Universidade de Brasília
Departamento de Estatística

Preservação Linguística através de Redes Neurais

Leonardo Gomes Duart

Relatório final apresentado para o Departamento de Estatística da Universidade de Brasília como parte dos requisitos necessários para obtenção do grau de Bacharel em Estatística.

Brasília
2024

Leonardo Gomes Duart

Preservação Linguística através de Redes Neurais

Orientador(a): Dra. Thais Carvalho Valadares Rodrigues
Coorientador(a): Dr. Thiago Costa Chacon

Relatório final apresentado para o Departamento de Estatística da Universidade de Brasília como parte dos requisitos necessários para obtenção do grau de Bacharel em Estatística.

Brasília
2024

Dedico aos meus pais, verdadeiros investidores da minha jornada, que não apenas dedicaram seu tempo, esforço e recursos, mas também acreditaram em mim incondicionalmente.

Agradecimentos

Gostaria de expressar minha profunda gratidão a todos que, de alguma forma, contribuíram para a realização deste trabalho. Primeiramente, agradeço aos meus colegas da universidade, que estiveram ao meu lado em cada etapa desta jornada, compartilhando conhecimento, desafios e apoio.

Aos meus professores, especialmente ao professor Dr. Guilherme Souza Rodrigues, por quem tenho grande admiração, pela paciência e dedicação exemplar como docente; à minha orientadora, Dra. Thais Carvalho Valadares Rodrigues, pelas valiosas contribuições ao longo deste trabalho e pelo apoio desde o início do projeto; e ao professor Dr. Thiago Chacon, meu coorientador, pela orientação especializada e valiosa assistência ao longo do desenvolvimento deste trabalho.

Aos meus pais, Duart e Roseli, que sempre acreditaram no meu potencial e me proporcionaram todo o suporte necessário ao longo dos anos de estudo, dedico este trabalho como um reflexo de todo o investimento e esforço que fizeram por mim. Desde a infância, abriram mão de muito para que eu tivesse acesso às melhores oportunidades educacionais, enfrentando longas distâncias e horários apertados para garantir que eu pudesse me preparar adequadamente. Seu apoio contínuo, tanto em minha formação escolar quanto na universidade, foi essencial para que eu chegasse até aqui.

Por fim, à minha namorada, Luna, pelo apoio constante e companheirismo inabalável, especialmente na reta final deste processo, quando seu suporte fez toda a diferença.

A todos, meu mais sincero e profundo agradecimento.

Resumo

Este trabalho investiga a preservação linguística de línguas indígenas por meio de redes neurais artificiais, focando na tradução automática de áudio para texto. Utilizando modelos de *deep learning*, como Redes Neurais Convolucionais (CNNs) e Redes Neurais Recorrentes (RNNs), o projeto desenvolve e avalia um sistema inspirado na arquitetura *Deep Speech 2*. A pesquisa abrange desde a coleta e pré-processamento dos dados até a avaliação do modelo, com ênfase em métricas como *Word Error Rate* (WER). Os resultados foram favoráveis para línguas usadas na preparação do ambiente, como português e inglês. Já para o Guarani a rede apresentou limitações na generalização e inferência, por conta da natureza dos dados de treino, especialmente pela diversidade de falantes em contraste com a quantidade reduzida de dados. Ainda assim, a transcrição se mostrou útil para reduzir o esforço manual, mesmo com dados de qualidade limitada.

Palavras-chaves: Áudio para Texto; Deep Speech 2; Línguas Indígenas; Preservação Linguística; Processamento de Linguagem Natural; Redes Neurais Convolucionais; Redes Neurais Recorrentes.

Lista de Tabelas

Lista de Figuras

Sumário

1 Introdução

As línguas indígenas encapsulam séculos de conhecimento, tradições e identidades únicas, representando um tesouro cultural e linguístico de valor inestimável. No entanto, muitas dessas línguas estão à beira da extinção, principalmente devido à falta de documentação e de recursos adequados. A preservação dessas línguas é vital não apenas para a manutenção da diversidade cultural, mas também para promover a inclusão e o reconhecimento das comunidades indígenas, que muitas vezes se encontram marginalizadas.

Nesse cenário, as tecnologias de redes neurais surgem como uma ferramenta poderosa para a preservação e revitalização das línguas indígenas. Capazes de capturar padrões complexos de fala, essas tecnologias podem ser aplicadas na criação de sistemas de transcrição automática de áudio para texto, facilitando tanto a documentação quanto a disseminação dessas línguas. Este trabalho propõe o desenvolvimento e avaliação de um sistema de transcrição automática de áudio em uma língua indígena específica, utilizando arquiteturas de redes neurais, como CNNs e RNNs, adaptadas para lidar com as particularidades linguísticas dessa língua.

A arquitetura utilizada neste trabalho é baseada no modelo Deep Speech 2 (AMODEI et al., 2016), uma rede neural profunda desenvolvida pela Baidu Research para transcrição automática de fala. O Deep Speech 2 introduz uma combinação de redes neurais convolucionais (CNNs) e recorrentes bidirecionais (RNNs), com técnicas avançadas de normalização e regularização para melhorar a precisão e a eficiência no reconhecimento de fala em diferentes idiomas. Neste projeto, foram realizadas adaptações na arquitetura original do Deep Speech 2 para adequá-la às particularidades dos dados disponíveis e às necessidades específicas das línguas indígenas em questão. Essas modificações incluem a redução do número de camadas e o ajuste dos hiperparâmetros, visando otimizar o desempenho e viabilidade de treinamento, com uma redução de parâmetros da rede de, aproximadamente, 74%.

Além de contribuir para a preservação do patrimônio cultural, o uso de redes neurais para transcrição automática oferece novas possibilidades para a pesquisa linguística e a educação, criando recursos que podem ser utilizados tanto por estudiosos quanto pelas próprias comunidades indígenas. Ao conectar a tradição com a inovação tecnológica, este projeto visa não apenas assegurar a sobrevivência dessas línguas, mas também promover sua revitalização e inclusão digital, garantindo que continuem a ser uma parte vital do nosso patrimônio global.

Este trabalho está organizado da seguinte forma: No Capítulo 2, é apresentada uma revisão bibliográfica sobre redes neurais artificiais, abordando suas principais arquiteturas e técnicas relevantes para o processamento de linguagem natural. O Capítulo 3 detalha a metodologia adotada, incluindo a escolha dos dados, o pré-processamento, a arquitetura da rede neural implementada e os métodos de avaliação. No Capítulo 4, são apresentados os resultados dos experimentos conduzidos, com análise comparativa entre as diferentes línguas estudadas. Por fim, o Capítulo 5 traz as conclusões do trabalho, discutindo as contribuições para a preservação linguística e possíveis direções para futuras pesquisas.

2 Revisão Bibliográfica

2.1 Redes Neurais Artificiais

As Redes Neurais Artificiais (RNAs) são modelos computacionais inspirados na estrutura e funcionamento do cérebro humano (HAYKIN, 1998). Elas são compostas por unidades básicas chamadas neurônios, organizadas em camadas, que processam informações e aprendem padrões a partir de dados (GOODFELLOW; BENGIO; COURVILLE, 2016). Cada neurônio recebe um conjunto de entradas, realiza uma soma ponderada dessas entradas e aplica uma função de ativação para produzir uma saída.

Uma RNA típica consiste em uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída (BISHOP, 2006). A camada de entrada recebe os dados brutos, enquanto as camadas ocultas e a camada de saída processam e transformam esses dados para realizar tarefas específicas, como classificação, regressão ou reconhecimento de padrões.

O processo de treinamento de uma rede neural envolve ajustar os pesos das conexões entre os neurônios para minimizar um erro ou função de perda. Isso é geralmente feito através de métodos de otimização, como a descida do gradiente e seus variantes, que ajustam iterativamente os pesos com base nos gradientes calculados pelo algoritmo de *backpropagation* (GOODFELLOW; BENGIO; COURVILLE, 2016).

As RNAs têm se mostrado extremamente eficazes em uma ampla gama de aplicações, incluindo reconhecimento de fala, visão computacional, processamento de linguagem natural e muito mais. Sua capacidade de modelar relações complexas e não lineares nos dados as torna uma ferramenta poderosa para resolver problemas que são difíceis de abordar com técnicas tradicionais de aprendizado de máquina (LECUN; BENGIO; HINTON, 2015).

2.2 Overfitting e Underfitting

Overfitting e *underfitting* são dois problemas comuns que surgem no treinamento de modelos de redes neurais e em outros métodos de aprendizado de máquina. Eles estão relacionados ao desempenho do modelo na generalização dos dados, ou seja, à sua capacidade de realizar previsões precisas em novos dados não vistos durante o treinamento.

Overfitting ocorre quando o modelo é treinado para capturar as nuances e detalhes do conjunto de dados de treinamento, incluindo o ruído e as anomalias. Isso resulta

em um modelo que se ajusta muito bem ao conjunto de treinamento, mas que falha ao generalizar para novos dados. Em outras palavras, o modelo tem um desempenho excelente no treinamento, mas baixo em dados de validação ou teste. O *overfitting* pode ser identificado quando há uma grande diferença entre a acurácia do treinamento e a acurácia do teste, com a última sendo significativamente menor.

Underfitting, por outro lado, ocorre quando o modelo é muito simples para capturar os padrões subjacentes no conjunto de dados. Isso pode acontecer se o modelo não tiver capacidade suficiente (por exemplo, não ter camadas ou neurônios suficientes) ou se o treinamento não for suficiente. Um modelo que sofre de *underfitting* apresentará baixo desempenho tanto nos dados de treinamento quanto nos dados de teste, indicando que ele não aprendeu adequadamente a relação entre as variáveis de entrada e saída.

Evitar o *overfitting* e o *underfitting* é um desafio central no desenvolvimento de modelos de aprendizado de máquina. Técnicas como validação cruzada, regularização (como L1 e L2), aumento dos dados de treinamento, e uso de *dropout* são frequentemente empregadas para mitigar esses problemas. A escolha adequada da complexidade do modelo, juntamente com um bom conjunto de dados de treinamento, é crucial para alcançar um equilíbrio entre *underfitting* e *overfitting*.

2.2.1 Funções de Ativação

As funções de ativação são componentes cruciais nas redes neurais artificiais, pois introduzem não-linearidades que permitem que as redes aprendam e representem relações complexas nos dados. Aqui discutimos algumas das funções de ativação mais comuns: Sigmoid, ReLU, Softmax e Tangente Hiperbólica.

Sigmoid

A função Sigmoid, também conhecida como função logística, é amplamente utilizada em redes neurais, especialmente em camadas de saída para problemas de classificação binária (BISHOP, 2006). Ela mapeia qualquer valor real em um intervalo entre 0 e 1, o que pode ser interpretado como uma probabilidade.

A função Sigmoid é definida como:

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

onde e é a base do logaritmo natural. Uma característica importante da função Sigmoid é sua derivada, que é usada no treinamento da rede:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

ReLU (*Rectified Linear Unit*)

A função ReLU é atualmente uma das funções de ativação mais populares devido à sua simplicidade e eficácia (HAHNLOSER et al., 2000). Ela resolve o problema do desvanecimento do gradiente, comumente encontrado em funções como a Sigmoid, permitindo um treinamento mais rápido e eficaz de redes neurais profundas.

A função ReLU é definida como:

$$\text{ReLU}(x) = \max(0, x).$$

Em outras palavras, a saída é zero para entradas negativas e igual à entrada para valores positivos.

Softmax

A função Softmax é frequentemente usada em camadas de saída de redes neurais para problemas de classificação multiclasse (BRIDLE, 1990). Ela converte um vetor de valores reais em um vetor de probabilidades, onde cada valor está entre 0 e 1, e a soma de todas as probabilidades é 1.

A função Softmax é definida como:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}},$$

onde z_i é a i-ésima entrada do vetor de entrada, e K é o número de classes. A saída da função Softmax pode ser interpretada como a probabilidade da entrada pertencer a cada classe.

Tangente Hiperbólica

A função Tangente Hiperbólica, ou tanh, é similar à função Sigmoid, mas mapeia valores reais para um intervalo entre -1 e 1 (BISHOP, 2006). Isso pode ser vantajoso porque os valores centrados em torno de zero podem facilitar o treinamento.

A função tanh é definida como:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

A derivada da função tanh é:

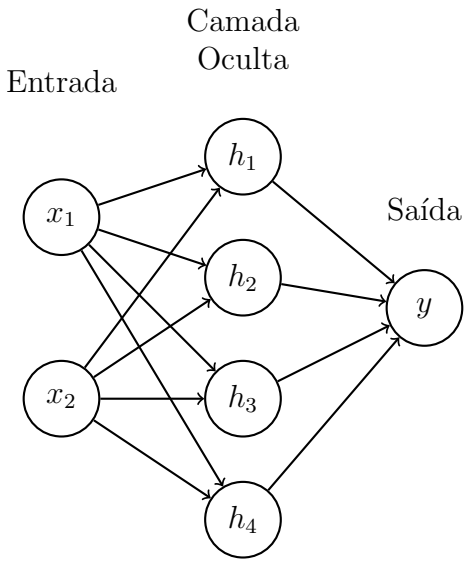
$$\tanh'(x) = 1 - \tanh^2(x).$$

Devido ao seu intervalo de saída, a tanh é frequentemente preferida em camadas ocultas de redes neurais.

