
Trabalho de Conclusão de Curso

# Avaliação de arquiteturas de redes neurais profundas para reconhecimento de sinais dinâmicos de Libras

Lucas Antônio Ferro do Amaral
lafa@ic.ufal.br

Orientadores:

Thales Miranda de Almeida Vieira
Tiago Figueiredo Vieira

Maceió, Abril de 2019

Lucas Antônio Ferro do Amaral

# Avaliação de arquiteturas de redes neurais profundas para reconhecimento de sinais dinâmicos de Libras

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência de Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientadores:

Thales Miranda de Almeida Vieira

Tiago Figueiredo Vieira

Maceió, Abril de 2019

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência de Computação do Instituto de Computação da Universidade Federal de Alagoas, aprovada pela comissão examinadora que abaixo assina.

---

Thales Miranda de Almeida Vieira - Orientador
Universidade Federal de Alagoas

---

Tiago Figueiredo Vieira - Orientador
Universidade Federal de Alagoas

---

Marcelo Costa Oliveira - Examinador
Instituto de Computação
Universidade Federal de Alagoas

Maceió, Abril de 2019

# Agradecimentos

A minha família por todo o suporte que recebi.
Aos meus orientadores, Thales Vieira e Tiago Vieira, que tornam possível a realização dessa pesquisa, todo apoio oferecido e principalmente a dedicação, que foram essenciais para conclusão desse trabalho.
Gostaria de agradecer a todos meus amigos que ajudaram para concluir esse trabalho, em especial agradeço aos meus amigos Antônio Manoel e Roland Gonçalves, por todo apoio e sugestões.
Agradeço a CNPq pela bolsa de iniciação cientifica.

*"A sorte favorece os audazes"*

– Alexandre, *O Grande*

# Resumo

Neste trabalho foram estudados algoritmos de aprendizado de máquina para o treinamento e reconhecimento de sinais dinâmicos da Língua Brasileira de Sinais (Libras). Mais especificamente, propomos e investigamos o uso de arquiteturas de redes neurais profundas para o reconhecimento de gestos dinâmicos, com foco no reconhecimento de sinais dinâmicos de Libras, usando imagens de profundidade. Examinamos combinações e variações de redes neurais compostas por camadas convolucionais e recorrentes, incluindo modelos de Long-term Recurrent Convolutional Networks (LRCNs) e Redes Neurais Convolucionais 3D (CNN3D). Experimentos foram realizados com uma nova base de dados que coletamos e disponibilizamos publicamente, composta por sinais dinâmicos representando algumas letras do alfabeto e algumas palavras comuns em Libras. A avaliação dos modelos propostos revela que os melhores modelos profundos atingem mais de 99% de acurácia e superam amplamente um método padrão.

**Palavras-chave**: Aprendizado de máquina, Redes Neurais, Libras, Visão Computacional

# Abstract

In this work we studied machine learning algorithms for training and recognizing dynamic signals of the Brazilian Sign Language (Libras). More specifically, we propose and investigate deep neural networks architectures for dynamic gesture recognition from depth data. We investigate combinations and variants of convolutional and recurrent layers, including Long-term Recurrent Convolutional Networks (LRCNs) and 3D Convolutional Neural Networks (3DCNNs). Experiments were performed using a novel dataset that we collected and are publicly available, composed of dynamic signs representing letters of the alphabet and a few words in Libras. The evaluation of the proposed models reveals that the best deep models achieve above 99 % of accuracy, and greatly outperforms a baseline method.

**Key-words**: Machine Learning, neural networks, Libras, Computer Vision

# Lista de Figuras

# Conteúdo

# Capítulo 1

# Introdução

## 1.1 Contextualização

Linguagens de sinais utilizadas por comunidades de surdos e mudos são importantes para a inclusão social dessas pessoas, pois permite construir um meio de comunicação entre elas. A inclusão social para pessoas surdas e mudas possibilita que elas se desenvolvam em vários pontos. Um deles, é o aspecto intelectual, possibilitando a comunicação entre professores ouvintes e alunos surdos e vice-versa, por exemplo. Porém, a comunidade de surdos fluentes em alguma língua de sinais é pequena quando comparada com a comunidade de falantes de linguagens orais. A inserção dessas pessoas utilizando alguma espécie de tradução de linguagem de sinais para uma língua mais amplamente falada como o português é de grande importância para essa comunidade (Felipe, 2007). Neste contexto, computadores têm o potencial de possibilitar o desenvolvimento de ferramentas automáticas de tradução.

Através dos últimos anos, tivemos avanços tecnológicos com sensores de profundidade e métodos de visão computacional baseados em aprendizagem profunda (*deep learning*), nos possibilitando construir métodos que sejam capazes de reconhecer sinais libras em tempo real com alta precisão.

Reconhecer sinais dinâmicos de Libras em tempo real usando aprendizado de máquina é uma tarefa desafiadora, devido a variações nas maneiras de se realizar um sinal: uma mesma pessoa vai realizando um determinado sinal com pequenas variações, que a máquina deve ser capaz de generalizar e reconhecer. Identificar as características relevantes das configuração das mãos, expressões faciais e corporais também é um problema complexo.

Para construção de um classificador robusto, é necessária uma base de dados apropriada, com amostras representativas. Construir uma base de dados com vídeos dos sinais de libras é uma tarefa trabalhosa, pois envolve cortar sinais de conversas ou cortar partes desinteressantes de transições no início da execução ou ao final, o que pode levar a um trabalho manual grande.

## 1.2 Trabalhos relacionados

Reconhecer sinais dinâmicos de Libras ou outras línguas de sinais usando o computador é uma tarefa que poucos estudos tentaram (Pizzolato et al., 2010), (Escobedo Cardenas and Camara-Chavez, 2018), (Pizzolato et al., 2010) e (Kim et al., 1996). Em particular, nenhum desses trabalhos investigou o uso de arquiteturas profundas de redes neurais, como as redes neurais recorrentes (RNN) ou redes neurais convolucionais tridimensionais (CNN3D).

