função de Limiar possui uma estrutura lógica de resultado binário que é baseada na saída do neurônio. Sua lógica pode ser explicada com base na Equação 3 (HAYKIN, 2010):

$$\varphi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{3}$$

O resultado da soma entre o *bias* b_k e a saída da Equação 1 u_k, que está sendo representado por x, vai resultar em duas possíveis saídas. Valor 1 caso a solução armazenada em x seja maior ou igual a 0, valor 0 caso a mesma solução seja menor que 0 (HAYKIN, 2010). A Figura 3 exibe o gráfico resultante da Função de Limiar com base na mudança do valor de x.

Apesar de possuir uma lógica simples de se entender, a Função de Limiar tem sua relevância no cenário de Aprendizado de Máquina. O trabalho elaborado por Golpour *et al.* (2021) comprova a afirmação anterior, pois faz uso dessa função para contribuir na predição de compostos fenólicos totais e fatores de atividade antioxidante.

Figura 3 – Gráfico da Função de Limiar



Fonte: (HAYKIN, 2010).

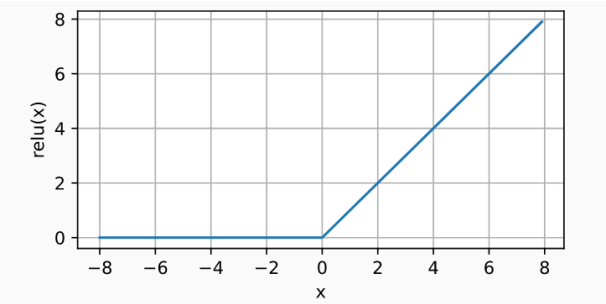
2.3.2 Função Relu

Segundo Academy (2019) e Zhang *et al.* (2021) a Função ReLU é uma das mais populares e utilizadas na elaboração de ANNs atualmente. Este sucesso acontece por conta de algumas características que essa função possui. O fato de ser uma função não linear faz com que sua aplicação possa ser feita em todas as camadas da rede (ACADEMY, 2019). A Função ReLU pode ser explicada com base na Equação 4:

$$\varphi(x) = MAX(0,x) \tag{4}$$

Aplicando-se a função de maximização MAX é que a função ReLU isola os valores negativos e, nesses casos, tem como saída final o valor 0, desativando parte dos neurônios da rede. Qualquer valor positivo e maior que 0 inserido em x terá como consequência uma saída positiva. A Figura 4 traz a representação gráfica da função ReLU com base na variação do valor de x:

Figura 4 – Gráfico da Função ReLU



Fonte: (ZHANG *et al.*, 2021).

O trabalho de Schmidt-Hieber (2020) aplica a função de ativação ReLU no desenvolvimento de uma rede neural profunda que realiza regressão não paramétrica.

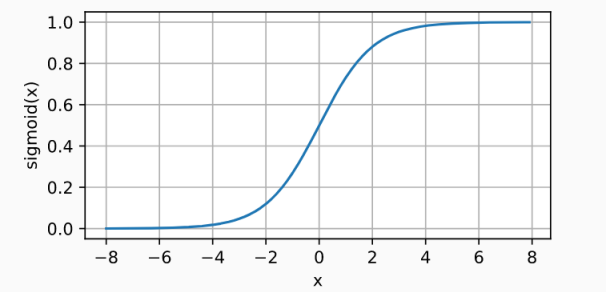
2.3.3 Função Sigmoid

Assim como a função de ativação ReLU (Seção 2.3.2), a função Sigmoid também é bastante popular no mundo do Aprendizado de Máquina. Essa função limita sua saída dentro do intervalo de 0 a 1 independente do valor de entrada inserido, tornando-a muito atrativa para problemas que necessitam realizar classificações binárias (ACADEMY, 2019). A estrutura dessa função pode ser explicada com base na Equação 5, adaptada de (ZHANG *et al.*, 2021):

$$\varphi(x) = \frac{1}{1 + exp(-x)} \tag{5}$$

Aplicando-se a função ReLU representada por φ no valor de x que representa o valor inserido na entrada, se restringe o valor final entre 0 e 1. A Figura 5 exibe o gráfico da função Sigmoide sendo aplicada com base na variação do valor de x:

Figura 5 – Gráfico da Função Sigmoid



Fonte: (ZHANG *et al.*, 2021).

A função ReLU é utilizada no trabalho de Wanto *et al.* (2017) para previsão da densidade populacional na Indonésia.

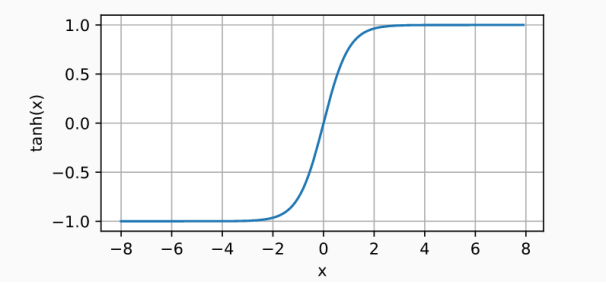
2.3.4 Função Tangente Hiperbólica

A função Tangente Hiperbólica é conhecida por ser muito semelhante a função Sigmoid (Seção 2.3.3), tendo como diferença o fato de ser simétrica ao redor da origem (SHARMA; SHARMA; ATHAIYA, 2017). Essa função insere os valores de entrada no intervalo de -1 até 1, além de ser uma função não linear. Por esse motivo a Função Tangente Hiperbólica é conhecida por corrigir o problema de resultar somente em valores positivos, pois nem sempre é desejável que os valores passados para os próximos neurônios sejam todos positivos, fator esse existente na Função Sigmoid (Seção 2.3.3). Sua estrutura pode ser explicada com base na seguinte equação:

$$\varphi(x) = \frac{1 - exp(-2x)}{1 + exp(-2x)} \tag{6}$$

E o gráfico resultante da aplicação da Equação 6 está sendo representado pela Figura 6:

Figura 6 – Gráfico da Função Tangente Hiperbólica



Fonte: (ZHANG *et al.*, 2021).

Dentre os trabalhos que aplicam a Função Tangente Hiperbólica está o de Elyassami *et al.* (2022), o qual detecta fraude financeira digital com base em ANNs.

2.3.5 Função Softmax

A função Softmax é um tipo de função Sigmoide, mencionada na Seção 2.3.3, porém com a possibilidade de trabalhar com mais de dois valores, colocando-os dentro do intervalo que vai de 0 até 1 (ALLA; ADARI, 2019). Essa função possibilita trabalhar com problemas de classificação multiclasse, ou seja, quando as opções de classificação possuem mais de duas classes. A Equação 7, adaptada de Academy (2019), demonstra a função Softmax, sendo e^{z_j} a transformação de z_j no intervalo de $0,1$ dividido pela soma dos outros valores, representado por $\sum_{k=1}^{K} e^{z_k}$, com j partindo de 1 até K, que representa o total de valores estudados.

$$\varphi(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}; j = 1,2,...,K. \tag{7}$$

2.4 Funções de Custo

No cenário do Aprendizado de Máquina é necessário otimizar o algoritmo, buscando a melhoria de seus resultados. Para cumprir essa função utiliza-se a função objetivo, que é caracterizada por validar o conjunto de pesos utilizados durante o treinamento de uma rede neural, buscando unificar todas as características do problema em uma saída para resultar em um número, permitindo assim classificar e comparar as soluções (REED; MARKSII, 1999). Ao minimizar essa função ela passa a ser conhecida como função de perda, custo ou erro (GOODFELLOW; BENGIO; COURVILLE, 2016). Nesta seção serão apresentadas alguns tipos de funções de custo.

2.4.1 Erro Quadrático Médio

A função de custo Erro Quadrático Médio é conhecida por mensurar a relação entre dois valores, comparando-os entre si e fornecendo como resultado um valor que descreve o grau de

similaridade e o nível de diferença entre eles (WANG; BOVIK, 2009). A Equação 8, adaptada de Wang e Bovik (2009), representa sua lógica:

$$L(x,y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{8}$$

O cálculo demonstrado na Equação 8 consiste na diferença entre o valor previsto y_i e o valor real x_i com base na multiplicação de $1/N$, sendo N o total de instâncias da base de dados, que resultam em uma saída positiva, por estar sempre corrigindo os erros encontrados.

2.4.2 Entropia Cruzada Categórica

A Entropia Cruzada é uma função de custo que pode ser utilizada como alternativa quando se deseja tornar as saídas dos neurônios em hipóteses independentes, sendo atribuído a cada nó uma situação única e a ativação do nó sendo comparada ao nível de confiança sobre aquela hipótese (ACADEMY, 2019). Esse função compara o valor alvo com o valor resultante do neurônio e determina o quão próximo esses valores estão. A Equação 9, adaptada de Academy (2019), é utilizada para aplicar essa função:

$$L(p,q) = -\sum_{x} p(x) log\left(q(x)\right) \tag{9}$$

Com base na Equação 9 observa-se que é feito o cálculo entre p, a distribuição probabilística alvo, e q, a aproximação da distribuição alvo. O termo $p(x)$ representa a probabilidade do evento x acontecer em p e $q(x)$ sendo a probabilidade do mesmo evento ocorrer em q. Por fim obtém-se a saída final, revelando a similaridade entre os dois valores.

2.4.3 Entropia Cruzada Binária

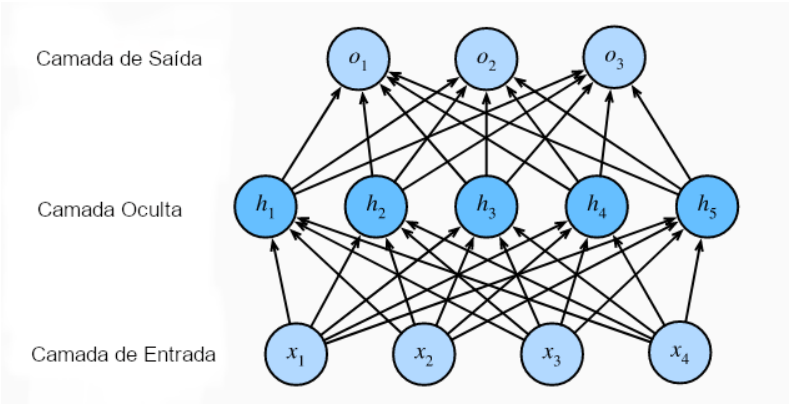
Diferente da função Entropia Cruzada Categórica mencionada na Seção 2.4.2, a função de custo Entropia Cruzada Binária permite trabalhar com problemas de classificação binária, ou seja, quando só existem duas classes para serem atribuídas a cada caso (WANG *et al.*, 2020). A Equação 10 traz a representação da função Entropia Cruzada Binária, em que y representa o rótulo, $p(y)$ a previsão da probabilidade alvo ocorrer para todos os N pontos, $log(p(y))$ sendo o logaritmo da probabilidade de ser o valor alvo e $log(1-p(y))$ sendo o logaritmo da probabilidade de não ser o valor alvo.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p(y_i)) + (1-y_i) log(1-p(y_i)) \tag{10}$$

2.5 Rede Multilayer Perceptron

Redes Multilayer Perceptron (MLP) são Redes Neurais Artificiais que utilizam um conjunto de nós interligados para criar um sistema que conecta, de forma não linear, um vetor de entrada e um vetor de saída (GARDNER; DORLING, 1998). Sua estrutura possui camadas de nós que estão conectados por pesos. Esse tipo de rede possui arquitetura variável, porém mantém em sua estrutura várias camadas de neurônios. Essa rede também é chamada de Feedforward porque a informação recebida flui por todas as camadas da rede (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 7 traz a representação de uma rede MLP.

Figura 7 – Rede Multilayer Perceptron



Fonte: Adaptado de Zhang *et al.* (2021).

A rede MLP representada pela Figura 7 possui quatro entradas x_1,x_2,x_3,x_4, na camada de entrada, cinco neurônios h_1,h_2,h_3,h_4,h_5, na camada oculta e somente três neurônios o_1, o_2,o_3, na camada de saída. Considerando que a camada de entrada não realizou nenhum cálculo para chegar no valor inserido, é determinado que essa rede possui um total de duas camadas, que estão totalmente conectadas (ZHANG *et al.*, 2021). A informação flui da camada de entrada, passando por um nó dessa camada, segue para a próxima camada, passando por todos os neurônios dessa camada e chega na camada final como uma saída, tendo seu valor alterado conforme as funções de ativação utilizadas para a rede (HAYKIN, 2010).

Esse tipo de rede possui grande importância no cenário de Aprendizado de Máquina. Redes convolucionais e recorrentes são exemplos de ANNs que surgiram com base nas redes MLP (GOODFELLOW; BENGIO; COURVILLE, 2016).