2.2.2 Arquitetura básica rede neural

Temos um exemplo de uma rede neural *multi layer perceptron* simples. A rede neural é composta por 2 neurônios de entrada, 4 neurônios na camada oculta e 1 neurônio na camada de saída. Todos os neurônios utilizam a função de ativação sigmoide, exceto o neurônio de saída, que utiliza a função de ativação ReLU.

Figura 1: Arquitetura de rede neural



Fonte: Elaborado pelo autor.

Para cada neurônio na camada oculta e na camada de saída, a entrada h é calculada como:

$$h = \sigma(\sum_{i=1}^{2} w_i \cdot x_i),$$

onde σ representa a função de ativação sigmoide. No entanto, para o neurônio de saída, a função de ativação utilizada é a função ReLU.

Isso significa que a saída da rede é calculada como:

$$y = ReLU(\sum_{h=1}^{4} w_h \cdot h_h).$$

2.2.3 Redes Neurais Convolucionais (CNN)

As Redes Neurais Convolucionais (CNNs) são uma classe especial de redes neurais projetadas para processar dados com uma estrutura de grade, como imagens (GOODFELLOW; BENGIO; COURVILLE, 2016). Uma CNN é composta por várias camadas, incluindo camadas convolucionais, camadas de *pooling* e camadas totalmente conectadas. A seguir, detalhamos a arquitetura e os componentes principais de uma CNN.

Camada Convolucional

A camada convolucional é a espinha dorsal de uma CNN. Ela aplica um conjunto de filtros (kernels) à entrada, produzindo mapas de ativação. Cada filtro é convoluído com a entrada, destacando diferentes características da imagem, como bordas e texturas (LECUN; BENGIO; HINTON, 2015).

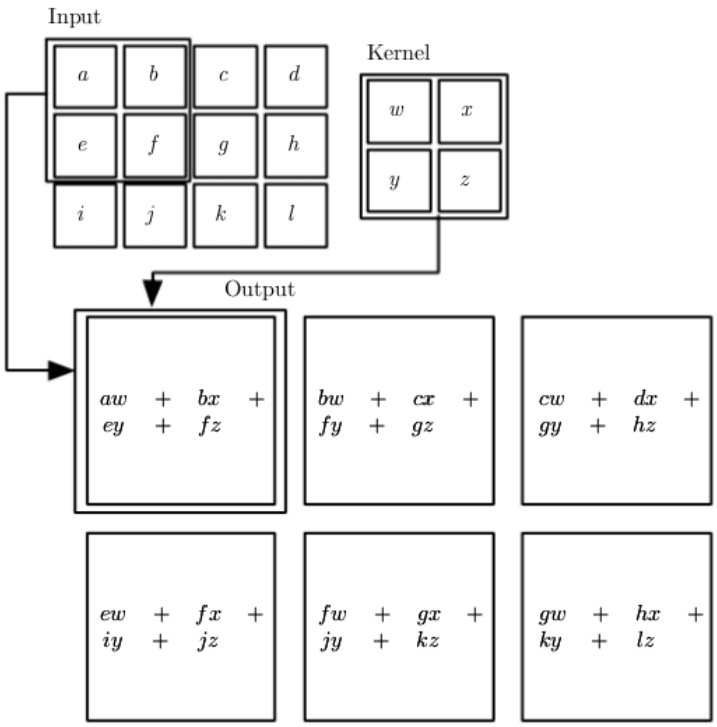
Matematicamente, a operação de convolução pode ser descrita como:

$$f_{i,j,k}^{(l)} = ReLU\left(\sum_{m=1}^{M}\sum_{n=1}^{N} x_{i+m,j+n,c} \cdot w_{m,n,c,k} + b_k\right),$$

onde $f_{i,j,k}^{(l)}$ é o valor do neurônio no mapa de ativação k-ésimo na posição (i, j) da l-ésima camada, $x_{i+m,j+n,c}$ é o valor da entrada na posição $(i + m, j + n)$ do canal c, $w_{m,n,c,k}$ são os pesos do filtro k com dimensões $M \times N$, e b_k é o bias associado ao filtro k. A função de ativação é geralmente a ReLU.

Abaixo temos o exemplo de uma convolução por meio dos Kernels.

Figura 2: Exemplo do método de convolução



Fonte: Goodfellow et al. Deep Learning Book (2016, p. 330).

Camada de Pooling

A camada de pooling é usada para reduzir a dimensionalidade espacial dos mapas de ativação, mantendo as características mais importantes. Uma operação de pooling comum é o max pooling, que toma o valor máximo em uma janela de $p \times p$.

Matematicamente, a operação de max pooling é definida como:

$$f_{i,j,k}^{(l)} = \max_{m,n \in \{0,\ldots,p-1\}} \left(f_{i+m,j+n,k}^{(l-1)} \right).$$

Camada Totalmente Conectada

Após várias camadas convolucionais e de pooling, as CNNs geralmente incluem uma ou mais camadas totalmente conectadas, onde cada neurônio está conectado a todos os neurônios da camada anterior. Isso é semelhante à arquitetura de uma rede neural tradicional.

O valor do neurônio na camada totalmente conectada é dado por:

$$f_j^{(l)} = \sigma \left(\sum_{i=1}^{I} w_{i,j} \cdot f_i^{(l-1)} + b_j \right),$$

onde $f_j^{(l)}$ é o valor do neurônio j-ésimo na l-ésima camada, $w_{i,j}$ são os pesos entre os neurônios da camada anterior e a camada atual, e b_j é o bias.

2.2.4 Redes Neurais Recorrentes (RNN)

As Redes Neurais Recorrentes (RNNs) são uma classe de redes neurais projetadas para processar dados sequenciais, como séries temporais e texto. Uma RNN é caracterizada por ter conexões recorrentes que permitem que a rede mantenha um estado interno, capturando a dependência temporal da sequência de entrada. A seguir, detalhamos a arquitetura e os componentes principais de uma RNN (GOODFELLOW; BENGIO; COURVILLE, 2016).

Camada Recorrente

A camada recorrente é o núcleo de uma RNN. Ela processa a sequência de entrada um elemento por vez, atualizando seu estado oculto em cada passo temporal. O estado oculto em cada passo t é uma função do estado oculto anterior e da entrada atual.

Matematicamente, a atualização do estado oculto h_t pode ser descrita como:

$$h_t = \sigma\left(W_h h_{t-1} + W_x x_t + b\right),$$

onde h_t é o estado oculto no tempo t, W_h é a matriz de pesos recorrentes, h_{t-1} é o estado oculto no tempo anterior $t-1$, W_x é a matriz de pesos de entrada, x_t é a entrada no tempo t, e b é o *bias*. A função de ativação σ é geralmente a tangente hiperbólica (tanh).

Camada de Saída

A camada de saída de uma RNN gera uma predição ou saída em cada passo temporal, baseada no estado oculto atual. Para uma tarefa de classificação, a saída pode ser passada por uma função softmax para obter probabilidades de classe.

Matematicamente, a saída y_t no tempo t é dada por:

$$y_t = \text{softmax}\left(W_y h_t + c\right),$$

onde W_y é a matriz de pesos de saída, h_t é o estado oculto no tempo t, e c é o bias da camada de saída.

2.2.5 Redes Neurais Recorrentes Bidirecionais

Redes Neurais Recorrentes (RNNs) são uma classe de redes neurais onde as conexões entre os neurônios formam um ciclo, permitindo que informações de estados anteriores influenciem estados futuros. Isso as torna particularmente úteis para dados sequenciais, como texto e séries temporais.

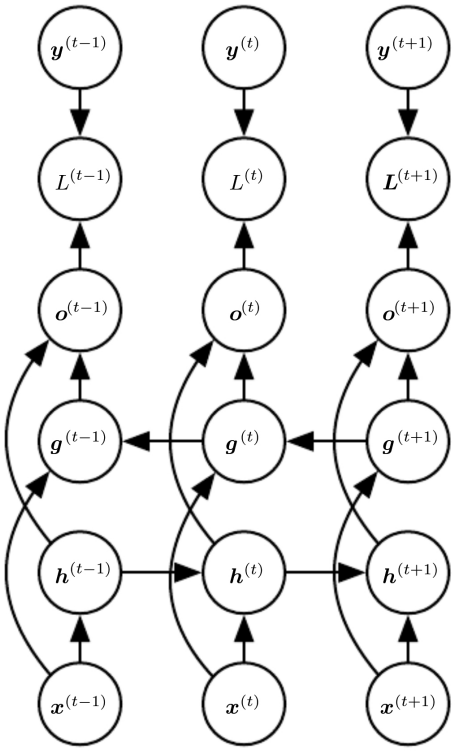
As Redes Neurais Recorrentes Bidirecionais (BRNNs) estendem as RNNs padrão permitindo que cada sequência de treinamento tenha duas direções: uma para frente (do início ao fim) e outra para trás (do fim ao início). Isso permite que a rede tenha acesso tanto ao contexto passado quanto ao futuro de um ponto específico na sequência

(GOODFELLOW; BENGIO; COURVILLE, 2016).

Em muitas aplicações, queremos prever um valor $y(t)$ que pode depender de toda a sequência de entrada. Por exemplo, no reconhecimento de fala, a interpretação correta do som atual como um fonema pode depender dos próximos fonemas devido à coarticulação e pode até depender das próximas palavras devido às dependências linguísticas entre palavras próximas: se houver duas interpretações da palavra atual que são acusticamente plausíveis, podemos ter que olhar longe no futuro (e no passado) para desambiguá-las.

Em uma BRNN (Figura 3), para cada ponto da sequência de entrada, existem duas camadas ocultas: uma processa a sequência na direção positiva (*forward*), e a outra na direção negativa (*backward*). A saída final para cada ponto da sequência é uma combinação das saídas dessas duas camadas ocultas.

Figura 3: Estrutura de uma RNN Bidirecional



Fonte: GOODFELLOW et al. Deep Learning Book (2016, p. 389).

2.3 Métodos de Otimização

A otimização de redes neurais envolve o ajuste dos pesos da rede para minimizar uma função de perda. Aqui discutimos a descida do gradiente, o *backpropagation* e

algumas variantes populares de otimização.

2.3.1 Descida do Gradiente

A descida do gradiente é um método de otimização iterativo usado para minimizar funções de perda. A ideia é atualizar os pesos da rede na direção negativa do gradiente da função de perda em relação aos pesos.

Dado um conjunto de dados de treinamento $\{(x_i, y_i)\}_{i=1}^{N}$, a função de perda L e os pesos W, a atualização dos pesos W em cada iteração t é dada por:

$$W^{(t+1)} = W^{(t)} - \eta \nabla_W L(W^{(t)}),$$

onde η é a taxa de aprendizado e $\nabla_W L(W^{(t)})$ é o gradiente da função de perda em relação aos pesos.

Esse processo é repetido iterativamente até que um critério de parada seja alcançado. Os critérios de parada comuns incluem:

- Um número máximo de iterações.
- Um limite inferior para o valor da função de perda.
- A variação da função de perda entre iterações sucessivas ser menor que um valor pré-definido.

2.3.2 Backpropagation

Backpropagation é o algoritmo utilizado para calcular o gradiente da função de perda em relação aos pesos da rede. Ele usa a regra da cadeia para propagar os erros da saída para as camadas anteriores.

1. Passo *Forward*: Calcule a saída da rede com os pesos atuais.

2. Cálculo do Erro: Calcule o erro da saída em relação ao valor esperado.

3. Passo *Backward*: Propague o erro para trás, calculando os gradientes parciais em cada camada.

Para um neurônio j na camada l, o gradiente da função de perda L em relação ao peso $w_{ij}^{(l)}$ é dado por:

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)},$$

onde $\delta_j^{(l)}$ é o erro da camada l e $a_i^{(l-1)}$ é a ativação do neurônio i na camada anterior $l-1$. Com esses gradientes calculados, os pesos são então atualizados utilizando a regra da descida do gradiente mencionada anteriormente. Esse processo é repetido até que um dos critérios de parada seja atingido, garantindo assim que a rede neural tenha aprendido uma representação adequada dos dados de treinamento.

2.3.3 Momentum

O método de *Momentum* acelera a descida do gradiente acumulando uma fração do gradiente anterior na direção atual. Isso ajuda a suavizar a convergência e escapar de mínimos locais (CHOLLET; ALLAIRE, 2018).

A atualização dos pesos W com Momentum é dada por:

$$v^{(t+1)} = \beta v^{(t)} + (1-\beta)\nabla_W L(W^{(t)}),$$
$$W^{(t+1)} = W^{(t)} - \eta v^{(t+1)},$$

onde v é o vetor de velocidade e β é o fator de Momentum (tipicamente entre 0.9 e 0.99).

2.3.4 Adam (Adaptive Moment Estimation)

Adam é um algoritmo de otimização que combina as ideias de Momentum e AdaGrad (Adaptive Gradient Algorithm). Ele adapta as taxas de aprendizado para cada parâmetro com base nas primeiras e segundas momentas dos gradientes (CHOLLET; ALLAIRE, 2018).