O reconhecimento de sinais através de imagens usando o computador é uma alternativa. O reconhecimento de sinais estáticos já possui boas soluções na literatura que apresentam ótima acurácia para reconhecimento. Um desses métodos como o dos autores Cardenas and Chávez (2015), que constrói um histograma de magnitudes cumulativas dos cossenos diretores do centroide em uma nuvem de pontos e classifica o histograma utilizando um SVM. Por outro lado, o autor Pizzolato et al. (2010), utiliza extração manual de características e classificando com uma rede neural 2 camadas Densas. O trabalho de Bastos et al. (2015) consiste também da utilização de extratores características e classificação com redes neurais com apenas camadas Densas, onde todos os métodos chegam a alcançar mais de 96% de acurácia em sinais estáticos.

Outros trabalhos têm como objetivo identificar sinais dinâmicos, em geral, ou de alguma linguagem específica, tal como Libras. O trabalho de Pizzolato et al. (2010), que mencionamos acima para sinais estáticos, mas que também propõe um método para reconhecer sinais dinâmicos utilizando redes neurais e imagens de profundidade, com um passo para filtrar o fundo e cortar a imagem, um passo que extrai a borda das imagens, e depois as envia para classificação usando uma rede neural multicamadas (MLP) com 2 camadas densas. No trabalho de de Souza Pereira Moreira et al. (2014) também temos a utilização de dados provenientes de sensores de profundidade: um pré-processamento normaliza os pontos da imagem em relação ao centro da mão e em seguida uma MLP é utilizada para classificação. Em Escobedo Cardenas and Camara-Chavez (2018), uma forma mais moderna de atacar o problema de reconhecimento de sinais dinâmicos é apresentada, utilizando uma combinação de arquiteturas de redes neurais convolucionais, para extrair características do esqueleto e dos sinais de imagens de profundidade separadamente pelas arquiteturas de redes neurais, e juntar a saída de ambas arquiteturas e classificar o resultado.

Diversos trabalhos tratam do reconhecimento de outras linguagens de sinais, como por exemplo a *Korean Sign Language*(KSL), que é investigada no trabalho de Kim et al. (1996). Nesse trabalho, uma luva é utilizada para capturar os sinais e fornecer os dados para uma rede neural *fuzzy min-max* para classificação. Em (Kurakin et al., 2012), um conjunto de 12 sinais dinâmicos da *American Sign Language*(ASL) é representados por vídeos de profundidade é avaliado usando um novo método que consiste de uma etapa de filtragem para normalizar a orientação dos sinais, uma etapa de extração de características, e só então etapa de classificação que utiliza um grafo de ação.

Em uma abordagem mais generalista de reconhecer gestos, sendo eles sinais de outras lín-

guas ou não, temos o trabalho de Wang et al. (2012), que detecta gestos utilizando vídeos de profundidade, passando a sequência de quadro a quadro ele filtra os quadros, extrai características para assim classificar o gesto utilizando um Modelo Oculto de Markov (HMM).

Não é de nosso conhecimento a existências de trabalhos que abordem o reconhecimento de sinais dinâmicos de Libras usando arquiteturas de redes neurais profundas.

## 1.3 Objetivos

O objetivo deste trabalho é avaliar arquiteturas profundas de redes neurais para reconhecimentos sinais dinâmicos de Libras. Em particular, investigamos arquiteturas baseadas em camadas convolucionais e recorrentes. Além disso, coletamos e disponibilizamos uma base de dados de sinais dinâmicos de libras utilizando vídeos compostos por imagens de profundidade.

Investigamos 2 classes de arquiteturas de rede neurais: uma combinação de camadas convolucionais bidimensionais com camadas recorrentes LSTM, chamada na literatura *Long-Term Recurrent Convolutional Network* (LRCN); e redes neurais convolucionais tridimensionais (CNN3D). Neste trabalho, focamos em classificar gestos que utilizem apenas as mãos (uma ou duas), pois, a inclusão de múltiplas interações envolvendo mãos com toques em localidades específicas do corpo possuem muita variação entre os usuários, e sinais que envolvem também expressões faciais e corporais seria necessário incluir mais um classificador ou outra arquitetura atrelada às atuais, assim podendo diminuir o poder da arquitetura. Conseguimos ter bons resultados com as melhores arquiteturas alcançando 99% de precisão na nossa base de dados. Esse trabalho foi publicado no XXIII CIARP (*Iberoamerican Congress on Pattern Recognition*, (Amaral et al., 2018))

O texto a seguir está dividido em mais cinco capítulos: uma breve descrição de conceitos básicos de Aprendizado de Máquina (Capítulo 2); um capítulo descrevendo tópicos de redes neurais relacionados com este trabalho (Capítulo 3); seguido pela Metodologia adotada neste trabalho (Capítulo 4). os experimentos realizados (Capítulo 5), mostrando a acurácia alcançada pelos melhores modelos e uma descrição de nossa base de dados que coletamos e publicamos, e por fim o apresentamos no Capítulo 6 as considerações finais e trabalhos futuros.

# Capítulo 2

# Aprendizado de Máquina

Aprendizado de Máquina é um dos ramos da Inteligência Artificial que estuda métodos para fazer uma máquina realizar tarefas específicas, sem terem sido programadas (Samuel, 1959). Aprendizado de Máquina pode ser definido como "Métodos que usam experiência(Dados) para melhorar o desempenho de fazer predições acuradas", segundo Mohri et al. (2012); ou "conjunto de métodos que podem automaticamente detectar padrões nos dados", segundo Murphy (2012). Um pouco diferente das definições anteriores Goodfellow et al. (2016) descreve Aprendizado de Máquina como "uma forma de estatística aplicada com ênfase no uso de computadores para estimar funções complicadas, e diminuir a ênfase em provar intervalos de confiança ao redor dessas funções". De fato, a área de Aprendizado de Máquina tem foco em utilização de dados, identificação de padrões e aprender a realizar uma tarefa.