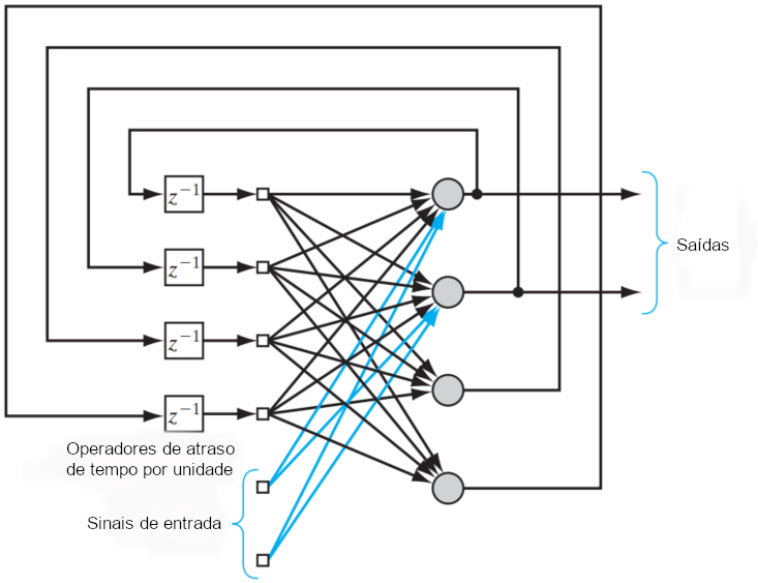
2.6 Redes Neurais Recorrentes

Redes Neurais Recorrentes (Recurrent Neural Networks - RNNs) são reconhecidas por estarem muito mais próximas do funcionamento do cérebro biológico se forem comparadas às Redes Feedforward (NIELSEN, 2015). Essa caracterização só é possível pelo fato das RNNs possuírem memória, e essa memória só existe por conta dos *loops* de *feedback* que esse tipo

de rede realiza, pois segundo Haykin (2010), uma Rede Recorrente possui pelo menos um *loop* de *feedback* que possui grande impacto na capacidade de aprendizado e na sua performance.

Diferente das Redes FeedForward, as RNNs possuem conexões que realizam ciclos (GRAVES, 2012). Esses ciclos são formados na camada oculta da rede e se baseiam na computação recorrente (ZHANG *et al.*, 2021). A camada oculta de uma Rede Recorrente realiza ciclos que partem de si para outros neurônios, além de receber o elemento de entrada do neurônio, influenciando assim as decisões tomadas com base no histórico de saídas da rede. Cada saída da rede é uma função dos membros anteriores e cada nova saída é produzida com base na mesma regra de atualização utilizada nas camadas anteriores (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 8 traz a representação de uma Rede Neural Recorrente que possui camadas ocultas e realiza ciclos.

Figura 8 – Rede Recorrente



Fonte: Adaptado de Haykin (2010).

Segundo Goodfellow, Bengio e Courville (2016), a camada oculta de uma RNN é capaz de capturar informações de camadas passadas até o momento presente. Essas correlações que acontecem com base no tempo em que a rede está processando a informação são chamadas de “Dependências de longo prazo”, isso porque o processamento no tempo depende do que aconteceu anteriormente na rede, ou seja, dos resultados já obtidos, que influenciam nas futuras tomadas de decisão (ACADEMY, 2019).

As RNNs conseguem manter a memória durante o processamento das informações através da Equação 11, adaptada de (ACADEMY, 2019).

$$h_t = \varphi(Wx_t + Uh_{t-1}) \tag{11}$$

Na Equação 11 h_t representa o estado oculto na etapa de tempo t. x_t representa uma função de entrada no mesmo tempo t do estado oculto, porém esse valor é modificado por uma

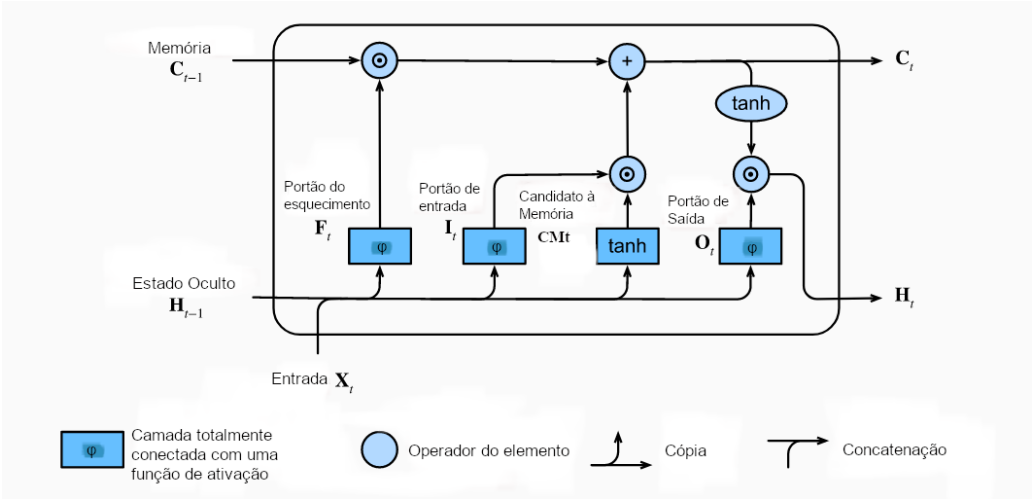
matriz de peso W que é somada ao estado oculto do passo de tempo anterior h_{t-1}, multiplicado pela sua própria matriz de estado oculto, simbolizado por U. São as matrizes de peso que determinam durante o cálculo qual o grau de importância deve ser atribuído à entrada atual e também ao estado oculto anterior. Por fim, o resultado da soma desses elementos será repassado à uma função φ, sendo normalmente a Sigmoide (Seção 2.3.3) ou a Tangente Hiperbólica (Seção 2.3.4).

2.6.1 Long Short Term Memory

Como já mencionado na Seção 2.6, as Redes Neurais Recorrentes possuem a habilidade de memorização, pelo fato de trabalharem com ciclos durante o processamento da informação passada na camada oculta, usando o contexto dessa informação entre a entrada e saída para determinar seu resultado final (GRAVES, 2012). Porém, o alcance do contexto que é utilizado na prática é limitado, acarretando na impossibilidade das RNNs tradicionais de memorizar informações por períodos maiores de tempo. Segundo Hochreiter (1998), esse fato é chamado de “Problema da Dissipação do Gradiente”. Para corrigir esse problema, a Rede *Long Short-Term Memory* (LSTM) foi criada (HOCHREITER; SCHMIDHUBER, 1997).

Redes LSTM são Redes Neurais Recorrentes que substituem a camada oculta original da RNN por uma célula de memória (HUANG; XU; YU, 2015). Essa célula consiste em um conjunto de sub-redes que são conectadas recorrentemente (GRAVES, 2012). Através dessa célula é que a rede LSTM consegue ter vantagem sobre a RNN tradicional, pois pode memorizar as informações úteis durante o treinamento, independente do tamanho da lacuna inserida nas entradas da rede (ACADEMY, 2019). Essa rede é adequada para solucionar problemas relacionados à classificação, processamento e previsão de séries temporais com intervalos de tempo desconhecidos. A Figura 9 traz a representação de um neurônio pertencente a uma rede LSTM.

Figura 9 – Neurônio da rede LSTM



Fonte: Adaptado de Zhang *et al.* (2021).

Para realizar o controle da célula de memória é preciso fazer uso de portões, ou seja, camadas de processamento criadas para realizar funções específicas. Nas Redes LSTM tradicionais, utiliza-se um portão para decidir qual informação será passada como saída para a próxima célula, chamada de portão de saída. Outro portão cumpre a função de decidir quando ler um dado dentro da célula e assim torna-lo relevante para o estado atual, chamado de portão de entrada. Por fim, existe o portão responsável por resetar informações não mais úteis ao estado da célula, chamado de portão do esquecimento (ZHANG *et al.*, 2021). Os portões permitem às redes LSTM armazenar e acessar informações ao longo do tempo, aliviando assim o problema da dissipação do gradiente (GRAVES, 2012). Os dados inseridos inicialmente nos portões são os valores de entrada no momento presente e o estado oculto oriundo do momento anterior. Esses dados são inseridos nos portões e utilizados como entrada na função Sigmoid (Seção 2.3.3), presente nos três portões. Os resultados por fim estarão dentro do intervalo de $(0,1)$ (ZHANG *et al*., 2021). As Equações 12, 13, 14, adaptadas de Zhang *et al.* (2021), representam o cálculo realizado no portão de entrada, portão de esquecimento e portão de saída, respectivamente.

$$I_t = \varphi(X_t W_{xi} + H_{t-1} W_{hi} + b_i) \tag{12}$$

$$F_t = \varphi(X_t W_{xf} + H_{t-1} W_{hf} + b_f) \tag{13}$$

$$O_t = \varphi(X_t W_{xo} + H_{t-1} W_{ho} + b_o) \tag{14}$$

Na Equação 12, I_t representa o portão de entrada, X_t o valor de entrada e H_{t-1} o estado oculto no momento anterior. Já na Equação 13 e Equação 14, F_t e O_t representam o portão de esquecimento e de saída, respectivamente. X_t e H_{t-1} são os mesmos valores inseridos no portão anterior, I_t. Já W_{xi}, W_{hi}, W_{xf}, W_{hf}, W_{xo}, W_{ho} representam os parâmetros de peso e b_i, b_f e b_o os parâmetros de *bias*. Por fim, φ representa a função Sigmoide (Seção 2.3.3).

Segundo Zhang *et al.* (2021), a célula de memória é uma ferramenta que é responsável por organizar o que será utilizado como entrada e o que será esquecido. Sua fórmula é representada pela Equação 15, adaptada de (ZHANG *et al.*, 2021).

$$C_t = F_t * C_{t-1} + I_t * CM_t \tag{15}$$

C_t representa a célula de memória no momento atual t, F_t o portão de esquecimento, C_{t-1} a célula de memória anterior, I_t o portão de entrada e CM_t a célula dos candidatos à memória, calculada pela Equação 16, em que X_t representa a entrada inserida no momento t, W_{xc} e W_{hc} sendo os parâmetros de peso, H_{t-1} representando o estado oculto no momento anterior $t-1$ e b_c o parâmetro *bias*. Por fim a função Tangente Hiperbólica (subseção 2.3.4) é aplicada, limitando a saída dos candidatos entre $(-1,1)$.

$$CM_t = tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c) \tag{16}$$

Caso o resultado da porta de esquecimento F_t for sempre próximo de 1 e o resultado da porta de entrada I_t for sempre próximo de 0, as células de memória anteriores C_{t-1}, ou seja, as informações coletadas nos tempos anteriores a t, serão salvas e repassadas para a etapa de tempo t atual.

Por fim, ocorre o cálculo do estado oculto H_t, resultado da multiplicação entre o portão de saída O_t e a aplicação da função Tangente Hiperbólica (subseção 2.3.4) na célula de memória C_t, como representado pela Equação 17, adaptada de (ZHANG *et al.*, 2021).

$$H_t = O_t * tanh(C_t) \tag{17}$$

Caso o portão de saída O_t seja próximo de 1, toda a informação de memória é passada. Caso seja próximo de 0, toda a informação é retida apenas dentro da célula de memória.

Existem diversas aplicações para redes LSTM, como reconhecimento de fala (GRAVES; JAITLY; MOHAMED, 2013), predição de séries temporais (CAO; LI; LI, 2019) e reconhecimento de escrita (CARBUNE *et al.*, 2020).

2.7 Treinamento

As Redes Neurais Artificiais passam pela fase de treinamento que é responsável por instruir a rede sobre qual caminho deve ser seguido para se alcançar o objetivo desejado. É através de métodos como a Descida do Gradiente (Seção 2.7.1) e *Backpropagation* (Seção 2.7.2) que os parâmetros da rede são alterados, resultando em saídas cada vez mais próximas dos valores objetivos (ZHANG *et al.*, 2021). Nesta seção, a Descida do Gradiente e o *Backpropagation* serão explicados.

2.7.1 Descida do Gradiente

Segundo Goodfellow, Bengio e Courville (2016), a tarefa de minimizar ou maximizar uma função $f(x)$ alterando o valor de x é chamada de otimização. Justamente é essa tarefa que é cumprida ao aplicar a Descida do Gradiente. Segundo Academy (2019), a Descida do Gradiente é um algoritmo de otimização utilizado para encontrar os valores dos parâmetros da rede, pesos e *biases*, de uma determinada função, os quais minimizem a função de custo. Esse algoritmo tem por objetivo encontrar um mínimo relacionado a uma função arbitrária.