As atualizações dos parâmetros W com Adam são dadas por:

$$m^{(t+1)} = \beta_1 m^{(t)} + (1-\beta_1)\nabla_W L(W^{(t)}),$$
$$v^{(t+1)} = \beta_2 v^{(t)} + (1-\beta_2)(\nabla_W L(W^{(t)}))^2,$$
$$\hat{m}^{(t+1)} = \frac{m^{(t+1)}}{1-\beta_1^{t+1}},$$

$$\hat{v}^{(t+1)} = \frac{v^{(t+1)}}{1-\beta_2^{t+1}},$$

$$W^{(t+1)} = W^{(t)} - \eta \frac{\hat{m}^{(t+1)}}{\sqrt{\hat{v}^{(t+1)}} + \epsilon},$$

onde m e v são as estimativas de primeira e segunda ordem dos momentos, β_1 e β_2 são os fatores de decaimento para os momentos (tipicamente 0.9 e 0.999), e ϵ é um pequeno valor para evitar divisão por zero.

2.4 Métodos de Normalização e Regularização

Para melhorar a performance e a eficiência do treinamento de redes neurais profundas, várias técnicas de normalização e regularização foram desenvolvidas. Aqui discutimos dois métodos populares: *Batch Normalization* e *Dropout*.

Batch Normalization

Batch Normalization é uma técnica que normaliza as ativações de uma camada de rede neural em mini-lotes durante o treinamento. Introduzida por Sergey Ioffe e Christian Szegedy em 2015, essa técnica resolve problemas como o desvanecimento do gradiente, acelera o treinamento e atua como uma forma de regularização, ajudando a prevenir o *overfitting* (CHOLLET; ALLAIRE, 2018).

A ideia principal é normalizar a saída de uma camada de rede para que tenha média zero e variância unitária. A normalização é realizada em mini-lotes durante o treinamento e é seguida por uma transformação linear que permite à rede aprender a melhor escala e deslocamento para as ativações.

A normalização em lote é realizada com as seguintes equações:

1. Calcule a média e a variância do mini-lote:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i.$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2.$$

2. Normalize a saída:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}.$$

3. Escale e desloque a saída normalizada:

$$y_i = \gamma \hat{x}_i + \beta.$$

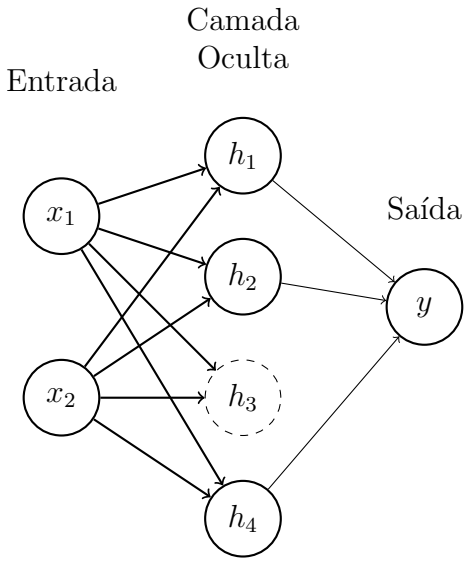
onde ϵ é um pequeno valor para evitar divisão por zero, e γ e β são parâmetros aprendidos pela rede.

Dropout

Dropout é uma técnica de regularização introduzida por Geoffrey Hinton e seus colaboradores em 2012. Durante o treinamento, dropout desativa aleatoriamente uma fração dos neurônios em cada camada, o que força a rede a aprender representações mais robustas e evita o *overfitting* (CHOLLET; ALLAIRE, 2018).

A ideia principal é simples: em cada iteração de treinamento, cada neurônio tem uma probabilidade p de ser "dropado"(Figura 4, neurônio h_3). Durante a inferência, todos os neurônios são utilizados, mas suas saídas são escaladas por p para manter a expectativa das ativações constante.

Figura 4: Exemplo de Dropout em uma camada de rede neural (neurônio desativado h_3)



Fonte: Elaborado pelo autor.

2.5 Espectrograma

O espectrograma (Figura 5) é uma representação visual da intensidade do sinal sonoro ao longo do tempo, dividido em diferentes frequências. Ele é amplamente utilizado em tarefas de processamento de sinais, especialmente no reconhecimento automático de fala, devido à sua capacidade de capturar informações acústicas que permitem identificar padrões sonoros.

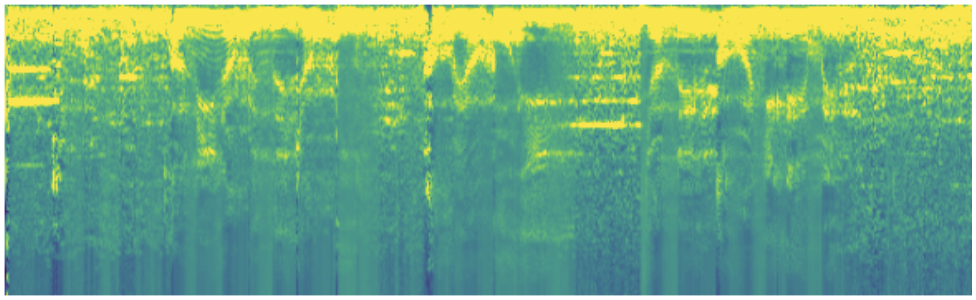
Um espectrograma é gerado ao aplicar a transformada de Fourier de curto tempo (STFT - Short-Time Fourier Transform) sobre um sinal de áudio. A STFT transforma uma sequência temporal em um conjunto de frequências locais, fornecendo uma visão detalhada de como as frequências variam ao longo do tempo. Na representação visual, o eixo horizontal corresponde ao tempo, o eixo vertical representa a frequência, e a intensidade de cada frequência é indicada pela cor ou pela escala de cinza.

No contexto de reconhecimento de fala, o espectrograma fornece uma rica fonte de informações que facilita a identificação de padrões vocais, como fonemas e entonação. As redes neurais convolucionais (CNNs) são frequentemente usadas para processar espectrogramas, pois podem extrair automaticamente as características mais relevantes das representações visuais, auxiliando na transcrição precisa da fala.

A utilidade dos espectrogramas é ampliada ao lidar com variações acústicas em diferentes idiomas e falantes, capturando nuances que não são facilmente detectáveis por métodos tradicionais de processamento de áudio. No entanto, a qualidade dos espectrogramas depende diretamente da qualidade do áudio, resolução de tempo e frequência escolhida na STFT, o que pode afetar a precisão do modelo de reconhecimento de fala.

A seguir tem-se um espectrograma de exemplo da frase “anína pendejapu ha pe’apo’iñane avañe’ẽme”:

Figura 5: Exemplo de espectrograma.



Fonte: Elaborado pelo autor.

3 Metodologia

3.1 Estudos de Viabilidade com Dados Dummy

Estudos preliminares foram conduzidos utilizando uma arquitetura de rede neural, baseada na *Deep Speech* 2 (AMODEI et al., 2016), para reconhecimento de fala, com dados estruturados para treino e teste em português e inglês. Esses estudos tiveram como objetivo avaliar a capacidade de aprendizado, generalização da arquitetura e viabilidade do projeto. Fornecendo uma base comparativa para os experimentos subsequentes com as línguas indígenas Arutani e Guarani.

3.2 Escolha do Conjuto de Dados

Inicialmente, foram considerados dados em Arutani, obtidos a partir de gravações de conversas em colaboração com o professor Thiago Chacon, coorientador deste trabalho. O acervo, composto por horas de gravações contínuas, apresenta um valor linguístico significativo, contendo transcrições em Arutani e traduções para português e inglês.

No entanto, devido à natureza dos dados, que foram produzidos em ambientes não controlados e são afetados por perturbações externas (como chuva, vento e vozes sussurradas), constatou-se que a qualidade desses dados pode não ser suficiente para garantir a convergência eficiente do modelo. Além disso, o volume limitado de dados disponíveis pode comprometer a eficácia da arquitetura, mesmo com a aplicação de técnicas de transferência de aprendizado. Esses fatores levaram à decisão de explorar uma alternativa mais robusta para a continuação do estudo.

3.2.1 Fonte de Dados Guarani

Como alternativa ao Arutani, optou-se pela utilização de dados em Guarani, disponibilizados pelo projeto *Mozilla Common Voice* (FOUNDATION, 2024). Esses dados incluem gravações de áudio em Guarani, acompanhadas de transcrições correspondentes, que foram coletadas em condições mais controladas, oferecendo uma base mais consistente para o treinamento da rede neural.

A mudança para os dados Guarani permitirá uma análise mais robusta e comparável, uma vez que o volume de dados e a qualidade das gravações são mais adequados para o desenvolvimento e a avaliação de modelos de reconhecimento de fala. Essa

adaptação na metodologia se justifica pela necessidade de garantir a viabilidade e a qualidade dos resultados obtidos, sem comprometer a integridade do estudo.

3.2.2 Estrutura Linguística e Alfabeto do Guarani

O Guarani é uma língua da família Tupi-Guarani, falada amplamente na América do Sul, especialmente no Paraguai, onde é uma das línguas oficiais. A língua possui uma estrutura aglutinativa, permitindo a formação de palavras complexas através da combinação de vários morfemas. Isso possibilita que uma única palavra em Guarani transmita significados que, em outras línguas, necessitariam de várias palavras ou até de uma frase completa.

O alfabeto Guarani é baseado no alfabeto latino e consiste em 33 letras, incluindo vogais que podem ser nasalizadas (ã, ẽ, ĩ, õ, ũ, ỹ), uma característica fonológica essencial que diferencia palavras de significados distintos. Além da nasalização, o Guarani utiliza acentos agudos (á, é, í, ó, ú, ý) para indicar a tonicidade das palavras. A letra "ñ"é usada para representar um som nasal palatal, e o "g̃"representa um som nasalizado único na língua.

No Guarani, o apóstrofo utilizado nas palavras, como em re ro'use e ka'aru, é uma marca ortográfica específica que representa a elisão de sons ou uma interrupção na pronúncia (glotalização). O correto é utilizar o apóstrofo reto, que é este símbolo: '. Ele é utilizado para indicar a ausência de um som que deveria estar presente ou para marcar uma pausa sonora.

3.2.3 Tratamento dos Dados Guarani

Os dados de áudio em Guarani utilizados neste estudo foram obtidos do projeto *Mozilla Common Voice* (FOUNDATION, 2024). Após o download dos arquivos, foi necessário realizar um pré-tratamento para adequar a taxa de amostragem dos áudios ao padrão exigido pela arquitetura de rede neural utilizada.

O pré-tratamento foi realizado no RStudio, utilizando ferramentas de processamento de áudio disponíveis no pacote *TuneR*. Os áudios foram convertidos para uma taxa de amostragem de 16.000 Hz, que é comumente utilizada em modelos de reconhecimento de fala devido ao seu equilíbrio entre qualidade de áudio e eficiência computacional. Essa taxa de amostragem permite que os modelos captem adequadamente as características importantes do sinal de fala, sem incluir informações redundantes que possam sobrecarregar

o treinamento da rede.

3.3 Estrutura dos dados

Portanto há três fontes de dados: Português, Inglês e Guarani. Os dados em inglês são conhecidos como LJ Speech Dataset (ITO, 2017). Os dados de português foram adquiridos por meio do GitLab do Grupo FalaBrasil (FB Audio Corpora, 2019).

Os dados são bem diferentes estruturalmente. Enquanto os dados em inglês são mais curtos e em maior quantidade, os em português são mais longos porém com menos de 10% da quantidade dos dados em inglês. Essa escolha foi intencional para testar como a rede se adaptaria a estruturas de dados distintas, antes de iniciar os estudos para o guarani.

- **Português:** Áudios de um homem lendo a Constituição, gravados em um ambiente sonoro bem controlado.
  - **Tamanho médio dos áudios:** 30 segundos;
  - **Tamanho total do dataset:** 1.250 áudios;
  - **Tempo de treinamento:** 2 horas para 50 épocas.
- **Inglês:** Áudios de uma mulher lendo livros, também em um ambiente sonoro bem controlado.
  - **Tamanho médio dos áudios:** $6,57$ segundos;
  - **Tamanho total do dataset:** 13.100 áudios;
  - **Tempo de treinamento:** 4 horas para 50 épocas.
- **Guarani:** Áudios de diversos falantes com ambientes, em geral, controlados.
  - **Tamanho médio dos áudios:** $5,71$ segundos;
  - **Tamanho total do dataset:** 2.520 áudios;
  - **Tempo de treinamento:** 1.5 hora para 50 épocas.

Os dados do guarani, diferente dos dados em inglês e português, possuem amostras de diversos falantes, porém, dentre os 2520 áudios em Guarani, apenas 1641 possuem rótulos de faixa etária e sexo. Desses, a distribuição é a seguinte:

Tabela 1: Distribuição de amostras por Faixa Etária (Sexo Masculino).

Faixa Etária	**Quantidade**
13-19	3
20-29	457
30-39	833
40-49	-
50-59	-
60-69	23

Tabela 2: Distribuição de amostras por Faixa Etária (Sexo Feminino).