## 2.1 Base de dados

Uma base de dados (Dataset) é um conjunto onde cada elemento consiste em uma amostra que foi coletada, e uma das utilidades de uma base de dados é servir de experiência para algoritmos de aprendizado. Em uma base de dados as amostras podem ser de duas formas, estruturadas e não estruturadas.

### 2.1.1 Dados estruturados

Amostras estruturadas consistem de um vetor de características $\mathbf{x} \in \mathbb{R}^n$, onde $n$ é a quantidade de características disponíveis para cada amostra nessa base de dados. Por exemplo, a base de dados Iris flowers (Dheeru and Karra Taniskidou, 2017) consiste em 150 amostras no total, com 50 amostras para cada uma das três espécies e possuindo as seguintes características disponíveis: Comprimento das Sépalas, Largura das Sépalas, comprimento das pétalas, largura das pétalas, com todas as características medidas em centímetros. Para essa base de dados, cada amostra é um vetor $\mathbf{x} \in \mathbb{R}^4$.

### 2.1.2 Dados Não-estruturados

Dados não-estruturados geralmente são dados que ainda não tiveram suas características extraídas ou organizados para representar informação contida neles. Comumente, dados não-estruturados são textos, áudio, imagens, vídeos e outras formas de dados multidimensionais. Atualmente já existem métodos de Aprendizado de Máquina que fazem essa etapa automaticamente, identificando as características presentes nos dados.

### 2.1.3 Rótulos

Amostras em uma base de dados podem possuir um rótulo indicando a classe a qual elas pertencem. Uma amostra representada por um vetor de características $x$ passa a ser uma tupla da forma $(x, y)$ onde $y$ é o rótulo relacionado a amostra. Esse rótulo não é de presença obrigatória, porém, é essencial para decidir qual tipo algoritmo de Aprendizado deverá ser aplicado.

## 2.2 Algoritmo de Aprendizado

A princípio, um algoritmo é um conjunto de regras e procedimentos lógicos, que levam a solucionar algum problema em um número finito de etapas. Entretanto, para serem suficientemente genéricos, algoritmos de aprendizado são construídos com objetivo de encontrar padrões nos dados.

Algoritmos de aprendizado buscam estimar uma função baseando-se nos dados que estão disponíveis em sua etapa de treino, otimizando uma métrica que avalia o seu desempenho na realização de uma tarefa, quando as amostras são rotuladas. Para amostras não rotuladas, já não é mais possível estimar o desempenho do algoritmo (Hastie et al., 2001).

É possível mensurar o erro entre rótulo predito e o rótulo real da amostra usando métricas de perda (*loss*). Por exemplo, o erro médio quadrado (MSE) dado por

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

é uma das métricas que podem ser utilizadas para avaliar o desempenho entre a predição e o esperado, com $\hat{y}$ sendo o valor predito e $y$ o rótulo. Existem outras funções que também servem para métricas de perda, como a entropia cruzada(cross-entropy) definida por

$$\text{cross-entropy}(\hat{y}, y) = -\sum_{i}^{n}\hat{y}_i ln(y_i), \tag{2.2}$$

a função *huber*, definida por

$$\text{huber}(\hat{y}, y) = \begin{cases} \frac{1}{2}(y-\hat{y})^2, & \text{se } |y-\hat{y}| \leq \delta \\ \delta(y - y\hat{y}) - \frac{1}{2}\delta, & \text{caso contrário} \end{cases} \tag{2.3}$$

e a função *hinge* definida por

$$\text{hinge}(\hat{y}, y) = \begin{cases} 0, & \text{se } 1 - y\hat{y} \leq 0 \\ 1 - y\hat{y}, & \text{caso contrário .} \end{cases} \tag{2.4}$$

## 2.3 Tipos de aprendizado de máquina

Vamos descrever abaixo três grandes tipos de aprendizado de máquina: aprendizado por reforço, aprendizado supervisionado e aprendizado não-supervisionado.

### 2.3.1 Por reforço

Um algoritmo de aprendizado por reforço é um agente que trabalha em atingir um objetivo fazendo interações com seu ambiente, mudando o estado atual do ambiente através de escolha de ações e recebendo uma recompensa para mudança de estado (Mohri et al., 2012).

Um agente é definido em Russell and Norvig (2009) como "qualquer coisa que percebe seu ambiente através de sensores, e age nesse ambiente através de atuadores", note que um modelo de aprendizado por reforço se encaixa nessas características, pois, ele percebe seu ambiente através dos estados e as recompensas e atua no ambiente pelas ações.

Um modelo amplamente utilizado para aprendizado por reforço é o processo de decisão Markoviano, que consiste em uma n-upla com seguintes itens:

- Conjunto de ações: $A$.
- Conjunto de estados: $S$.
- Estado inicial: $s_0 \in S$.
- Probabilidade de transição para os próximos estados, dado o estado atual e uma ação: $Pr[s_{n+1}|s_n, a]$.
- Probabilidade das recompensas para escolha da ação e o novo estado: $Pr[r_{n+1}|s, a]$

O problema fundamental que um agente em aprendizado por reforço enfrenta é escolher a melhor forma de alcançar seu objetivo, maximizando a recompensa, para isso o agente tem políticas de escolha de ações, que são mapeamentos entre estados e ações $M : S \to A$, onde a maior recompensa vai ser a da melhor política (Mohri et al., 2012).

### 2.3.2 Não-supervisionado

Utilizamos algoritmos de aprendizado não-supervisionado quando as amostras não possuem rótulos, estes algoritmos aprendem padrões significantes nos dados para detectar semelhança entre os dados e agrupá-los por algum padrão de semelhança que o algoritmo aprendeu.