A função a ser otimizada é chamada de função de critério ou objetivo. Ao minimizá-la, pode ser chamada de função de custo, de perda ou de erro (GOODFELLOW; BENGIO; COURVILLE, 2016). O processo se inicia atribuindo valores iniciais aos coeficientes da função.

Então esses valores são aplicados à função e é realizado o cálculo do custo, por meio de uma das funções de custo mencionadas na seção 2.4.

Então ocorre o cálculo da derivada do custo, que informará a inclinação da função em um determinado ponto. Essa inclinação será responsável por indicar o sinal correto a ser utilizado para alterar os valores dos parâmetros da função, buscando alcançar um menor valor de custo nas próximas iterações. A derivada é útil para indicar como alterar os parâmetros da função para se alcançar o menor valor global, ou seja, o resultado final mais próximo do valor alvo (GOODFELLOW; BENGIO; COURVILLE, 2016).

Um ponto que resulta no menor valor absoluto da função $f(x)$ é chamado de global mínimo, podendo existir um ou mais pontos desse tipo. Se alcança esse resultado ao obter o valor de 0, ou próximo de 0, na saída do valor de custo dos coeficientes. A Equação 18 demonstra o cálculo realizado para se atualizar os parâmetros da função na qual x' representa o novo valor do parâmetro, x o valor antigo, ε um escalar positivo que determina o tamanho da etapa é chamado de taxa de aprendizado, o qual pode ser determinado de várias formas e $\Delta_x f(x)$, representando um vetor contendo as derivadas parciais da função f.

$$x' = x - \varepsilon \Delta_x f(x) \tag{18}$$

Segundo Mitchell (1997), a Descida do Gradiente fornece a base necessária para o algoritmo de *Backpropagation*.

2.7.2 Backpropagation

O algoritmo *Backpropagation* possui grande relevância dentro do cenário das Redes Neurais Artificiais (ACADEMY, 2019) . Sua produção original aconteceu na década de 1970, porém somente no ano de 1986 este método começou a se popularizar por conta do trabalho realizado por Rumelhart, Hinton e Williams (1986). Esse algoritmo torna o treinamento de modelos de aprendizado profundo computacionalmente menos custoso, além de ser aplicado durante o aprendizado supervisionado. Utilizando a Descida do Gradiente (Seção 2.7.1), é possível acelerar o processo de treinamento em até dez milhões de vezes, em comparação à uma implementação simples (ACADEMY, 2019).

O algoritmo *Backpropagation* consiste em duas fases. A primeira fase é chamada de *Forward Pass* e realiza a passagem das entradas inseridas através da rede junto com as previsões obtidas. Esse processo varre a rede da camada inicial até a última camada. A segunda fase é chamada de *Backward Pass*, é aqui que se realiza o cálculo do gradiente, ou da derivada, da função de perda na camada final da ANN. Esse gradiente é utilizado no cálculo recursivo da Regra da Cadeia, que consiste na aplicação de derivadas parciais para a obtenção dos novos valores para os pesos da rede. O ajuste de pesos ocorre com base na Equação 18 da Seção 2.7.1 em que o novo valor do parâmetro é calculado com base na taxa de aprendizado e nas

derivadas parciais. O Algoritmo 1 traz o pseudocódigo do algoritmo *Backpropagation* no qual se inicia com a definição das variáveis que serão utilizadas no código. Em seguida inicia-se um *loop* que vai até o valor máximo das linhas da base de dados e realiza o passo do *Feedfoward* que calculando as ativações dos neurônios para cada camada. Após esse passo acontece o passo do *Backward* que calcula as derivadas da função de erro relacionada a cada camada de saída. Por fim ocorre a atualização dos pesos.

Algoritmo 1 – Pseudocódigo do Backpropagation

$Treinamento$
$X \leftarrow Base\ de\ dados\ de\ treinamento\ de\ tamanho\ MxN$
$y \leftarrow Valores\ alvo\ para\ X$
$w \leftarrow Os\ pesos\ para\ cada\ camada$
$l \leftarrow O\ numero\ de\ camadas\ da\ rede\ neural$
$D_{ij}^{(l)} \leftarrow O\ erro\ para\ todo\ l, i, j$
$t_{ij}^{(l)} \leftarrow 0. Para\ todo\ l, i, j$
$For\ i = 1\ to\ m$
  $a^l \leftarrow feedfoward(x^i, w)$
  $d^l \leftarrow a(L) - y(i)$
  $t_{ij}^{(l)} \leftarrow t_{ij}^{(l)} + a_j^{(l)}.t_i^{l+1}$
$if\ j \neq 0\ then$
  $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)} + \lambda w_{ij}^{(l)}$
$else$
  $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)}$
$where\ \frac{\delta}{\delta w_{ij}^{(l)}} J(w) = D_{ij}^{(l)}$

Fonte: Adaptado de Guo *et al.* (2021).

O algoritmo *Backpropagation* realiza portanto a atualização dos pesos com base no cálculo do erro estimado na última camada, alterando todos os pesos e tendo como ponto de partida o final da Rede Neural Artificial, ou seja, indo de traz para frente (CHAUVIN; RUMELHART, 1995). Esse método é utilizado em diversos tipos de ANN, como as Redes Perceptron Multicamadas e também as Redes Neurais Convolucionais.

2.8 Reconhecimento Automático de Fala

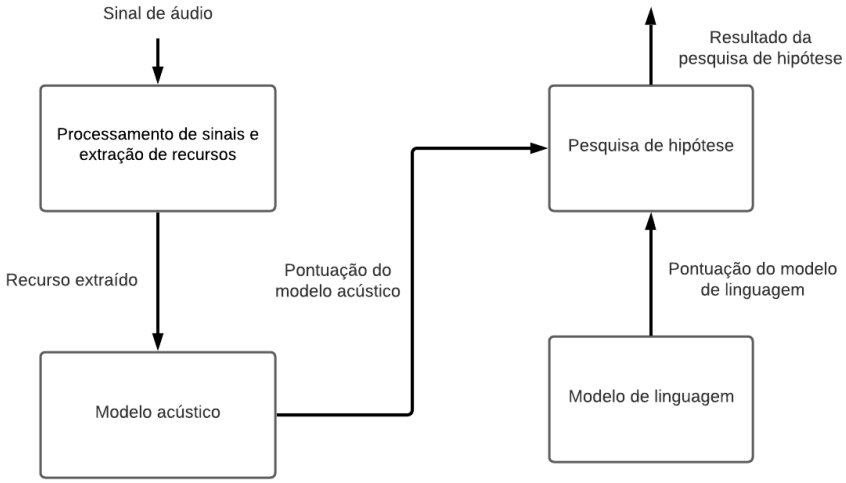
Reconhecimento Automático de Fala é um processo baseado em aprendizado de máquina que é responsável por transcrever e decodificar o discurso oral (LEVIS; SUVOROV, 2012). Através de sua aplicação esse processo traz consigo a melhoria da comunicação entre humanos e humanos e entre humanos e máquinas (YU; DENG, 2014).

Com o avanço da tecnologia a fala acabou se tornando um meio de comunicação essencial na interação entre humanos e máquinas, acarretando no aumento da necessidade de sistemas que realizam o reconhecimento de fala, tanto para transcrição de áudios quanto para auxiliar na criação de novas tecnologias relacionadas a esse tipo de comunicação.

A arquitetura de um ASR é baseada em quatro componentes principais, como representado pela Figura 10. O primeiro componente é responsável pelo processamento de sinais e

extração de recursos e realiza a melhoria da fala, removendo os ruídos e distorções de canal, altera o sinal, convertendo-o do domínio do tempo para o domínio da frequência e gera um vetor de características relevantes para o modelo acústico. O segundo é o modelo acústico, que realiza a integração entre a acústica e a fonética e gera uma pontuação para a sequência de características. O terceiro é o modelo de língua que tem por responsabilidade estimar a probabilidade de uma sequência de palavras hipotéticas com base no treinamento recebido. O quarto é a pesquisa de hipóteses, que recebe a saída do modelo de língua e a saída do modelo acústico e as combina, resultando na sequência de palavras de maior pontuação como o resultado do reconhecimento. (YU; DENG, 2014).

Figura 10 – Arquitetura de um ASR



Fonte: Adaptado de Yu e Deng (2014).

Com o objetivo de validar os resultados obtidos por um modelo ASR, deve-se atentar para a acurácia gerada pela saída do modelo e também para a velocidade de processamento que o ASR obteve (MALIK *et al.*, 2021). A velocidade é calculada utilizando a Equação 19 que representa o *Real-Time Factor*. A variável P simboliza o tempo gasto pelo modelo para processar a entrada inserida e a variável I a duração do áudio com a entrada. Caso o resultado obtido seja 1 então significa que o processamento ocorreu no chamado “Tempo Real”.

$$RTF = \frac{P}{I} \tag{19}$$

Já a acurácia do modelo ASR pode ser obtida através do cálculo da taxa de palavras erradas (*Word Error Rate - WER*) que é abordada na Seção 2.8.1.

2.8.1 Word Error Rate

Dentre as métricas utilizadas para avaliar o desempenho de um ASR encontra-se a Taxa de Palavras Erradas (Word Error Rate - WER), sendo esta uma das medidas de avaliação mais

utilizadas. Ela é responsável por realizar o cálculo da taxa do número de palavras que foram convertidas incorretamente (ALI; RENALS, 2018).

O cálculo do WER pode ser demonstrado com base na Equação 20, adaptada de (ALI; RENALS, 2018). Nesta equação a variável I representa o número de inserções, D as deleções, S as substituições e N o número de palavras na transcrição, todas essas correspondentes às frases que foram analisadas, seja inserindo, deletando ou substituindo uma nova palavra.

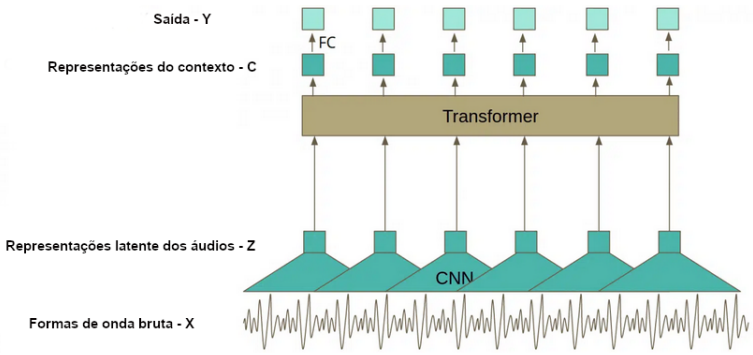
$$WER = \frac{I + D + S}{N} \times 100 \tag{20}$$

2.8.2 Wav2vec 2.0

O Wav2Vec 2.0 é considerado um dos modelos mais modernos para o reconhecimento automático de fala, devido a sua arquitetura e o modo de treinamento aplicado. Seu treinamento acontece em duas fases, a primeira fase consiste em um aprendizado auto-supervisionado utilizando dados não rotulados, tendo como objetivo obter a melhor representação da fala possível. Já a segunda fase consiste em aprendizado supervisionado *fine-tuning*, o qual os dados rotulados são usados para ensinar o modelo a prever palavras específicas ou fonemas (BAEVSKI *et al.*, 2020).

A arquitetura do modelo final utilizado para previsão possui três partes principais, como representado pela Figura 11. A primeira parte consiste em camadas convolucionais que processam a entrada na forma de onda bruta para obter a representação latente do áudio, denotada por Z. A segunda parte consiste em camadas de um transformador, responsável por criar uma representação contextualizada do áudio, denotada por C. A terceira parte consiste em uma projeção linear para a saída gerada, denotada por Y. As formas de onda bruta são representadas por X (BAEVSKI *et al.*, 2020).

Figura 11 – Arquitetura do modelo Wav2Vec



Fonte: Adaptado de Baevski *et al.* (2020).

3 MATERIAIS E MÉTODOS

Este capítulo tem como objetivo descrever os materiais que foram utilizados durante o experimento e os métodos aplicados. Bibliotecas de manipulação de dados e outras tecnologias foram utilizadas para a construção da Rede Neural Recorrente, elas estão descritas na Seção 3.1. Já os meios para o treinamento e manipulação da RNN estão descritos na Seção 3.2.

3.1 Materiais

Nesta seção os materiais utilizados estão detalhados, como o hardware da máquina responsável pelo treinamento da Rede Neural Artificial, as ferramentas necessárias para a construção da ANN e também o *dataset* implementado.

3.1.1 Hardware e tecnologias

A Tabela 1 traz as informações sobre o hardware do sistema que foi utilizado para o treinamento do modelo.

Tabela 1 – Especificações de hardware do computador

Especificações	Computador
Processador	AMD Ryzen 5 3600 6-Core
Memória Ram	8 GB
Placa de Vídeo	GeForce RTX 2080 Ti
Sistema Operacional	Ubuntu 18.04.6 LTS

Fonte: Autoria própria.