Faixa Etária	**Quantidade**
13-19	-
20-29	260
30-39	44
40-49	21
50-59	-
60-69	-

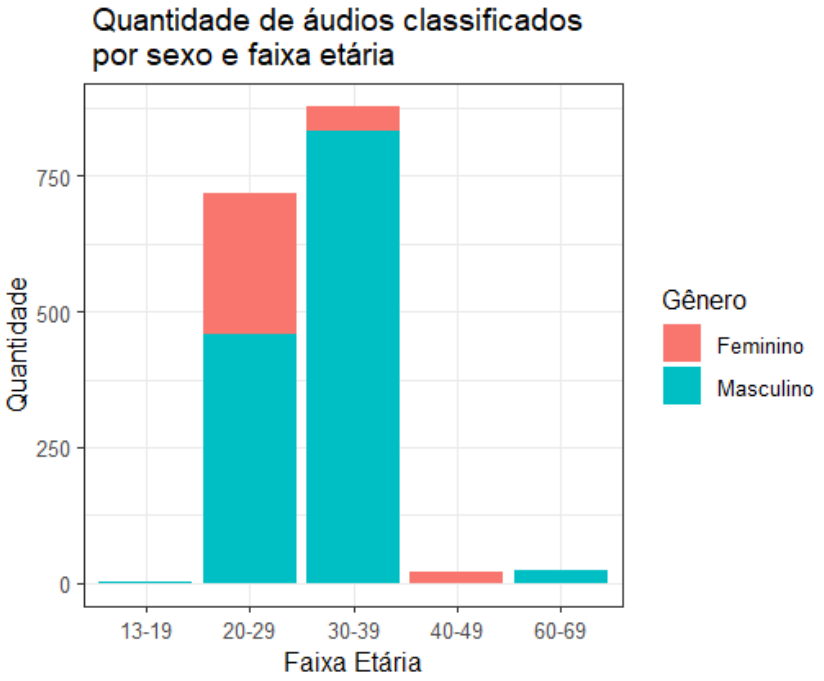
Conforme as tabelas apresentadas, observa-se que a distribuição de amostras por faixa etária é bastante desigual entre os sexos. No caso do sexo masculino (Tabela 1), a maioria das amostras está concentrada nas faixas etárias de 20-29 anos e 30-39 anos, com 457 e 833 amostras, respectivamente. As outras faixas etárias possuem uma representação muito baixa ou inexistente, com apenas 3 amostras na faixa de 13-19 anos e 23 amostras na faixa de 60-69 anos. Não há registros de amostras nas faixas de 40-49 e 50-59 anos.

Para o sexo feminino (Tabela 2), a distribuição também é desigual, mas de forma diferente. A maioria das amostras está concentrada nas faixas etárias de 20-29 anos e de 30-39 anos, com 260 e 44 amostras, respectivamente. As outras faixas etárias possuem pouca ou nenhuma representação, sem amostras nas faixas etárias equivalentes a 13-19 anos, 50-59 anos e 60-69 anos. A faixa etária de 40-49 anos conta apenas com 21 amostras.

Esses dados mostram que a amostragem não é uniforme e que certas faixas etárias e sexos estão sub-representados, o que pode impactar a eficácia de inferência do modelo de reconhecimento de fala.

Há mais uma forma de visualização dos dados.

Figura 6: Gráfico de colunas que retrata a distribuição de agrupamento de áudios rotulados por sexo e faixa etária.



Fonte: Elaborado pelo autor.

3.4 Ferramentas e Frameworks

Os principais instrumentos de pesquisa incluirão *software* de processamento de áudio para pré-processamento dos dados e ambiente de desenvolvimento com TensorFlow e Keras para implementação da rede neural, manipulados pela linguagem Python.

3.5 Implementação da Arquitetura Neural

A implementação da arquitetura neural será realizada utilizando o framework Keras devido à sua flexibilidade e documentação abrangente. Diferentes tipos de camadas serão exploradas, incluindo redes neurais convolucionais (CNNs) e recorrentes (RNNs), adaptados para processamento de sequências de áudio.

3.5.1 Arquitetura de interesse

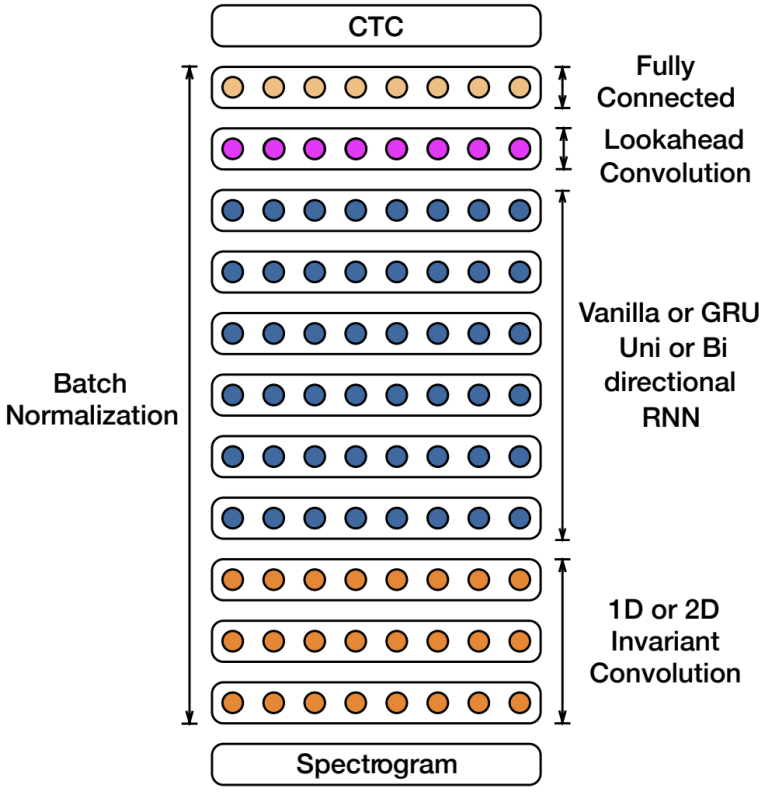
A arquitetura do modelo será inspirada em uma arquitetura já existente, chamada de *Deep Speech* 2 (AMODEI et al., 2016).

O *Deep Speech* 2 é um modelo de reconhecimento de fala que utiliza uma arquitetura de aprendizado profundo para converter áudio em texto. Ele foi desenvolvido pela Baidu Research e é uma melhoria em relação ao modelo original *Deep Speech* (AMODEI et al., 2016).

O *Deep Speech* 2 utiliza uma rede neural recorrente (RNN) em conjunto com a rede neural convolucional (CNN) com métodos de normalização e prevenção de *overfitting* para realizar o reconhecimento de fala de maneira *end-to-end*, ou seja, diretamente do áudio para o texto, sem a necessidade de etapas intermediárias de processamento. A figura 7 ilustra a Arquitetura do *Deep Speech* 2.

Para o treinamento desse modelo foram utilizadas em média 10000 horas de áudios de cada língua (Mandarim e Inglês), um número muito superior ao que tem-se para o estudo atual.

Figura 7: Arquitetura do *Deep Speech* 2

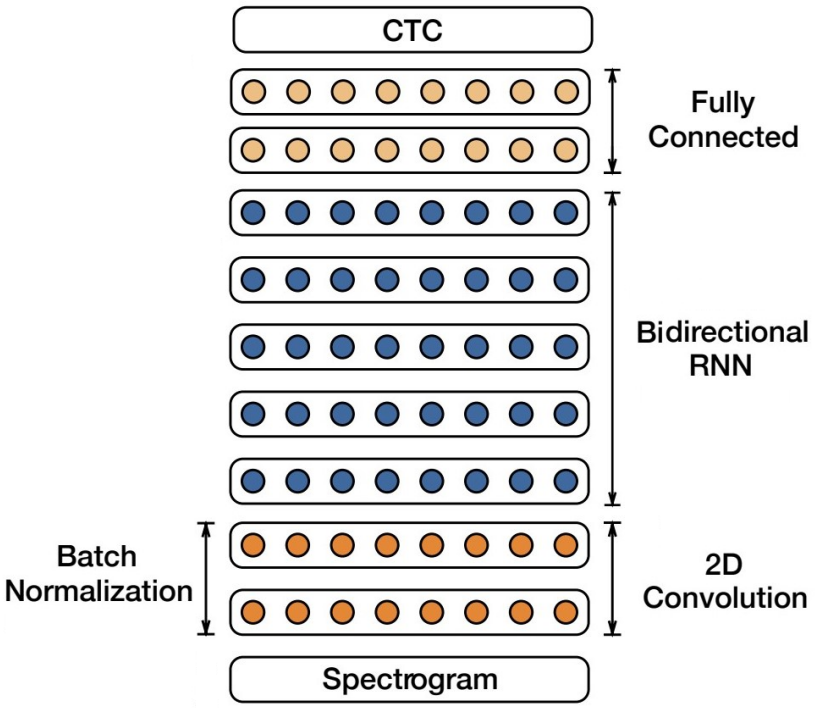


Fonte: AMODEI, D. et al. *Deep Speech* 2: End-to-end speech recognition in English and Mandarin (2016, p. 02).

Baseado nessa arquitetura será proposta uma similar: 2 camadas convolucionais 2D, com Batch Normalization e funções de ativação ReLU após cada camada, seguido de

5 camadas recorrentes bidirecionais com dropout entre elas, seguido de 1 camada densa com ativação ReLU e droupout e, por fim, outra camada densa de output com a função de ativação softmax, afim de gerar um vetor de probabilidades (BOUADJENEK; HUYNH, 2021). Essa estrutura conta com 26.628.480 parâmetros (Figura 8), enquanto o *Deep Speech* 2 possui cerca de 68 milhões de parâmetros.

Figura 8: Arquitetura alvo



Fonte: Elaborado pelo autor.

A arquitetura inspirada no *Deep Speech* 2 foi escolhida pela sua eficiência comprovada em tarefas de reconhecimento de fala, mas foi adaptada para reduzir a complexidade em comparação ao modelo original. Mas o *Deep Speech* 2, sendo uma rede bastante complexa, possui um número elevado de parâmetros, o que pode ser desvantajoso para bases de dados pequenas, como a utilizada neste projeto.

Com a quantidade limitada de dados, uma arquitetura tão pesada poderia não aprender de maneira eficiente, resultando em *overfitting* ou em tempos de treinamento muito longos. Por isso, optou-se por reduzir o número de parâmetros, mantendo os principais componentes da arquitetura, como camadas convolucionais e recorrentes, mas diminuindo a profundidade e o tamanho das camadas. Isso garantiu um treinamento mais rápido e adequado ao volume de dados disponível, sem sacrificar tanto a capacidade de

generalização do modelo.

3.5.2 Função de perda

Durante o treinamento do modelo, a função de perda utilizada será o CTC (Connectionist Temporal Classification) Loss.

O CTC Loss é uma função de custo usada principalmente em tarefas de reconhecimento de fala para treinar modelos de aprendizado profundo, como redes neurais recorrentes (RNNs) ou redes neurais convolucionais (CNNs), para converter sequências de entrada em sequências de saída. Em particular, é útil quando não há uma correspondência direta entre os elementos da sequência de entrada e os da sequência de saída, o que é comum em tarefas como o reconhecimento de fala.

A ideia principal por trás do CTC Loss é calcular a probabilidade de todas as sequências de saída possíveis que poderiam corresponder à sequência de entrada observada. Isso é feito somando as probabilidades de todas as maneiras diferentes de "alinhamento"entre a sequência de entrada e a sequência de saída.

A função de perda CTC é dada por:

$$L(y, \hat{y}) = -\log(\sum_{\pi \in \mathcal{B}^{-1}(\hat{y})} p(\pi|x)),$$

onde:

- y é a sequência de saída verdadeira.
- $\hat{y}$ é a sequência de saída predita pelo modelo.
- π é uma sequência de saída possível.
- $\mathcal{B}^{-1}(\hat{y})$ é o conjunto de todas as sequências possíveis que se "transformam"em $\hat{y}$ após a remoção de símbolos repetidos e símbolos de branco.
- $p(\pi|x)$ é a probabilidade da sequência π dado o sinal de entrada x.

Em essência, o CTC Loss penaliza as saídas do modelo que têm baixa probabilidade sob a distribuição de alinhamento entre a sequência de entrada e a sequência de saída verdadeira.

3.6 Treinamento do Modelo

O treinamento e os estudos prévios serão executado em ambiente local, com uma placa de vídeo RTX 4060ti, processador ryzen 5 5600 e 32GB de memória RAM e no ambiente de nuvem em GPUs alugadas na plataforma Google Colab, como a GPU NVIDIA L4 Tensor Core, permitindo uma aceleração significativa do processo de aprendizado.

Como a quantidade de dados é pequena, será utilizada uma separação de 90% dos dados para treino e 10% para teste.

3.7 Avaliação do Modelo

Após o treinamento, o modelo será avaliado utilizando métricas específicas para tradução de áudio para texto, como BLEU score e WER score.

Testes de inferência serão conduzidos com áudios de teste não vistos para verificar a capacidade do modelo de generalização e sua eficácia na tradução de diferentes tipos de áudio.

3.7.1 BLEU Score

O BLEU (*Bilingual Evaluation Understudy*) score é uma métrica comumente utilizada para avaliar a qualidade de traduções automáticas. Ele compara a tradução candidata com uma ou mais traduções de referência, calculando a precisão das palavras coincidentes e considerando a fluência e a ordem das palavras. A pontuação BLEU é normalmente apresentada como um valor entre 0 e 1, onde valores mais altos indicam uma maior semelhança com as traduções de referência.

A fórmula básica para calcular o BLEU Score é:

$$\text{BLEU} = \text{BP} \times \exp(\sum_{n=1}^{N} w_n \log p_n), \tag{3.7.1}$$

onde BP é o fator de penalidade de bilinguismo, w_n são os pesos atribuídos às n-gramas, p_n é a precisão das n-gramas e N é o número máximo de n-gramas.

O BLEU Score leva em consideração a precisão das n-gramas (geralmente até 4-gramas), penalizando traduções mais curtas e favorecendo aquelas que são mais semelhantes às traduções de referência.