Existem diversos algoritmos que podem servir para aprendizado não supervisionado citando alguns dos mais comuns temos: Aglomeração (K-means, (Murphy, 2012)), Autoencoders (Goodfellow et al., 2016), reduções de dimensionalidade (PCA, Kernel-PCA, (Murphy, 2012)).

### 2.3.3 Supervisionado

A classe de algoritmos que aprendem a mapear uma função entre os dados e os rótulos é conhecida como aprendizado supervisionado, pois aprendem sendo guiados (Murphy, 2012).

Como definido em Murphy (2012), "aprendizado supervisionado tem ênfase em conseguir generalizar uma função $y = F(x)$ onde $F$ é uma função desconhecida e o algoritmo aprende uma função $\hat{y} = \hat{F}(x)$ que seria a forma aproximada de $F$".

Problemas de aprendizado supervisionado podem ser divididos em duas grandes categorias, classificação e regressão. No problema de regressão, o rótulo é um valor(um número real), e para classificação o rótulo, é um número $k \in \mathbb{N}$ representando a classe da amostra. Uma outra forma muito utilizada para representação de rótulos é o *one hot encoding*, que consiste em um vetor $\mathbf{y} \in \mathbb{R}^n$ com a posição $y_k$ assumindo valor 1, indicando que a amostra pertence à classe $k$, e as outras posições com valor 0. Por exemplo, em uma base de dados $D$ que contém 5 classes, uma amostra $s \in D$ pertencente a terceira classe, tem rótulo $y = [0,0,1,0,0]^T$. Também existe a classificação múltiplas-classes em uma única amostra, onde o rótulo de uma amostra representa mais de uma classe, como o rótulo $y = [1,0,1,0,0]^T$, indicando que a amostra pertence às duas classes.

## 2.4 Validação

Na validação, a base de dados deve ser particionada para treinar e testar o modelo. Em geral, particionamos em tamanhos distintos, deixando mais amostras para o treino. Um fator com bastante influência no desempenho do modelo é o desbalanceamento da base de dados, onde uma classe possui um número muito superior de amostras de uma classe A do que da classe B. Estas bases desbalanceadas acarretam uma baixa performance na classificação de amostras da classe B.

Na fase de ajustes, onde o algoritmo ajusta os pesos do modelo, temos noção da performance pela minimização da função de perda aplicada à partição de treino. Porém, ainda é necessário avaliar a performance do modelo após termino da fase de ajustes, mensurando o quão bom o modelo se adequou para amostras que não participaram do seu ajuste, ou seja, amostras da

partição de teste.

### 2.4.1 Validação Cruzada

Segundo James et al. (2013), validação cruzada é uma forma geral de validação de modelos preditivos. Ela consiste das seguintes etapas: particionar em $k$ partes a base de dados, geralmente em tamanhos iguais, escolhendo uma das partes para teste e o restante para treino. O modelo é treinado outras $k$ vezes alternando-se a partição de teste.

Em geral, para aprendizado de máquina a escolha do $k$ tende a ser algo entre 5 e 10, porém, a escolha de um $k$ perfeito é uma tarefa complexa, a figura 2.1 mostra como seriam a escolha das partes, esse método é chamado de *k-fold cross validation* ou apenas *k-fold* (Mohri et al., 2012).

O intuito da validação cruzada é determinar que o algoritmo criou modelos em que se espera ter a mesma performance independente dos dados para treino (James et al., 2013), sendo assim, um bom modelo deve ter performance semelhante cada vez que for treinado, o resultado de um k-fold é a média de resultado entre todas as etapas $m = \frac{1}{k}\sum_{i=1}^{k}$ acurácia$(i)$.

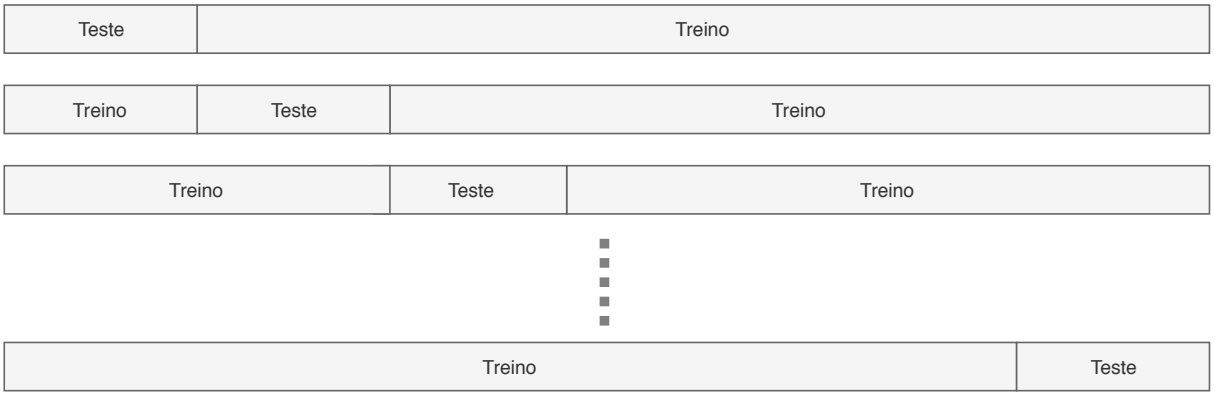


Figura 2.1: particionamento de uma base em k partes

### 2.4.2 Matriz de Confusão

É interessante visualizar os resultados da predição de um modelo, seja apenas a acurácia do modelo, ou a quantidade de acertos para cada classe. Uma das formas de visualizar os acertos de um modelo é a matriz de confusão, que consiste em uma matriz quadrada com as linhas representando as predições e as colunas quantidade de amostras disponíveis para uma classe.

Com auxílio da matriz de confusão podemos visualizar se o modelo avaliado teve um bom desempenho para cada classe individualmente. Na Tabela 2.4.3 segue um exemplo de uma matriz de confusão para um caso não binário.