As ferramentas que foram utilizadas para o desenvolvimento básico foram as seguintes:

- Anaconda[1]: é uma distribuição das linguagem de programação Python e R;
- Python[2]: é uma linguagem de programação de alto nível, de tipagem dinâmica e forte, além de ser interpretada e possuir o paradigma orientado a objetos, funcional e imperativo. É desenvolvida pela comunidade, permitindo a criação de novas bibliotecas;
- NumPy[3]: é uma biblioteca Python com suporte para o processamento de matrizes multi-dimensionais. Possui também um conjunto de funções matemáticas de alto nível;
- Keras[4]: é uma biblioteca de redes neurais de código aberto escrita na linguagem Python. É conhecida por ser eficiente na construção de redes neurais profundas e foi absorvida pelo Tensorflow;

[1] https://www.anaconda.com
[2] https://www.python.org
[3] https://numpy.org
[4] https://keras.io

- TensorFlow [5]: é uma biblioteca de código aberto para aprendizado de máquina com capacidade de ser aplicada em diversas tarefas. Possui funcionalidades que auxiliam na construção e análise de modelos de aprendizado de máquina;
- Pandas[6]: é uma biblioteca Python para manipulação e análise de dados, possui estruturas e operações que permitem trabalhar com conjuntos de dados.
- Matplotlib[7]: é uma biblioteca da linguagem Python voltada para criação de gráficos e visualizações de dados.

3.1.2 *Dataset* Original

A base de dados fornecida é consequência da aplicação de um ASR em uma junção de bases de dados abertas populadas por áudios na língua Portuguesa e que é um conjunto de diversas fontes diferentes (GRIS *et al.*, 2021) que são as seguintes:

- CETUC (ALENCAR; ALCAIM, 2008): esta base possui cerca de 145 horas de falas que foram distribuídas entre 100 locutores, sendo 50 masculinos e 50 femininos;
- LAPSBM 1.4[8]: possui um total de 35 locutores, sendo estes 10 mulheres e 25 homens que pronunciaram 20 falas únicas cada um, totalizando 700 sentenças;
- VoxFForge[9]: consiste em um projeto aberto com o objetivo de construir bases de dados para sistema acústicos. Essa base tem cerca de 100 locutores e aproximadamente 4130 senteças foram transcritas;
- Common Voice 6.1[10]: é um projeto para criação de uma base de dados gratuita para sistemas de reconhecimento de fala. Para composição desta base voluntários doam e validam vozes e a base possui vários idiomas disponíveis. A partição em Português possui cerca de 50 horas validadas e 1120 locutores.
- Multilingual LibriSpeech (PRATAP *et al.*, 2020): essa base de dados foi construída com base em áudios de audiolivros de domínio público. Contém 6.000 horas de áudios transcritos em vários idiomas. A partição em Português possui cerca de 284 horas de fala feitas por 30 locutores.

O resultado do ASR consiste em um arquivo *CSV* com cerca de 750 MB e 4.479.000 registros. Em suas colunas estavam informações relacionadas às frases, como a base de dados

[5] https://www.tensorflow.org/?hl=pt-br
[6] https://pandas.pydata.org
[7] https://matplotlib.org
[8] https://github.com/falabrasil/gitlab-resources
[9] http://www.voxforge.org/
[10] https://commonvoice.mozilla.org/en

utilizada para treinar o ASR, o modelo aplicado, as métricas de verificação de precisão e também a frase gerada pela ANN e a frase alvo.

A Tabela 2 traz um exemplo retirado do *dataset* original contendo a frase alvo e a frase gerada pelo ASR.

Tabela 2 – Exemplo da base de dados inalterada

Modelo	Frase Alvo	Frase Prevista	WER
brexcv18k16k	eh porque na internet hoje em dia gente tem acesso a tudo seja na internet seja na televisão seja no no ah filmes desenhos	a internet hoje em dia a gente tem acesso a tudo seja na internet seja na televisão se na do a em filmes desenhos	0,375
brexcv18k16k	sou executivo da embrapa porque que eu me tornei diretor executivo da embrapa e ia embora do brasil um ano antes	executivo da embrapa porque eu me tornei diretor jecutivo da mbrapa e eu i embora do brasil um ano antes	0,285
brexcv18k16k	não pro um pobre um negro ser respeitada pela sociedade ele precisa ser rico ele precisa ter status	para um pobre um negro ser respeitado ppela sociedade ele precisa ser rico ele precisa ter estatos	0,277

Fonte: Autoria própria.

3.2 Métodos

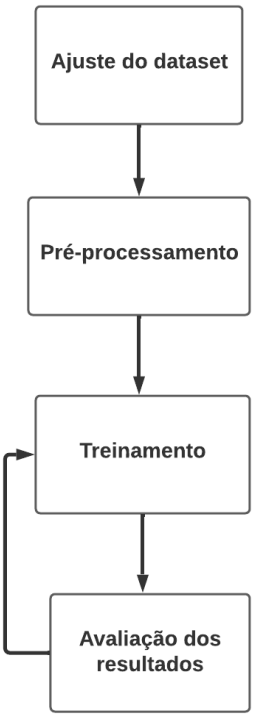
Esta seção é responsável por apresentar os métodos que foram aplicados neste trabalho. O trabalho seguiu o esquema apresentado na Figura 12.

3.2.1 Ajuste do *Dataset*

A partir da base de dados fornecida por Gris *et al.* (2021) foi possível realizar o treinamento da rede neural. Porém, foi necessário adaptá-la conforme a tarefa proposta neste trabalho. Para isso somente as colunas que contêm as frases geradas pelo ASR e as frases alvo foram utilizadas. Através de um *script* Python foram gerados dois novos arquivos, sendo um voltado para o experimento final e o outro para os experimentos preliminares. A base gerada para o experimento final era do tipo *CSV* com somente duas colunas e um total de 431.567 registros e a outra base possuía somente duas colunas também, era do tipo *CSV* e um possuía um total de 10.944 linhas.

A base utilizada no experimento final era composta por duas colunas, sendo a primeira responsável por armazenar a frase e com o símbolo “#” que a divide em duas partes, a esquerda do símbolo fica a frase que está sendo construída, a frase alvo, e a direita a frase que está sendo corrigida, a frase predita pelo ASR. A cada nova linha da base a frase é modificada, inserindo os caracteres corretos a esquerda e removendo os caracteres a direita do “#”.

Figura 12 – Fluxograma das atividades propostas



Fonte: Autoria própria (2022).

A segunda coluna contém o comando a ser executado pela rede, resultado da aplicação do cálculo da Distância de *Levenshtein*, que realiza o cálculo da distância entre palavras e encontra o resultado menos custoso para se chegar na palavra alvo (YUJIAN; BO, 2007). Esse cálculo resulta em uma sequência de comandos de inserção, exclusão, troca e não alteração de caracteres, representados na base como I[11], D[12], T[13], M[14], respectivamente. Nos casos de troca e inserção, além das letras T e I os comandos são acompanhados pelo caractere a ser trocado ou inserido, como por exemplo Ta, que significa troca pela letra a e Id, que significa insere a letra d. A Tabela 3 traz um exemplo de uma frase sendo corrigida e a Tabela 4 a representação da base de dados que foi utilizada nos experimentos preliminares, possuindo uma coluna com 100 caracteres e outra coluna com o próximo caractere a ser inserido.

3.2.2 Pré-Processamento

Para facilitar o processamento das entradas, os caracteres dos textos da base de dados foram convertidos para um vetor de números inteiros, pois assim o *framework* foi capaz de realizar a conversão dos caracteres, facilitando o processo de transformação para o vetor *one-hot*, que consiste em um vetor contendo somente 0 e 1, onde cada índice do vetor corresponde a

[11] I - Inserir
[12] D - Deletar
[13] T - Trocar
[14] M - Manter

Tabela 3 – Exemplo de correção de frase usando os comandos

Entrada	Saída
#o juau escoregou	M
o# juau escoregou	M
o #juau escoregou	M
o j#uau escoregou	To
o jo#au escoregou	Tã
o joã#u escoregou	To
o joão# escoregou	M
o joão #escoregou	M
[...]	[...]
o joão escorrego#u	M

Fonte: Autoria própria (2022).

Tabela 4 – Exemplo do dataset alterado

Texto	Próximo Caractere
em dois mil e dezoito eu decidi que eu ia apoiar empreendedores que fossem apaixonados pelo que fize	s
e que realmente acreditassem nos seus negócios não e quando eu fiquei sozinha naquela montanha que n	ã
com outra organização social chamada artemasia e essa organização ela fez um concurso de planos de ne	g

Fonte: Autoria própria (2022).

um caractere específico que é confirmado caso seu valor seja 1. Inicialmente o texto foi normalizado, removendo caracteres indesejados e também tornando as letras minúsculas. Então os caracteres do texto foram convertidos em um vetor de números inteiros, tendo como base um dicionário que relaciona cada carácter válido com um número inteiro. A lista de caracteres válidos foi de “a” até “z”, ponto final, ponto de interrogação, letras acentuadas e espaço em branco também. Essa lista foi armazenada em uma *string* que foi útil para a conversão dos valores. Além disso, pelo fato do experimento final utilizar uma coluna que possui os resultados do cálculo da distância de *Levenshtein*, ou seja, os comandos a serem executados pela rede para modificar a frase, foi necessário realizar a conversão desses comandos para valores inteiros e únicos. Portanto, cada comando passou a ser caracterizado por um valor único dentro de um vetor que continha todos os possíveis valores para a coluna do cálculo da distância de *Levenshtein*.

Por exemplo, realizando a conversão da frase “está chovendo.” e tendo como base a string mencionada com os caracteres válidos sendo “_abcdefghijklmnopqrstuvwxyzçãàáêéíóôõúûâ.?”, a conversão resultará em uma lista de valores (5,19,20,1,0,3,8,15,22,5,14,4,15,40), onde cada número inteiro representa uma única letra.

3.2.3 Treinamento

A topologia da Rede Neural Recorrente que foi utilizada nos experimentos preliminares tem como característica possuir algumas camadas, sendo a camada de entrada, oculta e de saída, as quais somente a camada oculta é do tipo RNN. A função de custo de Entropia Cruzada (Seção 2.4.2) e o otimizador Adam (ZHANG *et al.*, 2021) foram utilizados. Além disso foi aplicada a técnica de *Dropout*, que desliga parte dos neurônios da rede para evitar o *overfitting*, que acontece quando a rede acaba decorando os dados de treino e obtendo resultados satisfatórios, porém durante as fases de teste e validação ela não consegue performar bem. A taxa de aprendizado utilizada durante o processo de treinamento variou conforme os experimentos, assim como o número de épocas utilizadas.

Para o treinamento da rede, durante os experimentos preliminares, foi utilizada a base de dados representada na Tabela 4, já ajustada conforme explicado na Seção 3.2.2, a qual possui uma coluna com o texto a ser completado pela rede e outra coluna com o próximo caractere a ser inserido na frase. Dessa forma a rede foi instruída a preencher caracteres nas frases tendo como instrução o próximo elemento, corrigindo as frases incorretas e entendendo as regras para sua criação.

Já na fase de treinamento do experimento final foi utilizada a base de dados representada na Tabela 3. Esses dados possuem duas colunas, uma referente ao texto a ser corrigido utilizando o "#" para separar as frases e a outra possui o comando a ser executado pela rede, sendo este resultado da aplicação do cálculo da Distância de *Levenshtein*.

3.2.4 Avaliação dos Resultados

Após realizar o treinamento da Rede Neural Recorrente os resultados foram analisados com base nas métricas de acurácia e no resultado do cálculo do erro feito durante a aplicação do *Backpropagation*. Com base nessas duas métricas foi possível identificar durante o processo de treinamento a eficiência do modelo escolhido. Caso o resultado obtido estivesse fora do esperado então acontecia o reajuste dos parâmetros da rede e seria necessário retornar para o processo de treinamento (Seção 3.2.3), alterando os parâmetros conforme a necessidade.

Após encontrar o modelo que melhor se adapta aos padrões com os dados inseridos na fase de treinamento foram apresentados novos dados para o modelo, a fim de verificar qual seria sua precisão em corrigir essas novas frases inseridas e por fim comparar a taxa de palavras erradas gerada pelo modelo com a taxa já fornecida na base de dados. Dessa forma foi possível identificar se o modelo obteve sucesso na correção das frases.

Os resultados obtidos durante as análises realizadas podem ser encontrados no Capítulo 4.

4 RESULTADOS E DISCUSSÃO

Neste capítulo são descritos os resultados obtidos durante os experimentos realizados. Na Seção 4.1 estão os experimentos preliminares utilizando diferentes topologias para a rede LSTM e alterações na forma de uso da base de dados serão abordados. Na Seção 4.2 estão os resultados do experimento final e a avaliação do rendimento obtido pela rede.