É uma métrica útil para avaliar a qualidade de sistemas de tradução automática, oferecendo uma medida quantitativa da precisão e da fluidez das traduções geradas automaticamente.

3.7.2 Pontuação WER (Word Error Rate)

A pontuação Word Error Rate (WER) é uma métrica amplamente utilizada para avaliar a precisão de sistemas de reconhecimento automático de fala (ASR). O WER é calculado com base no número de substituições (S), inserções (I) e deleções (D) necessárias para transformar a transcrição gerada automaticamente (hipótese) na transcrição de referência (verdade). É a métrica que será mais utilizada nesse estudo. A fórmula matemática para calcular o WER é dada por:

$$\text{WER} = \frac{S + D + I}{N}, \tag{3.7.2}$$

onde N é o número total de palavras na transcrição de referência. Vamos detalhar cada componente da fórmula:

- **Substituições (S)**: Ocorre quando uma palavra na transcrição de referência é substituída por uma palavra diferente na hipótese.
- **Deleções (D)**: Ocorre quando uma palavra presente na transcrição de referência está ausente na hipótese.
- **Inserções (I)**: Ocorre quando uma palavra não presente na transcrição de referência é adicionada na hipótese.

Considere os seguintes exemplos de transcrição de referência e hipótese:

- **Transcrição de Referência**: “o gato está no telhado”
- **Hipótese**: “o rato está no jardim”

Para calcular o WER, precisamos primeiro identificar o número de substituições, deleções e inserções:

- Substituições (S): “gato”é substituído por “rato”, “telhado”é substituído por “jardim”. Então, $S = 2$.

- Deleções (D): Não há deleções, então $D = 0$.
- Inserções (I): Não há inserções, então $I = 0$.

O número total de palavras na transcrição de referência é $N = 5$. Portanto, o WER é calculado como:

$$\text{WER} = \frac{2 + 0 + 0}{5} = 0.4\,(40\%) \tag{3.7.3}$$

Este exemplo ilustra como o WER pode ser usado para quantificar a precisão de uma transcrição de fala gerada automaticamente em comparação com a transcrição de referência. Quanto menor o WER, mais precisa é a transcrição gerada.

4 Resultados

Neste capítulo, apresentamos não só os resultados preliminares dos experimentos conduzidos utilizando a arquitetura de rede neural baseada no Deep Speech 2 para a transcrição da fala do inglês e português, como também os resultados aplicados para o contexto do Guarani.

4.1 Configuração dos Experimentos

Os experimentos com inglês e português utilizaram dados mais estruturados e de melhor qualidade para testar a eficácia da arquitetura baseada no *Deep Speech* 2. Os treinamentos foram feitos separadamente, sem atualização de parâmetros ou *transfer learning*.

Os textos em português e em inglês não conterão acentos ortográficos ou pontuação nas frases, apenas palavras por extenso sem símbolos, tanto no treinamento quanto na previsão. Portanto, o dicionário de treino e preditivo utilizará *encoding* do alfabeto padrão, espaço ou vazio.

4.2 Configuração Específica para o Guarani

No caso do Guarani, devido às suas particularidades linguísticas e ao uso de caracteres especiais, o dicionário de treinamento foi adaptado para refletir com precisão os aspectos fonológicos da língua. O alfabeto utilizado incluiu não apenas as letras padrão, mas também caracteres específicos como o apóstrofo reto, o apóstrofo curvo além das vogais com acentuação e nasalização, como “á”, “ã”, “ĩ”, “õ”, entre outras.

Esse dicionário ampliado para o Guarani foi construído para garantir que o modelo de rede neural pudesse lidar com as especificidades da língua, preservando a integridade das palavras durante o processo de transcrição. Ao contrário dos experimentos com português e inglês, onde símbolos e acentos foram removidos, no Guarani foi crucial manter esses elementos para capturar corretamente as nuances fonéticas e morfológicas que são essenciais para a compreensão precisa da língua.

Durante o treinamento, o modelo foi exposto a dados de fala em Guarani que refletiam o uso completo desse alfabeto, incluindo as combinações de letras e símbolos que são características da língua. A previsão dos textos seguiu a mesma configuração,

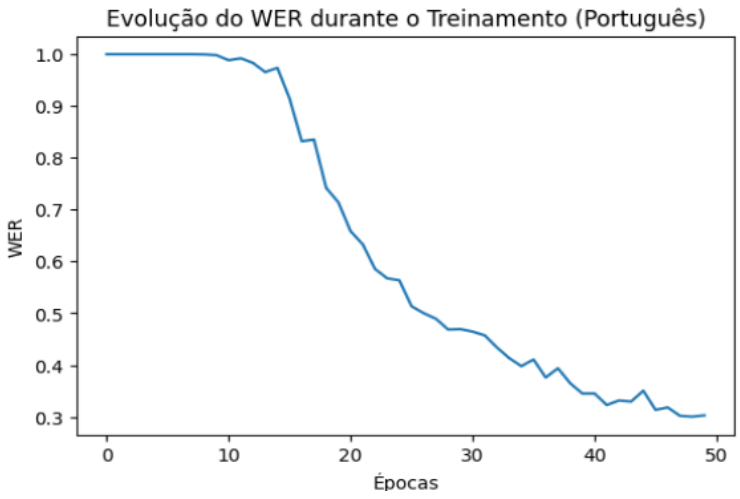
utilizando o dicionário com os caracteres descritos.

4.3 Resultados dos Experimentos

Os resultados preliminares mostram que a rede se adaptou muito bem a duas línguas distintas com estruturas de dados bem diferentes.

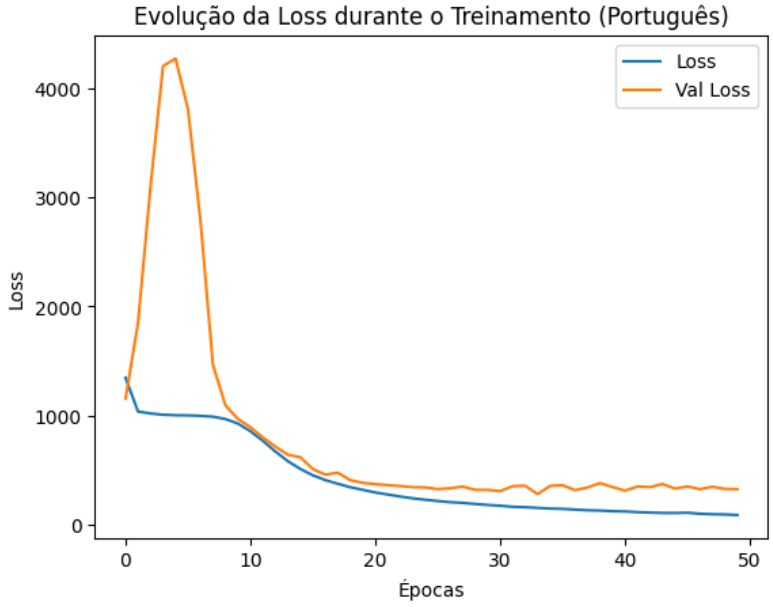
Conforme ilustrado na Figura 9, os dados em português foi alcançado um WER de cerca de $0,30$ após 50 épocas. Nota-se que houve uma certa demora para ocorrer a convergência da rede. Verifica-se, pela Figura 10, que o *Loss* na validação teve uma explosão antes de começar a convergir, mas finalizou em cerca de $90,50$ para o treino e para a validação $327,24$.

Figura 9: WER na base de teste para o dataset em português ao longo das épocas



Fonte: Elaborada pelo autor.

Figura 10: Desempenho da rede para o dataset em português ao longo das épocas



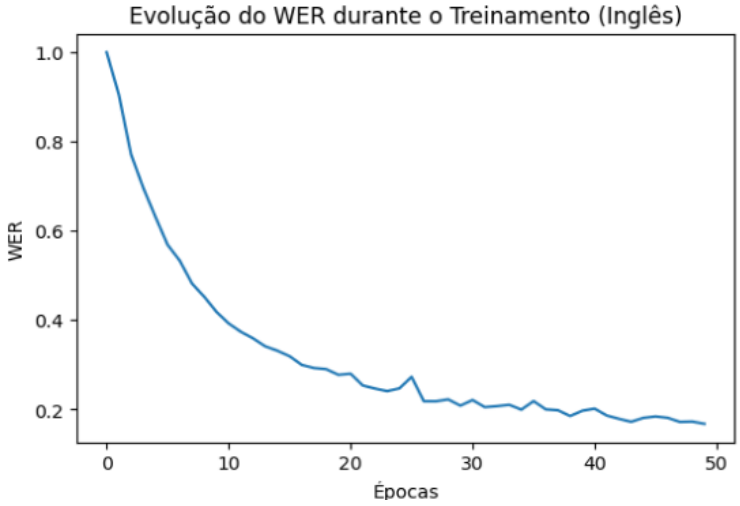
Fonte: Elaborada pelo autor.

A seguir tem-se uma transcrição correta e de uma transcrição predita:

- **Frase original:** "o tempo de servico dos servidores referidos neste artigo sera contado como titulo quando se submeterem a concurso para fins de efetivacao na forma da lei paragrafo segundo o disposto neste artigo nao se aplica aos ocupantes de cargos funcoes e empregos de confianca ou em comissao".
- **Frase predita:** "o tempo de servico dos servidores referidos neste artigo sera contado comustitolo quando ses submeterem a com curso para fins de efetivacao na forma da lei paragrafo segunto outisposto neste artigo nao se aplica aos cupantes de cargos funcoes e mpregos de confianca ou em conicao".

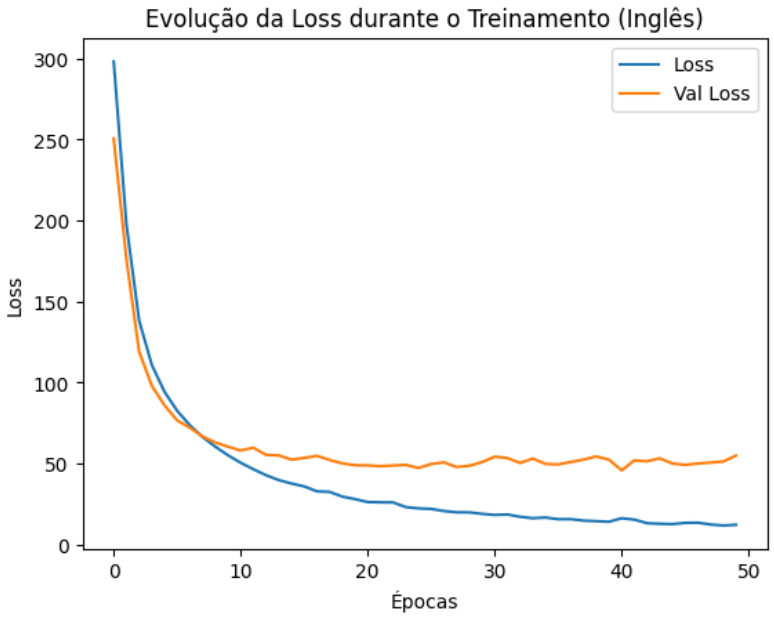
Já para o inglês, na Figura 11 nota-se que após as 50 épocas foi alcançado um WER de cerca de $0,17$. Não só a convergência foi alcançada em menos épocas, como também e as métricas atingiram valores preferíveis, o que é esperado por ser um banco de dados consideravelmente maior. A *Loss*, ao final do treino, atingiu cerca de $12,10$ para o treino e para a validação $54,81$, conforme ilustrado na Figura 12.

Figura 11: WER na base de teste para o dataset em inglês ao longo das épocas



Fonte: Elaborada pelo autor.

Figura 12: Desempenho da rede para o dataset em inglês ao longo das épocas



Fonte: Elaborada pelo autor.

A seguir tem-se uma transcrição-alvo e de uma transcrição predita:

- **Frase original:** “their treatment was also a matter of chance they still slept on rope mats on the floor herded together in companies of four or more to keep one another warm”.

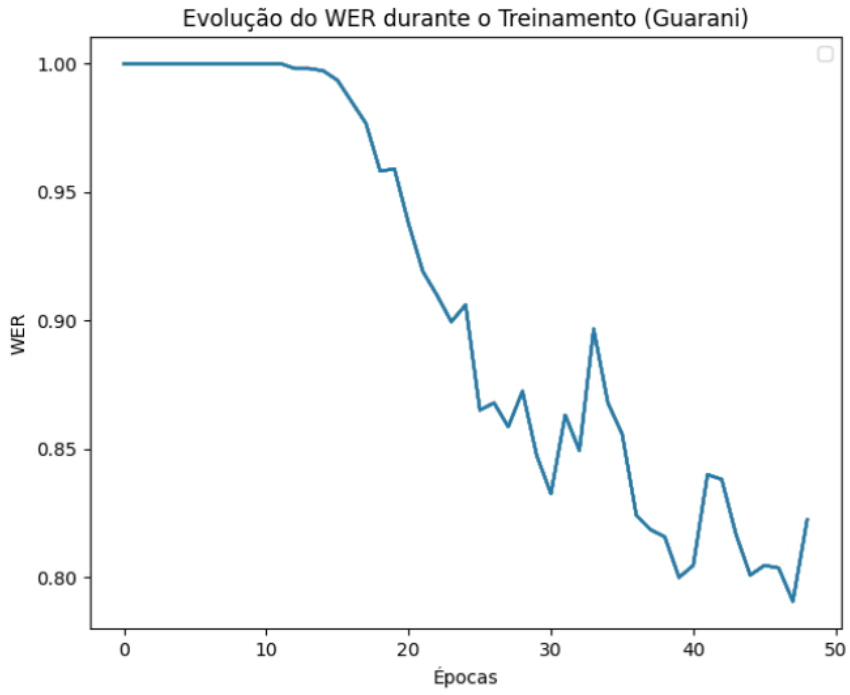
- **Frase predita:** “their treatment was also a matter of chance they still slept on rope mats on the floor hurded together in companies of for or more to keep one another worm”.