### 2.4.3 Métricas de validação

Pela matriz de confusão podemos identificar falsos positivos e negativos ($FP$ e $FN$), e de verdadeiros positivos e negativos ($TP$ e $TN$), para um caso de classificação binária, escolhendo uma única classe podemos contar os casos de falsos positivos e negativos e verdadeiros positivos e negativos, e usar alguma métrica para avaliar o desempenho do nosso modelo para uma dada classe. Existem diversas métricas, que podem ser extraídas da matriz de confusão como a acurácia, cobertura e a precisão. Temos a acurácia definida como: $\frac{TP+TN}{TP+TN+FP+FN}$, precisão: $\frac{TP}{TP+FP}$ e a cobertura $\frac{TP}{TP+FN}$. Pela definição, precisão é: Proporção de positivos preditos que foram corretamente classificados, cobertura: Real proporção de casos corretamente preditos e acurácia: é o quanto o modelo acertou de um conjunto.

Tabela 2.1: Matriz de confusão para Iris-Flower Dataset usando SVM para classificação e divisão das amostras com 30% para teste, com acurácia de 97%

| **classe** | $c_1$ | $c_2$ | $c_3$ |
|---|---|---|---|
| $c_1$ | 15 | 0 | 0 |
| $c_2$ | 0 | 15 | 1 |
| $c_3$ | 0 | 0 | 14 |

# Capítulo 3

# Redes Neurais Artificiais

## 3.1 Introdução

Redes Neurais Artificiais ou simplesmente Redes Neurais são um modelo computacional representado por um grafo dirigido com pesos como na Figura 3.1, inspirado no sistema nervoso central (Haykin, 2009). As arestas e vértices do grafo representam, respectivamente, as sinapses e os neurônios.

Redes Neurais são capazes de adaptar-se ao meio que estão inseridas, aprendendo a realizar uma tarefa através de ajuste dos pesos de suas sinapses, de acordo com as amostras (Haykin, 2009). Os pesos são ajustados por um algoritmo de otimização, que comumente utiliza o algoritmo de retropropagação.

Uma rede neural pode adaptar-se para reconhecer diversas categorias de padrões, tais como imagens, áudios, vídeos e outros diversos tipos de dados multidimensionais. Classificação de imagens é uma tarefa que comumente é feita utilizando redes neurais, como na base de dados MNIST (Deng, 2012) composta por amostras de números e letras escritas à mão.

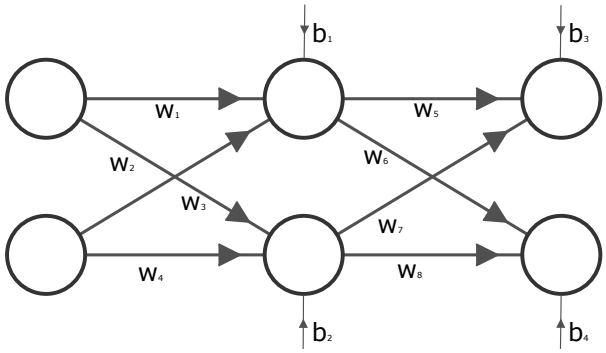


Figura 3.1: Rede neural simples

## 3.2 Neurônio e Perceptron

### 3.2.1 Modelo de McCulloch e Pitts

Um modelo de neurônio artificial comumente usado é o dos autores McCulloch and Pitts (1943), que é descrito como um neurônio que recebe $k$ sinais de entrada, $\mathbf{x} \in \mathbb{R}^k$, possui um vetor de pesos $\mathbf{w} \in \mathbb{R}^k$, um *bias* $b \in \mathbb{R}$, e uma função de ativação $\phi : \mathbb{R} \to \mathbb{R}$. A entrada para o neurônio é descrita como

$$\begin{aligned} net_{in} &: \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R} \\ net_{in}(\mathbf{x}, \mathbf{w}, b) &= \mathbf{x} \cdot \mathbf{w} + b, \end{aligned} \tag{3.1}$$

e a saída por uma função de ativação

$$\phi(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0, \end{cases} \tag{3.2}$$

que apenas indica se o neurônio ativou ou não. O modelo de neurônio de McCulloch e Pitts também é conhecido como limitante binário (Binary Threshold).

O modelo de neurônio descrito por McCulloch e Pitts pode ser facilmente generalizado apenas trocando a função de ativação utilizada, usando funções como as sigmóides. Isto permite ao neurônio ter uma funcionalidade diferente do modelo original que apenas transfere um valor binário, Tornando-se capaz de transferir um sinal mais significativo, além de continuar a ter o funcionamento semelhante ao modelo original fornecendo os valores binários caso necessário.

### 3.2.2 Perceptron

O Perceptron de Rosenblatt é um modelo de neurônio com pesos e bias ajustáveis, capaz de realizar classificações binárias de dados que sejam separáveis por um hiperplano (linearmente separáveis). O perceptron usa o modelo de neurônio de McCulloch e Pitts, sendo o perceptron o primeiro modelo de neurônio que descreve um algoritmo para ajustar os pesos de um neurônio.