4.1 Experimentos Preliminares

Esta seção é responsável por explicar os experimentos preliminares realizados. Cada abordagem aplicada neste capítulo possui diferentes características, tanto no número de épocas utilizadas durante o treinamento como também na topologia da rede.

4.1.1 Primeiro Experimento

Para este experimento foi utilizada a base de dados mencionada na Tabela 2, que é o resultado da aplicação de sistemas ASR em bases de dados de áudio em Português (GRIS *et al.*, 2021). Dentre as colunas observadas, somente a coluna “Frase Alvo” foi utilizada para reunir as frases alvo em um texto extenso e, assim, criar a base de dados usada como entrada neste experimento. Um script criado na linguagem Python foi aplicado para retirar as frases da base de dados original e adicioná-las na nova base, concatenando as frases para elaborar um texto corrido específico para os experimentos preliminares que possui um total de 700.312 caracteres e não faz uso do “#” para fazer a divisão das frases, como mencionado na Seção 3.2.1. A Tabela 5 traz uma parte do texto retirado da base de dados criada. A base de dados foi separada em treino, teste e validação, na qual cada uma das bases recebeu um valor diferente de caracteres. A base de treino possuiu 630.280 caracteres, a base de teste possuiu 42.019 caracteres e a base de validação possuiu 28.013 caracteres.

Tabela 5 – Exemplo do dataset criado utilizando um texto corrido

Texto
e porque o empreendedor ão assim eu comecei
o o projeto inspirado na minha família mas
o e o seu quarto aqui na sua casa como que ão
olha o quarto aqui na minha casa ão ão um
quarto mas que eu acho pequeno não mas não
um quarto legal tal e tem um espaço pequeno
tem um guarda-roupa a minha cama
minha outra cama pai totalmente paz e amor sossegado

Fonte: Autoria própria (2022).

Ao trabalhar com processamento de textos, é importante que seja estabelecido um valor padrão para a quantidade de caracteres ou palavras a serem observadas pela rede LSTM. Para este experimento foi utilizado o valor de 100 caracteres, ou seja, a rede recebeu essa quanti-

dade como valor de entrada e, com base no processamento feito, previa o próximo caractere. Além disso, a entrada utilizada, possuía um total de 100 valores inteiros que representavam os caracteres, sendo cada um desses valores únicos para cada caractere, a saída esperada possuindo somente 1 valor inteiro, o qual representou o próximo caractere e, assim, convertido para *one-hot*.

A topologia da rede LSTM, aplicada neste experimento, fez uso de *embeddings* para facilitar o processamento dos vetores que representaram os caracteres, sendo 100 o valor de entrada da camada de *embedding* (100 índices de caracteres), como representação dos caracteres e 128 o valor da saída. Em seguida, os dados foram passados para uma camada LSTM com 256 neurônios, uma função de ativação do tipo Gelu, o desligamento (*dropout*), de 20% dos neurônios da rede, e uma camada densa com saída de tamanho 45, correspondente a saída *one-hot* da rede. Além disso, foi utilizada uma taxa de aprendizado de 10^{-4}, a função de entropia cruzada categórica como função de custo e, também, o Adam como otimizador. Esses valores foram definidos empiricamente com base em experimentos preliminares. A Tabela 6 traz as informações das camadas da rede LSTM utilizada.

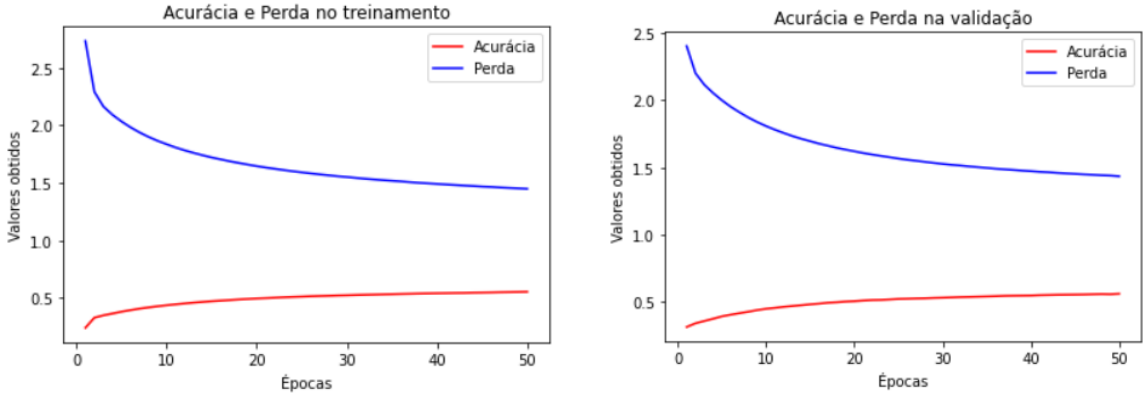
Tabela 6 – Informações sobre a rede LSTM

Camada	Camada de Saída	Parâmetro
Embedding	(None, 100, 128)	5760
LSTM	(None, 256)	394240
Ativação Gelu	(None, 256)	0
Dropout	(None, 256)	0
Densa	(None, 45)	11565

Fonte: Autoria própria (2022).

A rede LSTM foi treinada por 50 épocas utilizando um *batch size* de 512, o que permitiu com que aprendesse de maneira mais rápida. A Figura 13 traz os resultados obtidos com base no treinamento e na validação da rede utilizando a acurácia e a perda como métricas de desempenho.

Figura 13 – Resultados da rede inicial



Fonte: Autoria própria (2022).

Com base na Figura 13 é possível observar que a rede obteve um resultado que se inicia com um grande salto de aprendizado por conta da diminuição do valor da perda e o

aumento na acurácia, tanto no treinamento quanto na validação, e com o passar das épocas percebeu-se que os valores de treinamento e validação foram se estabilizando. Após o fim das 50 épocas estabelecidas, foi possível identificar que, para melhorar a avaliação das topologias, seria preciso realizar novos experimentos para refinar valores a serem passados dentro dos hiper-parâmetros.

Após o treinamento, foi feita outra avaliação na rede treinada com 50 épocas utilizando a base de dados de teste. Os resultados obtidos foram medidos utilizando a acurácia e a perda como métricas de avaliação e podem ser observados na Tabela 7.

Tabela 7 – Resultados do teste

Perda	Acurácia
1,408	0,567

Fonte: Autoria própria (2022).

Por fim, após toda a análise realizada, a rede gerou 200 caracteres com base em uma frase de 100 caracteres inserida como entrada. O resultado obtido pode ser observado na Tabela 8 e possui repetição de caracteres, gerado tanto por conta da ausência da aplicação do parâmetro de temperatura, que determina a confiança dos resultados obtidos durante a geração de caracteres, como também pelo fato da rede não ter se adaptado tão bem aos padrões presentes na base de dados, como demonstra a Tabela 7.

4.1.2 Segundo Experimento: Taxa de Aprendizado

Após analisar os resultados obtidos com base no experimento da Seção 4.1.1, foi estabelecido alterar o valor atribuído à taxa de aprendizado e manter os valores de outros hiperparâmetros, a fim de encontrar o melhor valor para aquele hiperparâmetro. Dessa forma, os experimentos relatados nesta seção foram feitos mantendo as métricas utilizadas no experimento anterior da Seção 4.1.1, alterando somente os valores da taxa de aprendizado para 10^{-2}, 10^{-3}, 10^{-5} e 10^{-6} e treinando a rede com a mesma base de dados. A Tabela 9 traz as informações das camadas da rede LSTM utilizada.

Tabela 8 – Frase gerada pela rede

Frase Completa:
Era um belo dia de sol quando joão resolveu ir ao parque. Ele juntou todas as suas coisas em uma mochila, como seu celular, lanche, mapa, bússola e sua garrafa da água. Então ele pegou um onibus e saiu de casa. Chegando ao parque ele aproveitou para praticar esportes e não demorou muito para ir desca
Frase Gerada:
a de contar a gente tem uma coisa de contar a gente tem uma coisa de contar a gente tem uma coisa de contar a gente tem uma coisa de contar a gente tem uma coisa de contar a gente tem uma coisa de con

Fonte: Autoria própria (2022).

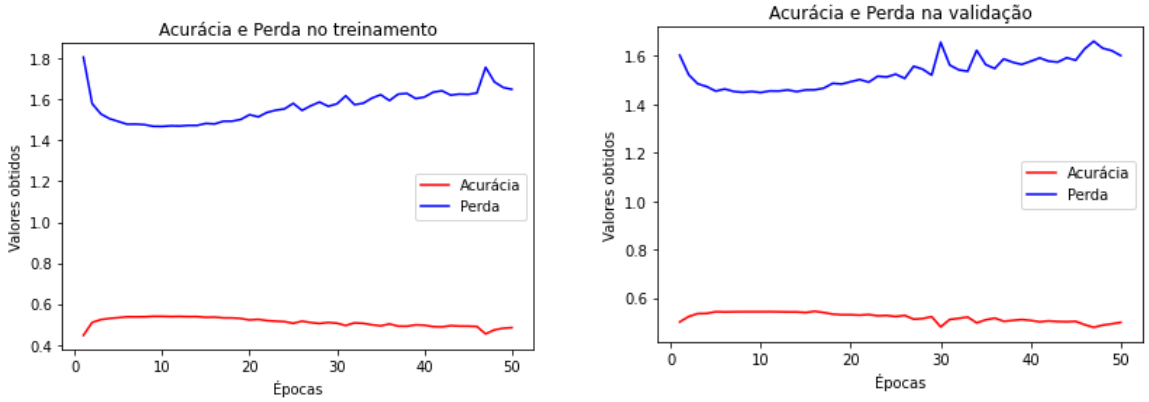
Tabela 9 – Informações sobre a rede LSTM

Camada	Camada de Saída	Parâmetro
Embedding	(None, 100, 128)	5760
LSTM	(None, 256)	394240
Ativação	(None, 256)	0
Dropout	(None, 256)	0
Densa	(None, 45)	11565

Fonte: Autoria própria (2022).

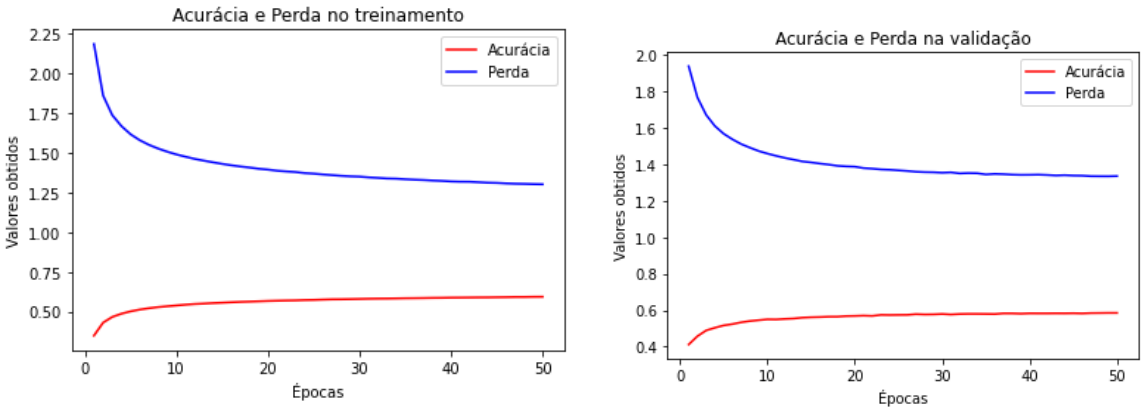
Assim como a rede mencionada na Seção 4.1.1, as redes aqui implementadas foram treinadas por 50 épocas com um *batch size* de 512. As Figuras 14, 15, 16 e 17 apresentam os resultados obtidos durante o treinamento e a validação utilizando-se as taxas de aprendizado de 10^{-2}, 10^{-3}, 10^{-5} e 10^{-6}, respectivamente.

Figura 14 – Resultados com taxa de aprendizado de 10^{-2}



Fonte: Autoria própria (2022).