4.4 Resultados para o Guarani

Para o Guarani, os resultados apresentaram uma diferença significativa em termos de WER se comparados ao inglês e ao português, refletindo os desafios inerentes ao conjunto de dados com mais falantes.

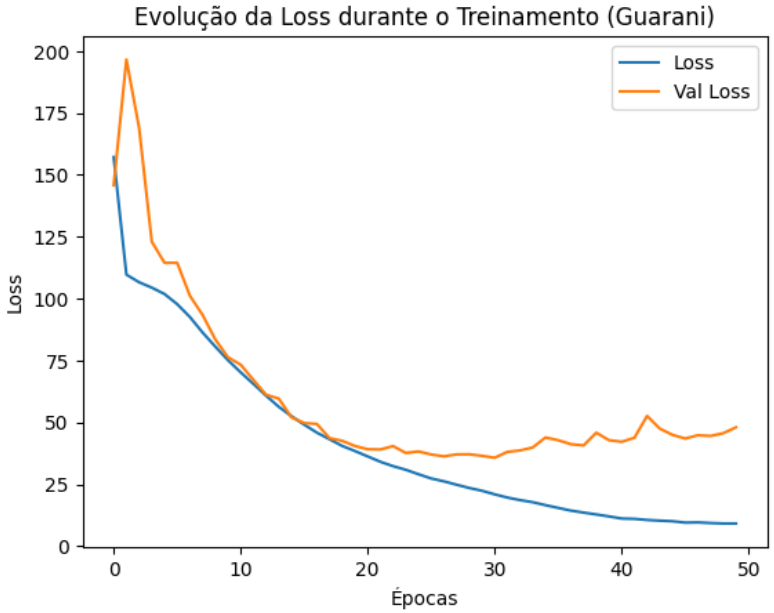
Conforme ilustrado na Figura 12, o treinamento no *dataset* em Guarani evidenciou sinais claros de *overfitting*. A *Loss* na validação não acompanhou adequadamente a *Loss* do treinamento após certo ponto, estabilizando-se em valores consideravelmente superiores. Ao final do treinamento, a *Loss* final para o treino foi de aproximadamente $9,38$, enquanto para a validação foi de $48,03$.

Figura 13: WER na base de teste para o dataset em guarani ao longo das épocas



Fonte: Elaborada pelo autor.

Figura 14: Desempenho da rede para o dataset em guarani ao longo das épocas



Fonte: Elaborada pelo autor.

Para efeito de comparação foram utilizadas previsões antes do *overfitting*, na época 23, e ao final do treinamento, na época 50.

Para uma frase dita por um falante do sexo masculino:

- **Frase original:** "amombe'uporãta ndéve mba'éichapa reguahẽkuaa ore rogamíme".
- **Frase predita 23 épocas:** "amombomorãtendee mba'eichaperenahekuaorerogamime".
- **Frase predita 50 épocas:** "amombuporãte ndee mba'éichapere nohekua ore rógamíme".

Para uma frase dita por um falante do sexo feminino:

- **Frase original:** "opa rire ñembo'eguasu añemboja ha añemongeta karai pa'i ndive".
- **Frase predita 23 épocas:** "opáire ñembo'eguachupañembojaha ñemone akaripainere".
- **Frase predita 50 épocas:** "opái re ñembo'eguasupa ñemboja ha ñemoneta karaipa'inive".

Apesar do *overfitting* sugerido pelo gráfico, os resultados melhoraram, e houve uma reflexão para o WER, onde foi de cerca de $0,95$ na época 23, para $0,80$ ao final de

50 épocas. Para o caso de menos épocas, a rede erra mais acentuação, apóstrofo e espaço entre as palavras. Então, apesar do overfitting, a rede obteve uma melhora na inferência em épocas mais avançadas.

O WER apresentou valores altos, possivelmente influenciado pela heterogeneidade dos dados. Diferentemente dos *datasets* em português e inglês, que consistem em falas de um único locutor, o conjunto de dados em Guarani foi coletado de múltiplos falantes, introduzindo variações de pronúncia, entonação e ritmo, que podem ter dificultado o processo de aprendizado do modelo.

Nesse contexto de dados de áudio de naturezas distintas é recomendado um nível de dados superior, que seria uma possível solução para auxiliar na convergência da rede.

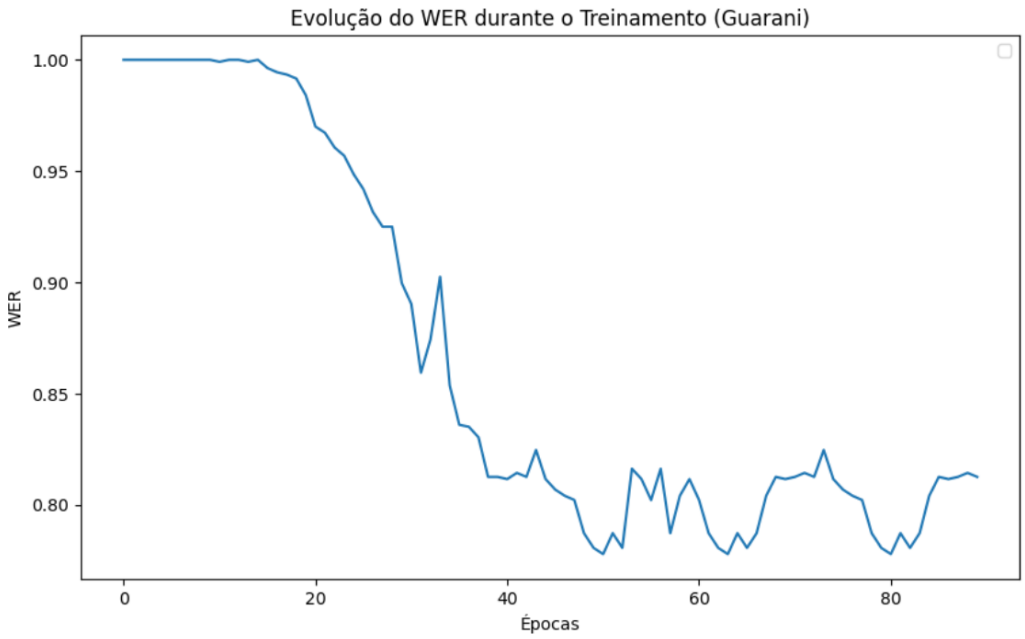
4.4.1 Mudanças na Arquitetura

Foram realizadas tentativas de mudanças na arquitetura, como a diminuição de uma camada recorrente, aumento do *learning rate* de 10^{-4} para 10^{-3} e aumento de *dropout* de $0,5$ para $0,6$. Elas foram realizadas individualmente e em grupo, quando possível.

A mudança de *dropout* com a finalidade de controlar o *overfitting* não se mostrou eficaz, afinal, *dropout* em excesso pode gerar dificuldades na convergência da rede. Após o aumento na taxa, a rede com a arquitetura original não convergiu durante 50 épocas, nem com a diminuição de complexidade por meio da remoção da camada recorrente, ou com o aumento do *learning rate*.

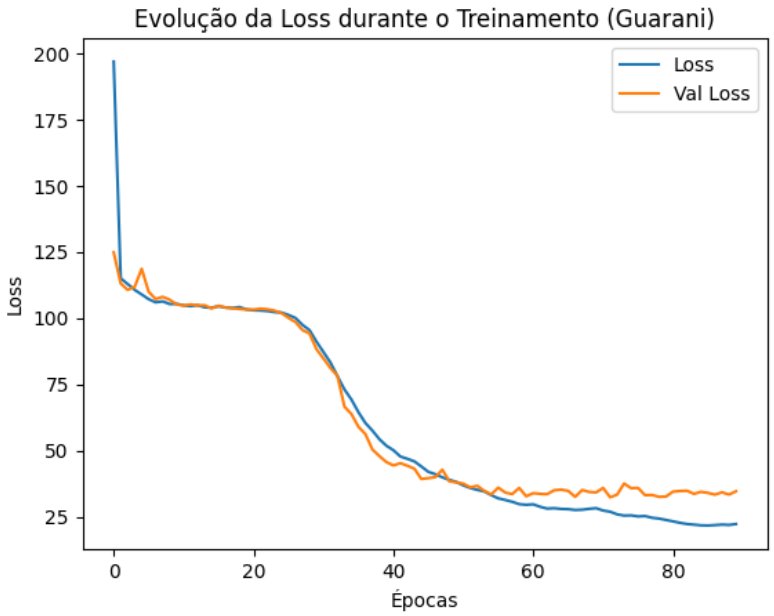
Já a mudança na quantidade de camadas recorrentes, que gerou a diminuição de 4.707.311 de parâmetros, fez com que o treinamento até a quinquagésima época fosse acelerado em 12% em comparação com a rede completa, com WER e *loss* similares a estrutura da rede inicial, com WER mínimo, até a quinquagésima época, de 0,78 para a rede reduzida e 0,79 para a rede original, enquanto o *loss* mínimo na rede reduzida foi de 38,01 em 50 épocas, e 33,02 em todo o treinamento, para a rede original foi de 44,61. Porém a rede demorou mais para alcançar uma convergência. O treinamento total foi programado para 90 épocas (Figura 16), com a duração total de 2 horas e 22 minutos, enquanto a rede completa, para 50 épocas, foi treinada em 1 hora e 30 minutos.

Figura 15: WER na base de teste para o dataset em guarani ao longo das épocas, rede reduzida



Fonte: Elaborada pelo autor.

Figura 16: Desempenho da rede reduzida para o dataset em guarani ao longo das épocas



Fonte: Elaborada pelo autor.

Ou seja, em baseado de métricas, a rede reduzida obteve resultados levemente

melhores no mesmo intervalo de épocas. Além disso, apresentou menos **overfitting** e maior estabilidade no WER durante o treinamento (Figura 15), em comparação com a rede completa (Figura 13).

A seguir tem-se inferências com a rede reduzida treinada até 90 épocas, os mesmos exemplos utilizados para a rede completa:

Para uma frase dita por um falante do sexo masculino:

- **Frase original:** "amombe'uporãta ndéve mba'éichapa reg̃uahẽkuaa ore rogamíme".
- **Frase predita (rede completa, 50 épocas):** "amombuporãte ndee mba'éichapere ñohekua ore rógamíme".
- **Frase predita (rede reduzida, 90 épocas):** "amomo porãtande mba'eichapere g̃uahẽguaorerogaíme".

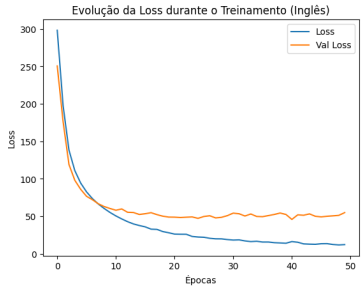
Para uma frase dita por um falante do sexo feminino:

- **Frase original:** "opa rire ñembo'eguasu añemboja ha añemongeta karai pa'i ndive".
- **Frase predita (rede completa, 50 épocas):** "opái re ñembo'eguasupa ñemboja ha ñemoneta karaipa'inive".
- **Frase predita (rede reduzida, 90 épocas):** "opaireñembo'eguacsu pa ñembojamaha ñemoreta kareipa'indive".

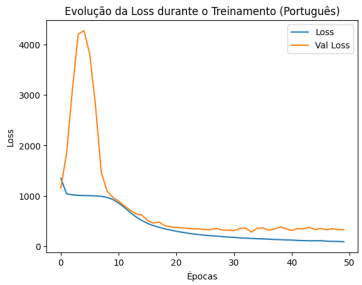
Para esses exemplos, a rede original completa demonstrou mais competência na inferência. Definiu melhor o espaço entre as palavras e necessita de menos correções para chegar a transcrição desejada.

A Figura 17 apresenta os valores de *loss* e WER para os treinamentos das redes principais. Como os valores são muito variáveis, eles não estão na mesma escala.

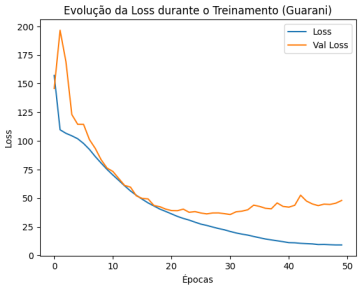
Figura 17: Gráficos de loss e WER das redes de inglês, português e guarani (rede completa).



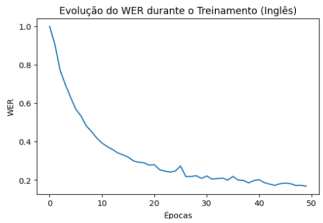
(a) Loss inglês.



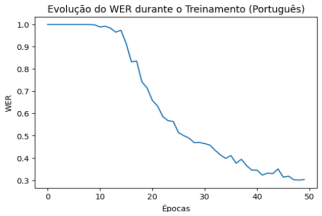
(b) Loss Português.