Considere uma base de dados $D$ com dados estruturados e duas classes representadas respectivamente por 1 e $-1$. Um neurônio construído com modelo do perceptron classifica uma entrada $\mathbf{x} \in D$ usando parâmetros $\mathbf{w_o}$ e $b_o$, consistindo do vetor de peso e bias, descobertos na etapa de treino, de acordo com

$$p(x) = \begin{cases} 1, & (\mathbf{w_o} \cdot \mathbf{x}) + b_o > 0 \\ -1, & (\mathbf{w_o} \cdot \mathbf{x}) + b_o \leq 0. \end{cases}$$

O algoritmo de convergência de um perceptron encontra $\mathbf{w}$ e $b$ ,ajusta um vetor de pesos iniciais $\mathbf{w_1}$ pelo seguinte procedimento: caso uma entrada $\mathbf{x}_n$ seja classificada corretamente

o procedimento não ajusta $\mathbf{w_1}$ e $b_1$, caso não seja corretamente classificado um vetor $\mathbf{w_1}$ é atualizado da seguinte forma

$$\begin{aligned} \mathbf{w}_{n+1} &= \mathbf{w}_n - \alpha\mathbf{x}, \text{ com } x \text{ pertencendo a classe de rótulo } 1 \\ \mathbf{w}_{n+1} &= \mathbf{w}_n + \alpha\mathbf{x}, \text{ com } x \text{ pertencendo a classe de rótulo } -1 \end{aligned} \tag{3.3}$$

Note que o bias é ajustado junto ao vetor de pesos, considerando-se o bias como um dos elementos do vetor de pesos $\mathbf{w}_1$, da seguinte forma: $[w_1, w_2, ..., w_k, b_1]$, e a entrada para o neurônio $[x_1, x_2, ..., x_k, 1]$. A força de ajuste dos pesos para cada passo do algoritmo é controlada pela taxa de aprendizado $\alpha$.

## 3.3 Treino de uma Rede Neural Artificial

Uma rede neural artificial nada mais é do que uma composição de diversos neurônios artificiais que são capazes de gerar uma função mais complexa, capaz de resolver problemas de classificação altamente não lineares. Em uma rede neural sem retroalimentação seus neurônios não formam ciclos: os dados fluem através da rede em uma direção única fornecendo uma saída. Inicialmente com os pesos desajustados a saída difere da saída esperada. Para treinar a rede, os pesos e biases da rede devem ser ajustados, minimizando-se uma função de perda. Descobrimos os novos pesos propagando os erros em sentido contrário ao fluxo da rede neural (Rumelhart et al., 1988).

Inicialmente os pesos $\mathbf{w}$ e biases $\mathbf{b}$ da rede neural são escolhidos aleatoriamente e os algoritmos do método do gradiente e retropropagação acham os valores de $\mathbf{w}$ e $\mathbf{b}$ que melhor se encaixam. Para rede fornecer as saídas corretas.

Minimizando uma função de perda (equação 3.4), método do gradiente é capaz de encontrar os pesos ótimos $w$ e $b$, caminhando iterativamente na direção vetor gradiente $\nabla_w loss(y, \hat{y})$ em sentido oposto.

$$\begin{aligned} l &: \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} \\ loss(\hat{y}, y) &= \sum l(\hat{y}_i, y_i) \end{aligned} \tag{3.4}$$

Para construir o vetor gradiente $\nabla_w loss(\hat{y}, y)$ é necessário conhecer cada uma de suas derivadas parciais. O algoritmo de retropropagação realiza diferenciação automática para encontrar as derivadas parciais que constituem o vetor gradiente.

Considere a rede neural simples exibida na Figura 3.1. Essa rede neural possui 8 pesos e 4 biases, onde cada neurônio está sendo representado por uma função $net_i : \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R} \to \mathbb{R}$. Os pesos relacionados ao neurônio de índice $i$ representados como vetor $\mathbf{w}_{n,m}$ indicando que os pesos $w_n \in \mathbb{R}$ e $w_m \in \mathbb{R}$ são os elementos do vetor, uma entrada $\mathbf{x}$ fluirá através da rede e terá

saída da forma

$$\begin{aligned} \mathbf{v}_{1,2} &= \Big[net_1(\mathbf{x}, \mathbf{w}_{1,3}, b_1), net_2(\mathbf{x}, \mathbf{w}_{2,4}, b_2)\Big]^T \\ \hat{y} &= \Big[net_3(\mathbf{v}_{1,2}, \mathbf{w}_{5,7}, b_3), net_4(\mathbf{v}_{1,2}, \mathbf{w}_{6,8}, b_4)\Big]^T. \end{aligned} \tag{3.5}$$

Consequentemente, o gradiente construído pelo algoritmo de retropropagação será dado por

$$\nabla_{\mathbf{w}} loss(y, \hat{y}) = \sum_{i=0}^{n} \left[\frac{\partial loss(y_i - \hat{y}_i)}{\partial w_1}, \ldots, \frac{\partial loss(y_i - \hat{y}_i)}{\partial w_8}, \frac{\partial loss(y_i - \hat{y}_i)}{\partial b_1}, \ldots, \frac{\partial loss(y_i - \hat{y}_i)}{\partial b_4}\right]^T \tag{3.6}$$

e a atualização dos pesos feita pelo método do gradiente consiste da seguinte forma

$$\Big[w_1', \ldots, w_8', b_1', \ldots, b_4'\Big]^T = \Big[w_1, \ldots, w_8, b_1, \ldots, b_4\Big]^T - \alpha \nabla_{\mathbf{w}} loss(y, \hat{y}), \tag{3.7}$$

onde a constante $\alpha$ indica a taxa de aprendizado (passo de atualização) do método do gradiente. As condições de parada do algoritmo de retropropagação depende da função de perda convergir ou da conclusão de uma determinada quantidade de épocas.

### 3.3.1 Variações do algoritmo de descida do gradiente

O método do gradiente original utiliza todas as amostras disponíveis sequencialmente em cada uma de suas iterações, podendo não ter memória disponível para uma única iteração ou ficar preso em mínimos locais por causa da taxa de aprendizado fixa. Outros algoritmos foram criados para melhorá-lo.

Um das variações é o método do gradiente estocástico. Este algoritmo realiza uma aproximação estocástica. Diferente do algoritmo original, o método do gradiente estocástico tem as amostras de entrada embaralhadas para cada época e as atualizações do vetor gradiente são feitas para um a única amostra (Ruder, 2016).

O método do gradiente por mini-lotes é uma aprimoração em relação método do gradiente estocástico e por lotes. No método do gradiente por mini-lotes é necessário escolher uma quantidade de amostras (lote) para uma única atualização do vetor gradiente. O método do gradiente por mini-lotes embaralha as amostras e atualiza o vetor para a quantidade $k$ de amostras escolhida.