Figura 15 – Resultados com taxa de aprendizado de 10^{-3}

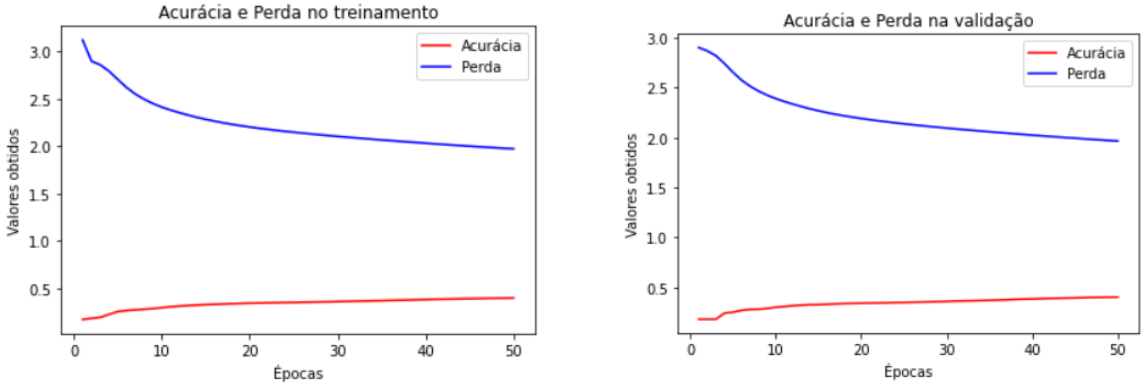


Fonte: Autoria própria (2022).

Então, assim como na Seção 4.1.1, após o final do treinamento foi feita outra avaliação nas redes treinadas por 50 épocas. A acurácia e a perda continuaram a ser utilizadas como métricas de avaliação e podem ser observados na Tabela 10 para todas as redes.

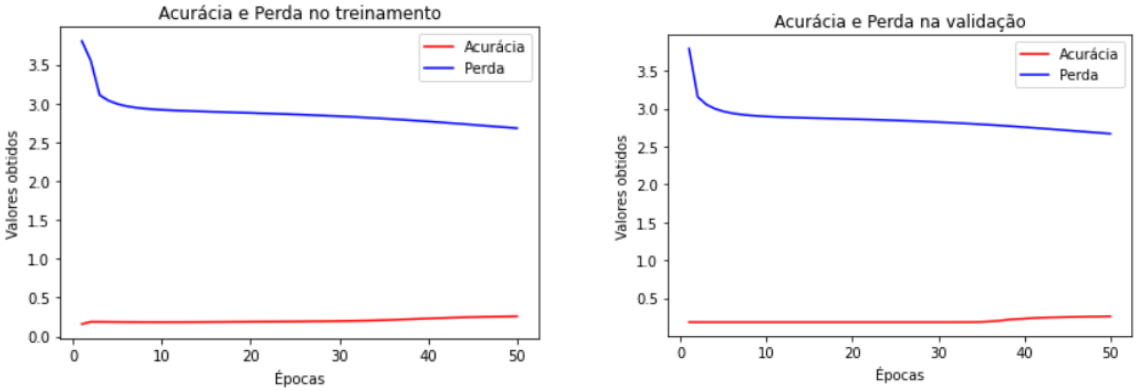
Após avaliar-se os resultados obtidos através das Figuras 14, 15, 16 e 17 e da Tabela 10 foi possível identificar que a taxa de aprendizado de 10^{-3} obteve resultados melhores se com-

Figura 16 – Resultados com taxa de aprendizado de 10^{-5}



Fonte: Autoria própria (2022).

Figura 17 – Resultados com taxa de aprendizado de 10^{-6}



Fonte: Autoria própria (2022).

Tabela 10 – Resultados dos testes

Taxa de Aprendizado	Perda	Acurácia
10^{-2}	1,572	0,508
10^{-3}	1,304	0,594
10^{-5}	1,938	0,414
10^{-6}	2,645	0,269

Fonte: Autoria própria (2022).

parada as taxas de aprendizado aplicadas nos experimentos até o momento, tendo as medidas de acurácia e perda obtidas durante a validação e o teste como parâmetros. Portanto a taxa de aprendizado de valor 10^{-3} foi mantida como padrão para os próximos experimentos.

4.1.3 Terceiro Experimento: *Embedding*

Após a realização de experimentos relacionados a taxa de aprendizado, mencionados na Seção 4.1.2, o próximo passo se baseou na aplicação de diferentes valores para o *embedding*, a fim de encontrar o valor que apresentasse os melhores resultados. Portanto foram adotados os valores de 20 e 30, além de realizar o treinamento da rede com a mesma base de dados

aplicada nos experimentos anteriores. As Tabelas 11 e 12 trazem as informações sobre as redes LSTM que utilizaram o *embedding* de valor 20 e 30 respectivamente.

Tabela 11 – Informações sobre a rede LSTM de *Embedding* igual a 20

Camada	Camada de Saída	Parâmetro
Embedding	(None, 100, 20)	900
LSTM	(None, 256)	283648
Ativação	(None, 256)	0
Dropout	(None, 256)	0
Densa	(None, 45)	11565

Fonte: Autoria própria (2022).

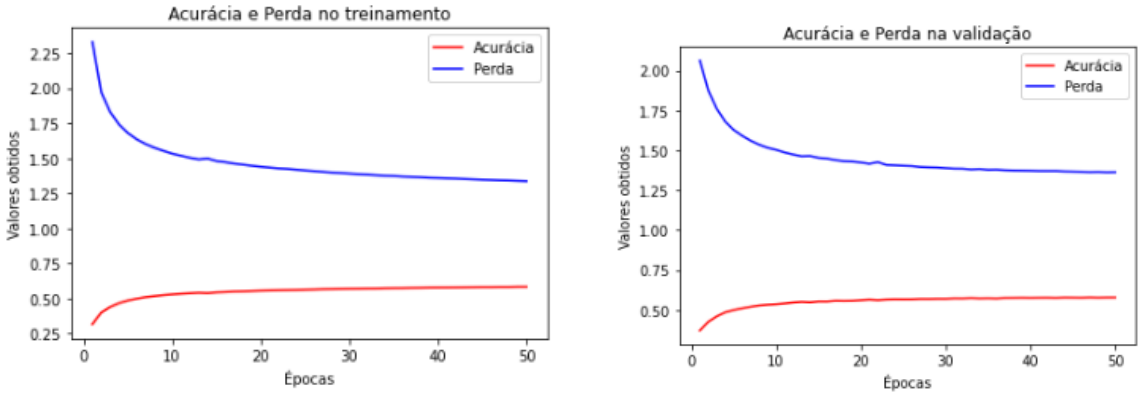
Tabela 12 – Informações sobre a rede LSTM de *Embedding* igual a 30

Camada	Camada de Saída	Parâmetro
Embedding	(None, 100, 30)	1350
LSTM	(None, 256)	293888
Ativação	(None, 256)	0
Dropout	(None, 256)	0
Densa	(None, 45)	11565

Fonte: Autoria própria (2022).

Os valores estabelecidos nas seções anteriores foram mantidos, ou seja, o treinamento aconteceu durante 50 épocas com um *batch size* de 512 e taxa de aprendizado de $10^{-}3$. As Figuras 18 e 19 trazem os resultados obtidos durante o treinamento e validação das redes que utilizaram os valores de 20 e 30 para o *embedding*, respectivamente, sendo a acurácia e a perda aplicadas como métricas de avaliação.

Figura 18 – Resultados com *embedding* de valor 20

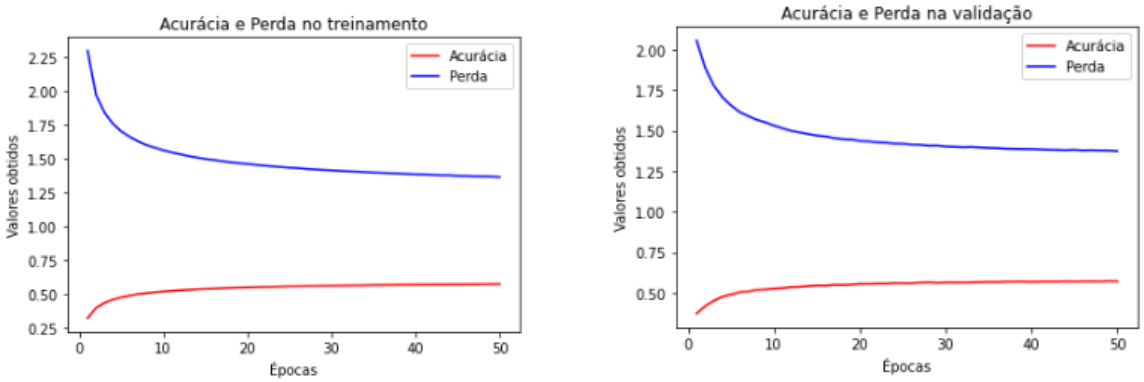


Fonte: Autoria própria (2022).

Após a validação dos resultados obtidos durante o treinamento ambas as redes foram expostas a novos dados de teste. A Tabela 13 traz os resultados obtidos com base nas métricas da acurácia e da perda para ambas as redes.

Por fim, com base nas Figuras 18, 19 e na Tabela 13 foi concluído que a alteração nos valores atribuídos ao *embedding* não resultaram em mudanças significadas, fazendo com que o valor de 30 fosse utilizado nos próximos experimentos.

Figura 19 – Resultados com *embedding* de valor 30



Fonte: Autoria própria (2022).

Tabela 13 – Resultados dos testes

Valor do *embedding*	Perda	Acurácia
20	1,332	0,585
30	1,339	0,585

Fonte: Autoria própria (2022).

4.1.4 Quarto Experimento: Aumento de Neurônios LSTM

Como último experimento preliminar, após a realização de experimentos que alteraram os valores da taxa de aprendizado e do *embedding*, foram alterados os valores referentes à quantidade de neurônios na camada LSTM da rede, buscando verificar se essa quantidade utilizada até então poderia ser melhorada. Portanto foram realizados três experimentos com diferentes valores de neurônios na camada LSTM, sendo utilizada a mesma base de dados aplicada nos experimentos anteriores e utilizando a taxa de aprendizado de 10^{-3} e o *embedding* de 30. As Tabelas 14, 15 e 16 trazem as informações referentes às redes que utilizaram os valores de 512, 600 e 700 para a quantidade de neurônios LSTM, respectivamente.

Tabela 14 – Informações sobre a rede com 512 neurônios LSTM

Camada	Camada de Saída	Parâmetro
Embedding	(None, 100, 30)	1350
LSTM	(None, 512)	1112064
Ativação	(None, 512)	0
Dropout	(None, 512)	0
Densa	(None, 45)	23085

Fonte: Autoria própria (2022).

Tabela 15 – Informações sobre a rede com 600 neurônios LSTM

Camada	Camada de Saída	Parâmetro
Embedding	(None, 100, 30)	1350
LSTM	(None, 600)	1514400
Ativação	(None, 600)	0
Dropout	(None, 600)	0
Densa	(None, 45)	27045

Fonte: Autoria própria (2022).

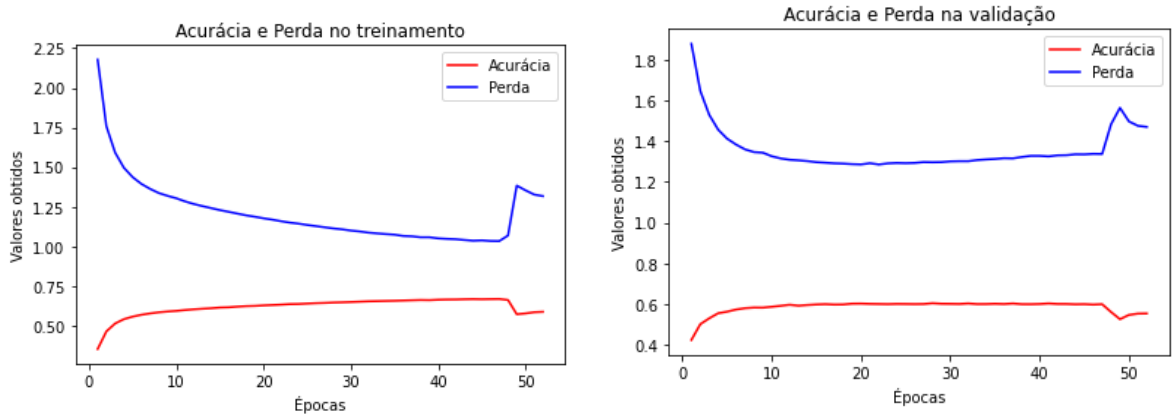
Tabela 16 – Informações sobre a rede com 700 neurônios LSTM

Camada	Camada de Saída	Parâmetro
Embedding	(None, 100, 30)	1350
LSTM	(None, 700)	2046800
Ativação	(None, 700)	0
Dropout	(None, 700)	0
Densa	(None, 45)	31545

Fonte: Autoria própria (2022).