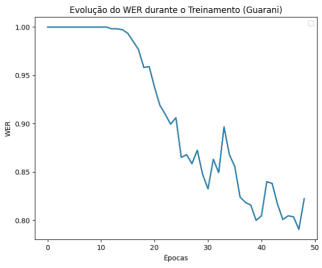
(c) Loss Guarani.



(d) WER inglês.



(e) WER português.



(f) WER guarani.

4.5 Inferência para dados externos

Nesta subseção, apresentamos os resultados da inferência realizada para dados externos, ou seja, áudios que não foram utilizados no treinamento das redes neurais. Para cada idioma — português, inglês e guarani — foram utilizados áudios gravados por três diferentes pessoas: um áudio em português por mim, um em inglês pela minha namorada, e um em guarani pela namorada do meu amigo. Todos os áudios foram gravados com microfones bons e ambientes sonoramente controlados. Os resultados demonstram como as redes treinadas com apenas um falante tiveram dificuldade em generalizar para vozes externas, como esperado. Foram gravados 2 áudios para cada língua.

Português:

- **Frase original (1):** "E aqui eu estou fazendo um teste em português com a minha própria voz."
- **Predição (1):** "m m nia reiu que os vazenaomuleis e uglor ptuini e s lo a mulemeraovriago aesno"
- **Frase original (2):** "Esse é um áudio para demonstrar que, este modelo que foi treinado com apenas um falante, vai ter dificuldades em transcrever outras vozes,

como a minha."

- **Predição (2):** "unt ro ee seao oto muar amenmo onstra as e emtgu andeiamo fotereanoano remnaico falra ente ig laa em diquio cundados emprastrereroa ou prasvoes o ai"

Inglês:

- **Frase original (1):** "Now I am doing a test in English with a female voice."
- **Predição (1):** "fo oud arcupoffel hr"
- **Frase original (2):** "This neural network cannot generalize, it was trained with just one speaker."
- **Predição (2):** "fo folert trengte tk hus isuent atrinsetd tefh"

Guarani:

- **Frase original (1):** "marã pi ko nde reipota upea, che rejamína."
- **Predição (1):** "ymahãm pi kane reipotá v upea che rejamíma."
- **Frase original (2):** "eichapa ne rei me? ko asaje porã. Che aime porã ha nde, mba'ei chapa ? Moõpa jahata tedia?"
- **Predição (2):** "eichapa me me ime ko 'anãjee porã eve porã onme mba'éi chepa vópe ñohkue pevia"

Os resultados indicam que o modelo treinado com apenas um falante por idioma apresentou dificuldade considerável para generalizar as predições em vozes de falantes diferentes. As transcrições preditas mostram uma grande discrepância em relação às frases originais, confirmando que o modelo não é capaz de lidar bem com variações de timbre, entonação e ritmo de fala, características de outros falantes, reforçando a necessidade de um treinamento com múltiplos falantes para maior robustez.

Apesar do Guarani ter apresentado os piores resultados nas métricas de WER durante o treinamento, foi a única rede que demonstrou alguma capacidade de generalização nos testes com dados externos. Isso pode ser atribuído ao fato de o conjunto de dados em Guarani ter sido treinado com múltiplos falantes, ao contrário dos dados em

português e inglês, que consistiam apenas em um único locutor. A presença de variações de entonação, ritmo e pronúncia no dataset de treinamento do Guarani pode ter proporcionado ao modelo uma maior flexibilidade para lidar com diferentes vozes, resultando em predições menos discrepantes, mesmo que a performance geral ainda tenha sido limitada.

5 Conclusão

Este trabalho investigou a aplicação de redes neurais baseadas na arquitetura Deep Speech 2 para a transcrição automática de fala em três línguas: português, inglês e guarani. Ao longo dos experimentos, constatamos que o desempenho do modelo variou significativamente entre as línguas, com os WERs mais baixos observados em português e inglês, e maiores desafios enfrentados com o guarani.

Os resultados dos experimentos demonstraram que, embora o modelo tenha apresentado bom desempenho em português e inglês, atingindo WERs relativamente baixos de, respectivamente, $0,3$ e $0,17$, o desafio foi maior para o guarani. A heterogeneidade dos dados em guarani, decorrente da presença de múltiplos falantes, variações de sotaque e uma quantidade de dados de treinamento baixa, contribuiu para um WER mais elevado de $0,79$ e sinais de *overfitting* durante o treinamento.

Apesar dos desafios, o modelo treinado para o guarani foi o único que mostrou capacidade de generalização para dados externos, devido à diversidade de falantes no conjunto de treinamento. Essa generalização sugere que a exposição a uma maior variação de vozes pode melhorar a robustez do modelo, um aspecto importante a ser explorado em trabalhos futuros com línguas de menor documentação.

Melhorias futuras podem incluir o uso de dados de maior qualidade e volume, o que permitiria uma melhor representação da língua e a mitigação dos efeitos de *overfitting*. Além disso, a investigação de diferentes variações de arquiteturas de redes neurais, aliada a métodos avançados de otimização, como a otimização bayesiana, poderia refinar ainda mais os resultados. Técnicas como *transfer learning* também são promissoras para aprimorar o desempenho, especialmente para línguas com recursos limitados. É válido também testar modelos de linguagem de correção de texto, que geram correções no texto predito.

Portanto, além de demonstrar a viabilidade de utilizar redes neurais para a transcrição automática em línguas majoritárias, este estudo ressalta os desafios e oportunidades específicos ao lidar com línguas menos comuns, como o guarani. Mesmo com limitações de dados, a transcrição automática com redes neurais profundas se mostra uma ferramenta promissora, com potencial para auxiliar especialistas em processos manuais e contribuir para a preservação de línguas minoritárias.

Referências

AMODEI, D. et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In: *Proceedings of The 33rd International Conference on Machine Learning (ICML)*. [S.l.: s.n.], 2016.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006.

BOUADJENEK, M.; HUYNH, N. *Description: Training a CTC-based model for automatic speech recognition*. 2021. Acesso em: 14 de abril 2024. Disponível em: ⟨https://keras.io/examples/audio/ctc_asr/⟩.

BRIDLE, J. S. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing*, Elsevier, v. 6, n. 3, p. 227–236, 1990.

CHOLLET, F.; ALLAIRE, J. *Deep Learning with R*. Shelter Island, New York: Manning Publications, 2018. ISBN 1-63835-163-5.

FB Audio Corpora. *FB Audio Corpora*. 2019. Acesso em: 19 de abril 2024. Disponível em: ⟨https://gitlab.com/fb-audio-corpora⟩.

FOUNDATION, M. *Mozilla Common Voice*. 2024. Acesso em 5 de agosto, 2024. Disponível em: ⟨https://commonvoice.mozilla.org/en/datasets⟩.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. ⟨http://www.deeplearningbook.org⟩.

HAHNLOSER, R. H. et al. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, Nature Publishing Group, v. 405, n. 6789, p. 947–951, 2000.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*. [S.l.]: Prentice Hall, 1998.

ITO, K. *The LJ Speech Dataset*. 2017. Acesso em: 19 de abril 2024. Disponível em: ⟨https://keithito.com/LJ-Speech-Dataset/⟩.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, 2015.

Apêndice

A Códigos em R

```
####
#Conversão dos dados de Guarani para 16.000hz e para Wave
####

library(tidyverse)
library(readr)

# Caminho Base
setwd("D:/UNB/11_semestre/TCC1/Dados_GUARANI/")

# Leitura do txt para modificações
validated <- read_delim("validated.txt",
                        delim = "\t", escape_double = FALSE,
                        trim_ws = TRUE) %>% select(path, sentence)

mp3_vetor_validados = validated$path

library(tuneR)

# Diretórios
input_dir <- "clips"
output_dir <- "Dados_estruturados/wavs"

# Listar todos os arquivos MP3 na pasta de entrada
mp3_files <-
    list.files(input_dir, pattern = "\\.mp3$", full.names = T) %>%
    as.tibble() %>%
    mutate(verifica = str_replace(value,"clips/", "")) %>%
    filter(verifica %in% mp3_vetor_validados) %>%
    select(value) %>% pull()
```

```
# Função para converter MP3 para WAV no formato correto
convert_mp3_to_wav <- function(mp3_file, output_dir, contador) {
  # Ler o arquivo MP3
  audio_mp3 <- readMP3(mp3_file)

  # Definir a taxa de amostragem para 16000 Hz
  target_sample_rate <- 16000

  # Resample o áudio para a taxa de amostragem desejada
  audio_resampled <- downsample(audio_mp3, target_sample_rate)

  # Normalizar o áudio para garantir que esteja no formato PCM de 16 bits
  audio_normalized <- normalize(audio_resampled, unit = "16")

  # Criar o caminho para salvar o arquivo WAV
  file_name <- basename(mp3_file)
  wav_file <- sub("\\.mp3$", ".wav", file_name)
  output_path <- file.path(output_dir, wav_file)

  # Salvar o áudio como WAV (não extensível)
  writeWave(audio_normalized, output_path, extensible = FALSE)

  # Verificar se a conversão foi bem-sucedida

  if (file.exists(output_path)) {
    cat("Conversão concluída com sucesso: ",
            output_path, "\n",
            round(contador / 2520, 2), "\n")

  } else {
    cat("Erro na conversão de: ", mp3_file, "\n")
  }
}


# Converter todos os arquivos MP3 para WAV
```

```
contador = 1
for (mp3_file in mp3_files) {
  convert_mp3_to_wav(mp3_file, output_dir, contador)
  contador = contador +1
}

cat("Conversão de todos os arquivos concluída.\n")

validated %>% mutate(path = str_sub(path,1,-5),
                     sentence_abnt = gsub("[,!.?\";:()]", "",sentence_abnt)) %>%
                     select(1,3) -> metadata

write.table(metadata,"metadata.csv", col.names = F)



####
#Análise descritiva Guarani
####

# Carrega os pacotes necessários
library(tidyverse)
library(readr)

# Lê o arquivo validated.txt e seleciona as colunas de interesse
validated_sentences <- read_delim("validated.txt",
                                  delim = "\t", escape_double = FALSE,
                                  trim_ws = TRUE) %>%
                       select(8, 9)

# Gera uma tabela com a contagem de ocorrências por gênero e faixa etária
table(validated_sentences)

# Agrupa os dados por gênero e idade, removendo valores ausentes
bs = validated_sentences %>%
  group_by(gender, age) %>%
  summarise(n = n()) %>%
```

```
  na.omit() %>%
  # Ajusta valores das variáveis para um formato mais legível
  mutate(gender = case_when(gender == 'female_feminine' ~ 'Feminino',
                            TRUE ~ 'Masculino'),
         age = case_when(age == 'teens' ~ "13-19",
                         age == 'twenties' ~ "20-29",
                         age == 'thirties' ~ "30-39",
                         age == 'fourties' ~ "40-49",
                         age == 'twenties' ~ "50-59",
                         age == 'sixties' ~ "60-69")) %>%
  rename('Gênero' = 'gender')

# Cria um gráfico de barras mostrando a quantidade de áudios por gênero e faixa etária
ggplot(bs, aes(x = age, y = n, fill = Gênero)) +
  geom_col() + theme_bw() +
  labs(y = "Quantidade", x = "Faixa Etária",
       title = "Quantidade de áudios classificados \n por sexo e faixa etária")


####
#Extração dos caracteres da base Guarani
####

caracteres_unicos <- unique(unlist(strsplit(as.character(tolower(validated$X2)), "")))

# Ordenar os caracteres e colapsar em uma string
caracteres_unicos_ordenados <- paste(sort(caracteres_unicos), collapse = "")

# Capturar todos os caracteres de treinamento e teste
print(caracteres_unicos_ordenados)
```