Um dos problemas para algoritmos de descida do gradiente são relacionados a escolha da taxa de aprendizado $\alpha$, como ilustra a Figura 3.2. Sendo essencial uma boa taxa de aprendizado, alguns algoritmos definem formas para a taxa de aprendizado ser atualizada junto das iterações do método do gradiente. Uma taxa de aprendizado dinâmica pode se ajustar melhor a superfícies não convexas, evitando pontos de mínimo locais e uma convergência lenta do algoritmo.

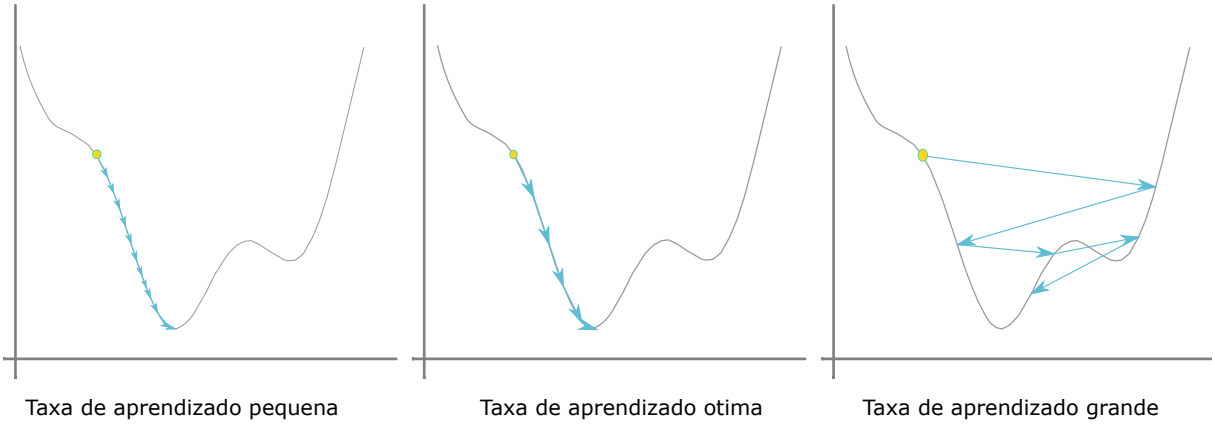


Figura 3.2: 1
Influência da taxa de aprendizado no Método do Gradiente.

Variações dos método do gradiente estocástico tais como o AdaGrad, RMSProp e Adam, tem sua taxa de aprendizado controlada pelo próprio algoritmo, variando a taxa de aprendizado de acordo com a função de otimização ou a importância dos pesos.

A **AdaGrad** (Adaptative Gradient) tem uma taxa de aprendizado para cada parâmetro e aprimora a taxa de aprendizado usando atualizações prévias (em relação ao tempo) através de uma matriz diagonal $G_{i,i}$ onde cada elemento da diagonal é descrito como

$$\sum_{t=1}^{n} g_{i,t}^2, \tag{3.8}$$

onde cada elemento $g_{i,t}$ da matriz é o gradiente em relação a um peso $w_i$ no tempo $t$.

A atualização dos pesos é realizada pela equação

$$w_{t+1,i} = w_{t,i} - \frac{\alpha}{\sqrt[2]{g_{i,i} + \varepsilon}}, \tag{3.9}$$

onde o valor $\varepsilon$ existe por motivos numéricos, evitando divisões por zero.

O método **RMSProp** foi desenvolvido com intuito de melhorar a forma como o gradiente é atualizado pelo método AdaGrad. No método AdaGrad, o gradiente pode atingir valores infinitesimais fornecendo pouca ou nenhuma atualização para algum peso, parando de aprender. O método RMSProp calcula o gradiente apenas em relação ao tempo anterior utilizando a esperança $E[g]$ do gradiente a atualização do gradiente é dada por

$$E[g^2]_{t+1} = 0.9E[g^2]_t + 0.1g_t^2. \tag{3.10}$$

e a atualização dos pesos respeitando a equação

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt[2]{E[g^2]_t + \varepsilon}}. \tag{3.11}$$

O **Adam** (Adaptive Moment Estimation) (Kingma and Ba, 2014) é uma variação do método RMSProp. O método Adam mantém uma taxa de atualização dos últimos gradientes e tem uso de momentos.

O método Adam armazena o decaimento exponencial da média dos gradientes usando $v_t$ e o decaimento da média quadrada $m_t$, respeitando as equações

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1-\beta_1) g_t \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2) g_t. \end{aligned} \tag{3.12}$$

O objetivo é evitar que a atualização do gradiente fique enviesada. O Adam calcula os momentos corrigindo os viés de acordo com as equações

$$\begin{aligned} \hat{m}_{t+1} &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_{t+1} &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{3.13}$$

Em seguida, a a atualização dos pesos ocorre de acordo com a equação

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt[2]{\hat{v}_{t-1}} + \varepsilon} \hat{m}_{t-1}. \tag{3.14}$$

## 3.4 Redes Neurais Convolucionais

Para dados não-estruturados é comum a utilização de extratores de características para a obtenção de dados estruturados. Um dos problemas tratados pela área de processamento de imagem é justamente a extração de características (LeCun et al., 1999). Muitos dos extratores de características para imagens consistem de filtros de convolução. Porém, nem sempre um determinado tipo de convolução pode extrair características relevantes para um probleam específico. Uma das formas de resolver isso e criar filtros mais adequados ao problema a ser tratado é a utilização de redes neurais convolucionais. Essas redes aprendem o conjunto de filtros que melhor se adaptam para as necessidades dos dados e do problema de classificação(LeCun et al., 1989).