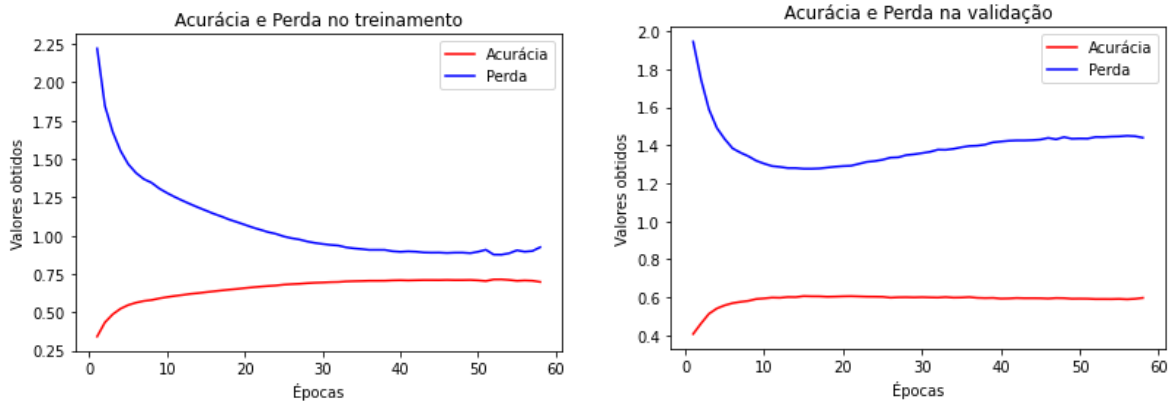
Com o objetivo de observar o comportamento da rede com o aumento do número de épocas foi utilizado o valor de 80 para a quantidade de épocas em que as redes foram treinadas e validadas, além de ter sido implementado o método da parada antecipada, que pausava o treinamento caso a rede não apresentasse melhoria nas próximas 10 épocas e armazenava a rede que obteve melhor resultado. As Figuras 20, 21 e 22 apresentam os resultados obtidos durante o treinamento e a validação das redes que utilizaram 512, 600 e 700 neurônios na camada LSTM, respectivamente, aplicando a acurácia e a perda como métricas de avaliação.

Figura 20 – Resultados com 512 neurônios na camada LSTM



Fonte: Autoria própria (2022).

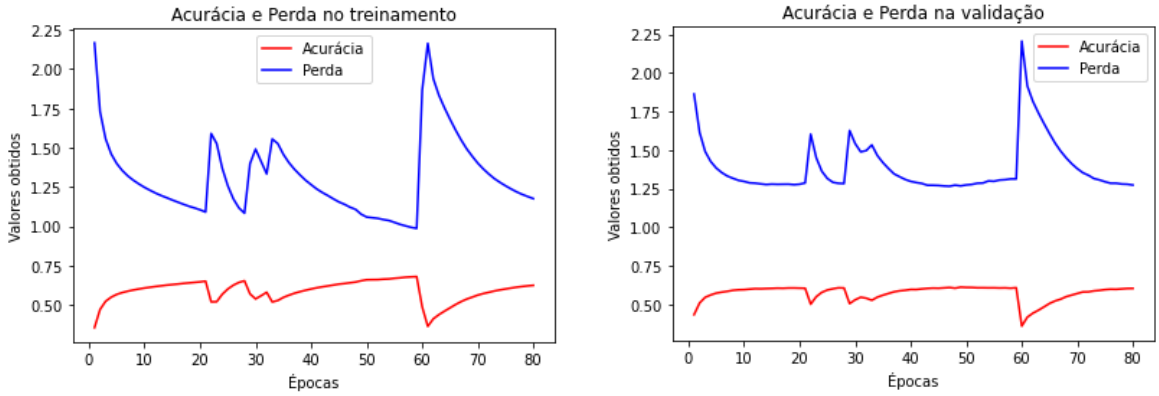
Figura 21 – Resultados com 600 neurônios na camada LSTM



Fonte: Autoria própria (2022).

Após a realização do treinamento e validação das redes, novos dados foram apresentados com o objetivo de validar o treinamento realizado e com isso foi feito um período de testes.

Figura 22 – Resultados com 700 neurônios na camada LSTM



Fonte: Autoria própria (2022).

A Tabela 17 traz os resultados obtidos das três redes estudadas neste tópico, mantendo a acurácia e perda como métricas de avaliação.

Tabela 17 – Resultados dos testes

Quantidade de neurônios LSTM	Perda	Acurácia
512	1,436	0,563
600	1,399	0,601
700	1,245	0,611

Fonte: Autoria própria (2022).

Por fim, é possível identificar que os experimentos apresentados nas figuras 20 e 21, realizados utilizando 512 e 600 neurônios na camada LSTM, apresentaram resultados interessantes quanto ao aprendizado, nos quais com o passar das épocas as redes identificavam cada vez mais os padrões da base de dados de treinamento, enquanto na fase de validação as redes aprenderam até determinada época, porém não conseguiram prosseguir com esse aprendizado. Portanto nestes casos as redes sofreram de *overfitting*, ou seja, não conseguiram aprender a prever caracteres com a precisão esperada durante o treinamento e acabaram decorando os dados, mas quando confrontadas com novos dados não foram capazes de entender os padrões ali estabelecidos, já que aprenderam somente até determinado momento.

Já a rede que utilizou 700 neurônios na camada LSTM, apresentada na Figura 22, demonstrou um comportamento inesperado durante o treinamento e validação, já que, com o passar das épocas, os valores de acurácia e perda oscilaram constantemente. Por fim, analisando a Tabela 17 a rede que utilizou 700 neurônios foi a que apresentou melhores resultados, tendo como métricas a perda e a acurácia. Porém, por conta do comportamento inconstante apresentado na Figura 22, a rede com 700 neurônios na camada LSTM não será utilizada durante o experimento final, assim como as redes que utilizaram 512 e 600 neurônios na camada LSTM também não são serão utilizadas, já que acabaram sofrendo de *overfitting*. Uma hipótese que explique o fato das redes apresentarem esse comportamento inconstante é pelo fato da medida de regularização *dropout* estar sendo utilizada, fazendo com que parte dos neurônios não sejam utilizados e a rede perca parte da informação aprendida durante o treinamento. Portanto a

rede com 256 neurônios na camada LSTM, utilizada em todos os testes anteriores, foi utilizada durante o experimento final.

4.2 Experimento Final

Após a realização dos experimentos preliminares mencionados na Seção 4.1, foi possível encontrar os valores de alguns parâmetros da rede LSTM que obtiveram os melhores desempenhos. Esses valores foram utilizados durante a criação da rede LSTM aplicada no experimento realizado nesta seção. Diferente da maneira com que os dados foram apresentados as redes até então, neste experimento a base foi pré-processada utilizando a metodologia mencionada na Seção 3.2.1, ou seja, o formato do texto apresentado foi alterado, a variável alvo também foi modificada, além do fato de que o pré-processamento dos dados foi feito com base nos resultados do cálculo da distância de *Levenshtein*. Portanto, a rede utilizada neste experimento possui uma taxa de aprendizado de 10^{-3}, valor de *embedding* de 30, 256 neurônios na camada LSTM, a função de ativação Gelu e a função de perda Entropia Cruzada, além de aplicar a regularização através do *dropout* de 20% dos neurônios e uma camada densa de tamanho 86 por conta dos valores únicos referentes aos comandos da variável alvo. A Tabela 18 apresenta as informações referentes à rede utilizada.

Tabela 18 – Informações sobre a rede do experimento final

Camada	Camada de Saída	Parâmetro
Embedding	(None, 369, 30)	2580
LSTM	(None, 256)	293888
Ativação	(None, 256)	0
Dropout	(None, 256)	0
Densa	(None, 86)	22102

Fonte: Autoria própria (2022).

Após a definição da estrutura, iniciou-se a fase de treinamento, na qual a rede foi configurada para ser treinada por 80 épocas, porém, por conta do método da parada antecipada, a rede acabou sendo treinada por 30 épocas. A Figura 23 apresenta os resultados obtidos com base nos gráficos da performance da rede durante o treinamento e validação, utilizando as métricas da acurácia e da perda.

Por fim, após a realização do treinamento e validação, a rede foi apresentada a novos dados com o objetivo de realizar um novo período de testes. A Tabela 19 traz os resultados obtidos durante essa fase, utilizando a acurácia e a perda como métricas de avaliação.

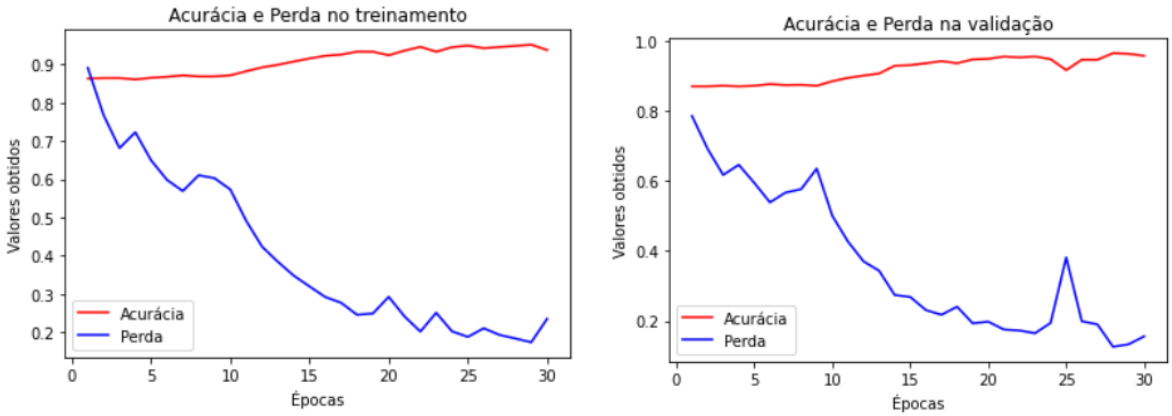
Tabela 19 – Resultados do teste final

Perda	Acurácia
0,163	0,956

Fonte: Autoria própria (2022).

Após a realização das etapas já mencionadas, uma matriz de confusão foi construída utilizando uma amostra com 6000 linhas dos dados com o objetivo de avaliar a performance do

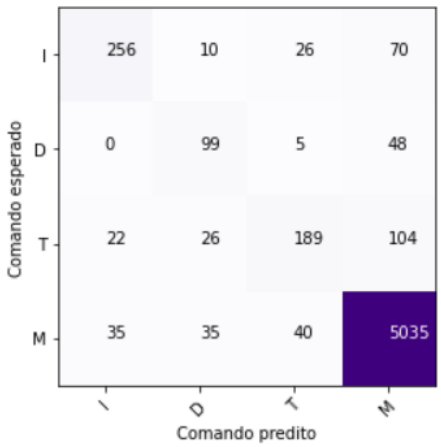
Figura 23 – Resultados da rede final



Fonte: Autoria própria (2022).

modelo utilizando outras métricas além da acurácia e da perda, além de visualizar o desempenho da rede para cada uma das opções de saída disponíveis. Foi utilizada uma amostra pelo fato de problemas técnicos, as quedas da conexão com o servidor utilizado para o treinamento da rede, terem acontecido quando tentou-se utilizar todas as linhas da base. A matriz de confusão construída possui quatro opções de saída, I para insere, D para delete, T para troca e M para mantém. O total de respostas para cada uma das opções foi contabilizado, tanto para a variável utilizada como gabarito como para a variável de predição do modelo. Cada linha da matriz representa comando da coluna alvo, enquanto as colunas representam os comandos que o modelo classificou para as frases. Através dessa matriz é possível identificar o comando que a rede mais classificou corretamente e o que ela menos conseguiu entender, além de utilizar das métricas de precisão, *recall* e *f1-score* para avaliar os resultados de maneira isolada. A Figura 24 traz a matriz de confusão da amostra dos dados que a rede classificou e a Tabela 20 apresenta as métricas de precisão, *recall* e *f1-score* obtidas.

Figura 24 – Matriz de confusão com os comandos preditos



Fonte: Autoria própria (2022).

Além disso, uma frase foi apresentada ao modelo já treinado com o objetivo de construir uma nova frase com base nas saídas obtidas. Ou seja, com base nos comandos que o modelo

Tabela 20 – Métricas da matriz de confusão

Comando	Precisão	Recall	F1-Score
I - Inserir	81,78%	70,71%	75,85%
D - Deletar	58,23%	65,13%	61,49%
T - Trocar	72,69%	55,42%	62,89%
M - Manter	95,77%	97,86%	96,80%

Fonte: Autoria própria (2022).

forneceu a frase foi reconstruída e reapresentada ao modelo, até ser totalmente corrigida. A frase inserida inicialmente foi "a caza deli", tendo como objetivo corrigi-la até chegar na frase "a casa dele". A Tabela 21 apresenta o processo de correção da frase e seu resultado.