B Códigos em Python

```
####
#Importação de bibliotecas google colab
####
```

```
# Importação de pacotes necessários para processamento e modelagem
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython import display
from jiwer import wer
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

####
#Importação dos dados em inglês
####

# URL do conjunto de dados de fala
url_dados = "https://data.keithito.com/data/speech/LJSpeech-1.1.tar.bz2"
caminho_dados = keras.utils.get_file("LJSpeech-1.1", url_dados, untar=True)
caminho_wavs = caminho_dados + "/wavs/"
caminho_metadata = caminho_dados + "/metadata.csv"

# Leitura e processamento do arquivo de metadados
df_metadados = pd.read_csv(caminho_metadata, sep="|", header=None, quoting=3)
df_metadados.columns = ["nome_arquivo", "transcricao", "transcricao_normalizada"]
df_metadados = df_metadados[["nome_arquivo", "transcricao_normalizada"]]
df_metadados = df_metadados.sample(frac=1).reset_index(drop=True)

####
#Importação dos dados em português e guarani
####

# Definindo os caminhos para os dados e arquivos de áudio
diretorio_base = '/content/'

# Caminho para os arquivos de áudio
diretorio_audios = f"{diretorio_base}wavs/"
# Caminho para o arquivo de metadados
```

```
arquivo_metadados = f"{diretorio_base}metadata.csv"

# Leitura e embaralhamento dos metadados
df_audios = pd.read_csv(arquivo_metadados, delimiter=",", header=None, quoting=2)
df_audios.columns = ["arquivo_audio", "texto_normalizado"]
df_audios = df_audios.sample(frac=1).reset_index(drop=True)

####
#Divisão do Conjunto de Dados e Mapeamento de Caracteres
####

# Dividindo o conjunto de dados em treino e validação
divisao = int(len(df_metadados) * 0.90)
df_treino = df_metadados[:divisao]
df_validacao = df_metadados[divisao:]

print(f"Tamanho do conjunto de treino: {len(df_treino)}")
print(f"Tamanho do conjunto de validação: {len(df_validacao)}")

# Conjunto de caracteres aceitos na transcrição
caracteres_aceitos = [c for c in "'- '\"aáãbcdeéfgg̃hiíĩjklmnñoóõpqrstuúũvxyýz"]

# Para o caso de português ou inglês
# caracteres_aceitos = [c for c in " abcdefghijklmnopqrstuvwxyz"]

# Mapeamento de caracteres para inteiros
char_para_num = keras.layers.StringLookup(vocabulary=caracteres_aceitos, oov_token="")
# Mapeamento de inteiros para caracteres originais
num_para_char = keras.layers.StringLookup(
    vocabulary=char_para_num.get_vocabulary(), oov_token="", invert=True
)

print(
    f"O vocabulário é: {char_para_num.get_vocabulary()} "
    f"(tamanho = {char_para_num.vocabulary_size()})"
)
```

```
####
#Processamento de Amostras de Áudio e Rótulos
####

# Definindo parâmetros do processamento de áudio
tam_frame = 256  # Tamanho do frame em amostras
passo_entre_frames = 160  # Passo entre os frames
tam_fft = 384  # Tamanho do FFT aplicado

def processar_amostra(arquivo_audio, rotulo_texto):

    # Carregar o arquivo de áudio
    caminho_completo = caminho_wavs + arquivo_audio + ".wav"
    conteudo_audio = tf.io.read_file(caminho_completo)

    # Decodificar o conteúdo do arquivo WAV
    sinal_audio, _ = tf.audio.decode_wav(conteudo_audio)
    sinal_audio = tf.squeeze(sinal_audio, axis=-1)

    # Converter o sinal para o tipo float32
    sinal_audio = tf.cast(sinal_audio, tf.float32)

    # Calcular o espectrograma
    espectrograma_audio = tf.signal.stft(
        sinal_audio, frame_length=tam_frame,
        frame_step=passo_entre_frames, fft_length=tam_fft
    )

    # Extrair apenas a magnitude do espectrograma
    espec_mag = tf.abs(espectrograma_audio)
    espec_mag = tf.math.pow(espec_mag, 0.5)

    # Normalização do espectrograma
    media_valores = tf.math.reduce_mean(espec_mag,
                                        axis=1, keepdims=True)
    desv_padrao = tf.math.reduce_std(espec_mag,
```

```
                                    axis=1, keepdims=True)
    espec_norm = (espec_mag - media_valores) / (desv_padrao + 1e-10)

    # Converter o rótulo para minúsculas
    rotulo_texto = tf.strings.lower(rotulo_texto)

    # Separar o rótulo em caracteres individuais
    rotulo_caracteres = tf.strings.unicode_split(rotulo_texto,
                                                 input_encoding="UTF-8")

    # Converter os caracteres para seus correspondentes numéricos
    rotulo_numerico = char_para_num(rotulo_caracteres)

    # Retornar o espectrograma e o rótulo processados
    return espec_norm, rotulo_numerico


####
#Configuração dos Conjuntos de Dados de Treinamento e Validação
####

tam_lote = 32  # Tamanho do batch

# Definição do dataset de treinamento
dados_treino = tf.data.Dataset.from_tensor_slices(
    (df_treino["nome_arquivo"].tolist(), df_treino["transcricao_normalizada"].tolist())
)
dados_treino = (
    dados_treino.map(codificar_amostra_unica, num_parallel_calls=tf.data.AUTOTUNE)
    .padded_batch(tam_lote)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)

# Definição do dataset de validação
dados_validacao = tf.data.Dataset.from_tensor_slices(
    (df_validacao["nome_arquivo"].tolist(),
```

```
        df_validacao["transcricao_normalizada"].tolist())
)
dados_validacao = (
    dados_validacao.map(codificar_amostra_unica, num_parallel_calls=tf.data.AUTOTUN
    .padded_batch(tam_lote)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)


####
#Implementação da Função de Perda CTC
####

def perda_CTC(rotulo_verdadeiro, predicao):
    # Calcular o valor da perda durante o treinamento
    # Tamanho do lote (batch size)
    tamanho_lote = tf.cast(tf.shape(rotulo_verdadeiro)[0], dtype="int64")

    # Comprimento da entrada (predição)
    comprimento_entry = tf.cast(tf.shape(predicao)[1], dtype="int64")

    # Comprimento do rótulo verdadeiro
    comprimento_rotulo = tf.cast(tf.shape(rotulo_verdadeiro)[1], dtype="int64")

    # Criar tensores com os comprimentos das entradas e dos rótulos
    comprimento_entry = comprimento_entry * tf.ones(shape=(tamanho_lote, 1),
                                                    dtype="int64")
    comprimento_rotulo = comprimento_rotulo * tf.ones(shape=(tamanho_lote, 1),
                                                      dtype="int64")

    # Calcular a perda CTC para o lote
    perda = keras.backend.ctc_batch_cost(rotulo_verdadeiro,
                                         predicao,
                                         comprimento_entry,
                                         comprimento_rotulo)
```

```
    return perda  # Retornar o valor da perda


####
#Definição do modelo
####


def construir_modelo(dimensione_entrada,
                        dimensione_saida,
                        camadas_rnn=5,
                        unidades_rnn=128):

    # Entrada do espectrograma, baseada na dimensão de entrada
    espectrograma_entrada = layers.Input(shape=(None, dimensione_entrada),
                                            name="entrada_espectrograma")

    # Expansão das dimensões para usar a CNN 2D
    x = layers.Reshape(target_shape=(-1, dimensione_entrada, 1),
                        name="expandir_dimensao")(espectrograma_entrada)

    # Primeira camada convolucional
    x = layers.Conv2D(
        filters=32,
        kernel_size=(11, 41),
        strides=(2, 2),
        padding="same",
        use_bias=False,
        name="convolucao_1",
    )(x)
    x = layers.BatchNormalization(name="batchnorm_1")(x)
    x = layers.Activation("relu", name="relu_1")(x)

    # Segunda camada convolucional
    x = layers.Conv2D(
        filters=32,
```

```
        kernel_size=(11, 21),
        strides=(1, 2),
        padding="same",
        use_bias=False,
        name="convolucao_2",
    )(x)
    x = layers.BatchNormalization(name="batchnorm_2")(x)
    x = layers.Activation("relu", name="relu_2")(x)

    # Remodelar a saída da CNN para servir de entrada para as camadas RNN
    x = layers.Reshape(target_shape=(-1, x.shape[-2] * x.shape[-1]),
                                        name="remodelar")(x)

    # Adicionar camadas RNN bidirecionais
    for camada in range(1, camadas_rnn + 1):
        gru = layers.GRU(
            units=unidades_rnn,
            activation="tanh",
            recurrent_activation="sigmoid",
            use_bias=True,
            return_sequences=True,
            reset_after=True,
            name=f"gru_{camada}",
        )
        x = layers.Bidirectional(
            gru, name=f"bidirecional_{camada}", merge_mode="concat"
        )(x)
        if camada < camadas_rnn:
            x = layers.Dropout(rate=0.5, name=f"dropout_{camada}")(x)

    # Camada densa para processamento adicional
    x = layers.Dense(units=unidades_rnn * 2, name="densa_1")(x)
    x = layers.Activation("relu", name="densa_1_relu")(x)
    x = layers.Dropout(rate=0.5, name="dropout_final")(x)

    # Camada de classificação final
```

```
    saida = layers.Dense(units=dimensione_saida + 1,
                         activation="softmax",
                         name="saida_classificacao")(x)

    # Definir o modelo
    modelo = keras.Model(inputs=espectrograma_entrada,
                         outputs=saida,
                         name="Modelo_Audio")

    # Compilação do modelo com otimizador Adam
    otimizador = keras.optimizers.Adam(learning_rate=1e-4)
    modelo.compile(optimizer=otimizador, loss=perda_CTC)

    return modelo

# Configurando o modelo com dimensões específicas e exibindo o resumo
modelo = construir_modelo(
    dimensione_entrada=tamanho_fft // 2 + 1,
    dimensione_saida=char_para_num.vocabulary_size(),
    unidades_rnn=512,
)



####
#Decodificação de Saídas e Avaliação Durante o Treinamento
####

# Função utilitária para decodificar a saída da rede
def decodificar_predicoes_lote(predicoes):
    comprimento_entrada = np.ones(predicoes.shape[0]) * predicoes.shape[1]
    resultados = keras.backend.ctc_decode(
        predicoes, input_length=comprimento_entrada, greedy=True
    )[0][0]
    # Iterar sobre os resultados e recuperar o texto
    texto_saida = []
    for resultado in resultados:
```

```
        resultado = tf.strings.reduce_join(
            num_para_char(resultado)
        ).numpy().decode("utf-8")
        texto_saida.append(resultado)
    return texto_saida

# Classe de callback para exibir algumas transcrições durante o treinamento
class AvaliacaoCallback(keras.callbacks.Callback):

    def __init__(self, dados_validacao):
        super().__init__()
        self.dados_validacao = dados_validacao

    def on_epoch_end(self, epoca: int, logs=None):
        predicoes = []
        rotulos_verdadeiros = []
        for lote in self.dados_validacao:
            entradas, rotulos = lote
            predicoes_lote = modelo.predict(entradas)
            predicoes_lote = decodificar_predicoes_lote(predicoes_lote)
            predicoes.extend(predicoes_lote)
            for rotulo in rotulos:
                rotulo = tf.strings.reduce_join(
                    num_para_char(rotulo)
                ).numpy().decode("utf-8")
                rotulos_verdadeiros.append(rotulo)
        # Calcular a taxa de erro de palavras (WER)
        pontuacao_wer = wer(rotulos_verdadeiros, predicoes)
        print("-" * 100)
        print(f"Taxa de Erro de Palavras: {pontuacao_wer:.4f}")
        print("-" * 100)
        for i in np.random.randint(0, len(predicoes), 2):
            print(f"Rótulo    : {rotulos_verdadeiros[i]}")
            print(f"Previsão  : {predicoes[i]}")
            print("-" * 100)
```

```
####
#Forçar a utilização da placa de vídeo do colab
####

from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())

from keras import backend as K
K.tensorflow_backend._get_available_gpus()

####
#Treinamento
####

# Define o número de épocas para o treinamento
epocas = 50

# Callback para avaliar as transcrições no conjunto de validação
callback_validacao = AvaliacaoCallback(dados_validacao)

# Variáveis para armazenar os melhores pesos e a menor perda de validação
melhor_perda_validacao = float('inf')
melhores_pesos = None

# Dicionário para acumular o histórico de métricas
historico_acumulado = {
    'perda': [], 'perda_validacao': [],
    'acuracia': [], 'acuracia_validacao': []
}

# Treinar o modelo
with tf.device('/device:GPU:0'):
    for epoca in range(epocas):
        historico = modelo.fit(
            dados_treino,
            validation_data=dados_validacao,
```

```
            epochs=1,  # Treina uma época por vez
            callbacks=[callback_validacao],
        )

        # Acumular dados de histórico
        for chave in historico_acumulado.keys():
            if chave in historico.history:
                historico_acumulado[chave].extend(historico.history[chave])

        # Obter a perda de validação após cada época
        perda_validacao = historico.history['val_loss'][-1]

        # Verificar se a perda de validação é a melhor até agora
        if perda_validacao < melhor_perda_validacao:
            melhor_perda_validacao = perda_validacao
            melhores_pesos = modelo.get_weights()  # Salvar os melhores pesos
            print(
                f'Melhor val_loss melhorou para {melhor_perda_validacao:.4f} '
                f'na época {epoca + 1}'
            )

        print(f'Época {epoca + 1} de {epocas}')


####
#Selecionar pesos finais e aplicar inferência
####

# Após o treinamento, carregar os melhores pesos
model.set_weights(best_weights)

# Inferencia
previsoes = []
rotulos_verdadeiros = []

for lote in dados_validacao:
```

```
    entradas, rotulos = lote
    previsoes_lote = modelo.predict(entradas)
    previsoes_lote = decodificar_predicoes_lote(previsoes_lote)
    previsoes.extend(previsoes_lote)
    for rotulo in rotulos:
        rotulo_decodificado = tf.strings.reduce_join(
            num_para_char(rotulo)
        ).numpy().decode("utf-8")
        rotulos_verdadeiros.append(rotulo_decodificado)

# Calcular a taxa de erro de palavras (WER)
pontuacao_wer = wer(rotulos_verdadeiros, previsoes)
print("-" * 100)
print(f"Taxa de Erro de Palavras: {pontuacao_wer:.4f}")
print("-" * 100)

# Exibir algumas previsões e seus rótulos correspondentes
for i in range(5):
    print(f"Rótulo    : {rotulos_verdadeiros[i]}")
    print(f"Previsão  : {previsoes[i]}")
    print("-" * 100)



####
#Criar os gráficos finais
####

# Plotar a perda
plt.plot(accumulated_history['loss'], label='Loss')
plt.plot(accumulated_history['val_loss'], label='Val Loss')
plt.title('Evolução da Loss durante o Treinamento (Guarani)')
plt.xlabel('Épocas')
plt.ylabel('Loss')
plt.legend()
plt.show()
```