Tabela 21 – Correção da frase com base nos comandos do modelo

Entrada	Saída do modelo
#a caza deli	M
a# caza deli	M
a #caza deli	M
a c#aza deli	M
a ca#za deli	M
a caz#a deli	M
a caza# deli	M
a caza #deli	M
a caza d#eli	M
a caza de#li	M
a caza del#i	M
a caza deli#	#

Fonte: Autoria própria (2022).

Com base nos resultados obtidos na fase de treinamento da rede, da correção da frase utilizando as saídas do modelo, da geração de uma matriz de confusão utilizando uma amostra dos dados e das métricas para avaliar os resultados da matriz, foi possível identificar que, apesar da rede apresentar valores otimistas para as métricas de acurácia e perda, segundo a Tabela 19, e um comportamento de aprendizado nos gráficos de treinamento e validação representados na Figura 23, ela não foi capaz de obter os resultados esperados para estar apta a corrigir frases com base na geração de caracteres. É possível identificar que a rede consegue entender bem os comandos para manter os caracteres, porém não se desenvolve tão bem nos outros comandos, como apontado pelas métricas presentes na Tabela 20 e na correção da frase na Tabela 21.

Observando cada comando, é possível identificar que, com base na Tabela 20 o comando com maior taxa de acerto nas métricas de Precisão, que utiliza o total de comandos que o modelo resultou como parâmetro, de *Recall*, que utiliza o total de comandos da variável alvo como parâmetro, e da *F1-Score*, que utiliza as duas métricas mencionadas anteriormente como parâmetro, foi o de Manter, já que as frases possuem muitos caracteres que serão mantidos e não precisam sofrer alterações. O comando de Inserção obteve um desempenho interessante quanto a métrica de Precisão, porém não foi tão bem dentro da métrica de *Recall* e *F1-Score*, o que significa que dentro dos resultados do modelo ele obteve uma melhor performance. Já

os comandos de Deletar e Trocar não obtiveram resultados satisfatórios com base nas três métricas. Portanto, a rede entende bem a ação de manter os caracteres, mas não se desenvolve bem nos outros comandos.

5 CONCLUSÃO

Após a realização dos experimentos propostos, foi possível concluir que o objetivo geral proposto inicialmente não foi alcançado, já que consistia principalmente em realizar o aprimoramento das saídas obtidas por sistemas que realizam o reconhecimento automático de fala, ou seja, corrigir os textos gerados por áudios utilizando aprendizado de máquina. Ao decorrer dos experimentos foram identificados parâmetros da rede neural implementada que poderiam ser alterados para obtenção de novos resultados, como a taxa de aprendizado, o número de *embeddings* aplicados e também o número de neurônios na camada LSTM da rede. Após a identificação dos melhores valores para esses parâmetros, foi implementada uma rede LSTM que fazia uso desses valores, além de receber a base de dados em um novo formato. Novos testes foram realizados e concluiu-se que, apesar de apresentar valores satisfatórios para algumas métricas, a rede não era capaz de corrigir textos gerados por sistemas de ASR da maneira correta.

Porém, dentre os objetivos específicos deste trabalho foi possível realizar a implementação de uma Rede Neural Recorrente que, apesar de não ter cumprido com o propósito principal, foi capaz de identificar alguns padrões nos textos e gerar caracteres, mesmo tendo um baixo aproveitamento. Outro objetivo cumprido foi a realização da comparação dos resultados gerados pela rede treinada com a entrada dos dados, utilizando este método para validar a performance do modelo através de diversas métricas que contribuíram para a avaliação final dos experimentos. Além disso, pode-se afirmar que durante a fase de experimentação para o experimento final, a aplicação do cálculo da distância de *Levenshtein* fez com que a rede entendesse melhor os padrões dos textos.

5.1 Trabalhos Futuros

Por fim, com base nos resultados da realização dos experimentos utilizando redes LSTM, percebeu-se que poderia ser interessante aplicar uma nova sequência de experimentos que fizesse uso da base de dados resultante do cálculo da distância de *Levenshtein*, mas que utilizasse uma rede Bi-LSTM. Dessa maneira a rede teria a capacidade de interpretar a frase a direita e a esquerda do “#”, aprimorando seu processamento e entendimento dos textos.

Outra maneira de experimentação seria aplicar um balanceamento nos dados utilizados no experimento final, já que o comando com maior aparição é o de manter, causando um desbalanceamento nos valores da variável alvo. Um aumento no número de dados utilizados durante o treinamento, buscando inserir novos dados com comandos diferentes do manter, poderia ser uma boa solução para o problema.

REFERÊNCIAS

ACADEMY, D. S. **Deep Learning Book**. 2019. Disponível em: https://www.deeplearningbook.com.br/.

AL-NASER, M.; ELSHAFEI, M.; AL-SARKHI, A. Artificial neural network application for multiphase flow patterns detection: A new approach. **Journal of Petroleum Science and Engineering**, Elsevier, v. 145, p. 548–564, 2016.

ALENCAR, V.; ALCAIM, A. Lsf and lpc-derived features for large vocabulary distributed continuous speech recognition in brazilian portuguese. In: IEEE. **2008 42nd Asilomar conference on signals, systems and computers**. [*S.l.*], 2008. p. 1237–1241.

ALI, A.; RENALS, S. Word error rate estimation for speech recognition: e-wer. In: **Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)**. [*S.l.*: *s.n.*], 2018. p. 20–24.

ALLA, S.; ADARI, S. **Beginning Anomaly Detection Using Python-Based Deep Learning: With Keras and PyTorch**. Apress, 2019. ISBN 9781484251775. Disponível em: https://books.google.com.br/books?id=QG21DwAAQBAJ.

BAEVSKI, A. *et al.* wav2vec 2.0: A framework for self-supervised learning of speech representations. **arXiv preprint arXiv:2006.11477**, 2020.

BONACCORSO, G. **Machine learning algorithms**. [*S.l.*]: Packt Publishing Ltd, 2017.

CAO, J.; LI, Z.; LI, J. Financial time series forecasting model based on ceemdan and lstm. **Physica A: Statistical Mechanics and its Applications**, Elsevier, v. 519, p. 127–139, 2019.

CARBUNE, V. *et al.* Fast multi-language lstm-based online handwriting recognition. **International Journal on Document Analysis and Recognition (IJDAR)**, Springer, v. 23, n. 2, p. 89–102, 2020.

CHAUVIN, Y.; RUMELHART, D. E. **Backpropagation: theory, architectures, and applications**. [*S.l.*]: Psychology press, 1995.

CISCO. **VNI Complete Forecast Highlights**. 2018. Disponível em: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_Device_Growth_Traffic_Profiles.pdf. Acesso em: 08 de outubro de 2021.

DONG, C. *et al.* Image super-resolution using deep convolutional networks. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 38, n. 2, p. 295–307, 2015.

ELYASSAMI, S. *et al.* Artificial intelligence-based digital financial fraud detection. **Lecture Notes in Networks and Systems**, v. 308, p. 214–221, 2022. Cited By 0. Disponível em: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85115272119&doi=10.1007%2f978-3-030-85577-2_25&partnerID=40&md5=e47e1d531182457bfb759791a7c50820.

FACELI, K. **Inteligência artificial: uma abordagem de aprendizado de máquina**. Grupo Gen - LTC, 2011. ISBN 9788521618805. Disponível em: https://books.google.com.br/books?id=4DwelAEACAAJ.

GARDNER, M. W.; DORLING, S. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. **Atmospheric environment**, Elsevier, v. 32, n. 14-15, p. 2627–2636, 1998.

GOLPOUR, I. *et al.* Extraction of phenolic compounds with antioxidant activity from strawberries: Modelling with artificial neural networks (anns). **Foods**, v. 10, n. 9, 2021. ISSN 2304-8158. Disponível em: https://www.mdpi.com/2304-8158/10/9/2228.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [*S.l.*]: MIT press, 2016.

GRAVES, A. **Supervised Sequence Labelling with Recurrent Neural Networks**. Springer Berlin Heidelberg, 2012. (Studies in Computational Intelligence). ISBN 9783642247965. Disponível em: https://books.google.com.br/books?id=4UauNDGQWN4C.

GRAVES, A.; JAITLY, N.; MOHAMED, A.-r. Hybrid speech recognition with deep bidirectional lstm. In: IEEE. **2013 IEEE workshop on automatic speech recognition and understanding**. [*S.l.*], 2013. p. 273–278.

GRIS, L. R. *et al.* Desenvolvimento de um modelo de reconhecimento de voz para o português brasileiro com poucos dados utilizando o wav2vec 2.0. In: **Anais do XV Brazilian e-Science Workshop**. Porto Alegre, RS, Brasil: SBC, 2021. p. 129–136. ISSN 2763-8774. Disponível em: https://sol.sbc.org.br/index.php/bresci/article/view/15798.

GUO, H. *et al.* Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach. **Resources Policy**, Elsevier, v. 74, p. 101474, 2021.

HAYKIN, S. **Neural networks and learning machines, 3/E**. [*S.l.*]: Pearson Education India, 2010.

HOCHREITER, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. **International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems**, World Scientific, v. 6, n. 02, p. 107–116, 1998.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HUANG, Z.; XU, W.; YU, K. Bidirectional lstm-crf models for sequence tagging. **arXiv preprint arXiv:1508.01991**, 2015.

JUANG, B.-H.; RABINER, L. R. Automatic speech recognition–a brief history of the technology development. **Georgia Institute of Technology. Atlanta Rutgers University and the University of California. Santa Barbara**, v. 1, p. 67, 2005.

KIM, I.-J.; XIE, X. Handwritten hangul recognition using deep convolutional neural networks. **International Journal on Document Analysis and Recognition (IJDAR)**, Springer, v. 18, n. 1, p. 1–13, 2015.

LEVIS, J.; SUVOROV, R. Automatic speech recognition. **The encyclopedia of applied linguistics**, Wiley Online Library, p. 1–8, 2012.

MALIK, M. *et al.* Automatic speech recognition: a survey. **Multimedia Tools and Applications**, Springer, v. 80, n. 6, p. 9411–9457, 2021.

MITCHELL, T. Machine learning. McGraw hill Burr Ridge, 1997.

NIELSEN, M. A. **Neural networks and deep learning**. [*S.l.*]: Determination press San Francisco, CA, 2015. v. 25.

PANG, Z.; NIU, F.; O'NEILL, Z. Solar radiation prediction using recurrent neural network and artificial neural network: A case study with comparisons. **Renewable Energy**, Elsevier, v. 156, p. 279–289, 2020.

PRATAP, V. *et al.* Mls: A large-scale multilingual dataset for speech research. **arXiv preprint arXiv:2012.03411**, 2020.

REED, R.; MARKSII, R. J. **Neural smithing: supervised learning in feedforward artificial neural networks**. [*S.l.*]: Mit Press, 1999.

ROSENBLATT, F. **Principles of neurodynamics. perceptrons and the theory of brain mechanisms**. [*S.l.*], 1961.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **nature**, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

RUSSELL, S.; NORVIG, P. Artificial intelligence: a modern approach. 2002.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of research and development**, IBM, v. 3, n. 3, p. 210–229, 1959.

SCHMIDT-HIEBER, J. Nonparametric regression using deep neural networks with ReLU activation function. **The Annals of Statistics**, Institute of Mathematical Statistics, v. 48, n. 4, p. 1875 – 1897, 2020. Disponível em: https://doi.org/10.1214/19-AOS1875.

ŞEKER, S.; AYAZ, E.; TÜRKCAN, E. Elman's recurrent neural network applications to condition monitoring in nuclear power plant and rotating machinery. **Engineering applications of artificial intelligence**, Elsevier, v. 16, n. 7-8, p. 647–656, 2003.

SHARMA, S.; SHARMA, S.; ATHAIYA, A. Activation functions in neural networks. **towards data science**, v. 6, n. 12, p. 310–316, 2017.

WANG, Q. *et al.* A comprehensive survey of loss functions in machine learning. **Annals of Data Science**, Springer, p. 1–26, 2020.

WANG, Z.; BOVIK, A. C. Mean squared error: Love it or leave it? a new look at signal fidelity measures. **IEEE signal processing magazine**, IEEE, v. 26, n. 1, p. 98–117, 2009.

WANTO, A. *et al.* Use of binary sigmoid function and linear identity in artificial neural networks for forecasting population density. **IJISTECH (International Journal of Information System & Technology)**, v. 1, n. 1, p. 43–54, 2017.

YU, D.; DENG, L. **Automatic Speech Recognition: A Deep Learning Approach**. Springer London, 2014. (Signals and Communication Technology). ISBN 9781447157793. Disponível em: https://books.google.com.br/books?id=rUBTBQAAQBAJ.

YUJIAN, L.; BO, L. A normalized levenshtein distance metric. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 29, n. 6, p. 1091–1095, 2007.

ZHANG, A. *et al.* Dive into deep learning. **arXiv preprint arXiv:2106.11342**, 2021.