### 3.4.1 Convolução

A convolução é uma operação de grande importância para área de processamento de sinais e imagens definida pela equação

$$f(t) \star k(t) = \int_{-\infty}^{-\infty} f(\tau) g(t-\tau) d\tau. \tag{3.15}$$

A operação de convolução consiste em relacionar o sinal de entrada $f$ com outro sinal $k$ e ter um sinal de saída $(f \star k)(\tau)$ que define o quão $f$ foi modificada por $k$. O sinal $k$ é normalmente

chamado de *kernel* (Smith, 1997; Gonzalez and Woods, 2006).

Contudo, geralmente é impossível ter a representação de um sinal de forma contínua no computador. Logo, um sinal é representado de forma discreta assumindo uma forma matricial. Por exemplo, uma função $F(x,y) = \sqrt[2]{r^2 - x^2 - y^2}$ é representada através de uma discretização, como na matriz

$$\begin{bmatrix} F(-r+0,r-0) & F(-r+1,r-0) & \dots & F(-r+2r,r-0) \\ F(-r+0,r-1) & F(-r+1,r-1) & \dots & F(-r+2r,r-1) \\ \vdots & \vdots & \vdots & \vdots \\ F(-r+0,r-2r) & F(-r+1,r-2r) & \dots & F(-r-2r,r-2r) \end{bmatrix}, \tag{3.16}$$

Quando esta matriz representa uma imagem, podemos visualizá-la como na Figura 3.4.1. Desse modo, é de extrema relevância trabalhr com convoluções discretas. No caso unidimensional, uma convolução é definida de acordo com a equação

$$f(x) \star k(x) = \sum_{m=-\infty}^{\infty} f(x-m)k(m). \tag{3.17}$$

Já uma convolução bidimensional respeita a equação

$$f(x,y) \star k(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\inf} f(x-m,y-n)k(m,n). \tag{3.18}$$

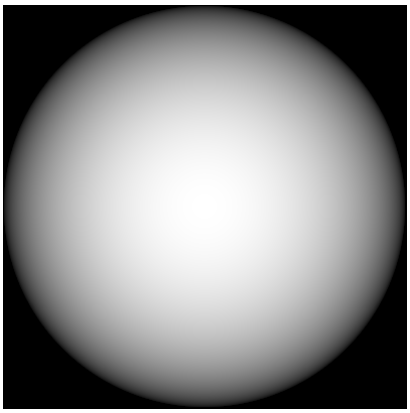

Figura 3.3: Image gerada a partir da função $F(x,y) = \sqrt[2]{r^2 - x^2 - y^2}$ com resolução de $200 \times 200$ *pixels*.

A convolução é uma operação essencial em imagens, usada para extrair características como contornos ou formas, filtrar ruídos, realçar, borrar e extrair características de imagens.

### 3.4.2 Redes Neurais Convolucionais

A arquitetura de rede neural convolucional em geral é organizada como na Figura 3.4: sequencialmente, uma imagem é dada de entrada para camadas de convolução e acumulação, que aplicam filtros, obtendo mapas de características que são dados de entradas para outras camadas convolucionais. Em seguida, camadas densas são responsáveis por produzir a classificação da imagem.

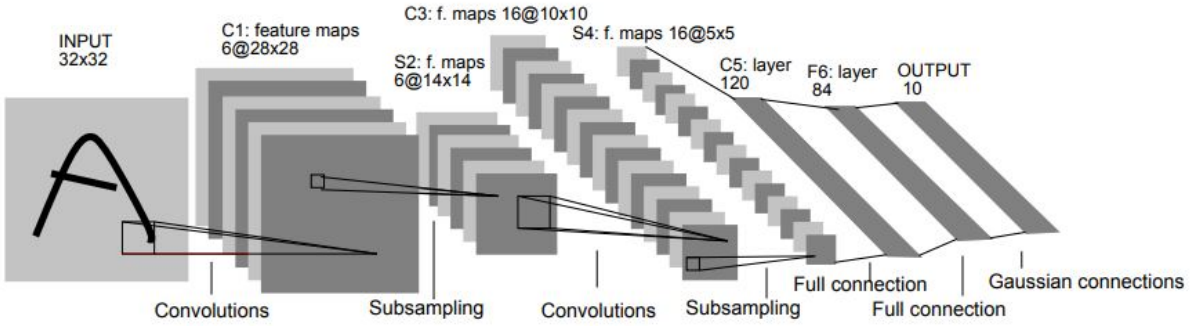


Figura 3.4: Arquitetura da LeNet-5 uma Rede Neural Convolucional (LeCun et al., 1999)

Uma camada de rede neural convolucional tem intenção de identificar padrões sem alterar a topologia dos dados e evitar sofrer da maldição da dimensionalidade. Para achar padrões em imagens podemos utilizar filtros convolucionais que destaquem as características necessárias. O ato de criar um novo tipo de filtro convolucional que seja útil para resolver determinado problema pode demandar muito esforço e tempo. Uma forma de encontrar filtros importantes é utilizar redes neurais convolucionais que aprendem de forma automática os filtros necessários, através de um processo de otimização.

Em uma camada de rede neural convolucional temos um conjunto de neurônios que compartilham pesos entre si atrelados a uma convolução. Cada conjunto de neurônios de uma convolução produz um ou mais mapas de características (*feature map*) dependendo apenas da quantidade sinais de entrada recebidos em uma única amostra. Por exemplo, em uma rede convolucional com 2 camadas convolucionais aplicando 2 filtros por camada, inicialmente uma imagem de entrada é filtrada pelos 2 filtros da primeira camada, produzindo dois mapas de características, e a segunda camada convolucional produz dois mapas de características para cada mapa obtido da primeira camada.

Os neurônios de uma camada de rede neural convolucional são organizados de forma a cada um ser um elemento na construção do mapa de características, onde cada neurônio na camada convolucional recebe uma entrada atrelada a uma região da imagem anterior delimitada pelo tamanho dos filtros de convolução dessa camada. Por exemplo, um neurônio conectado a uma

região 3*x*3 teria ligações com os seguintes *pixels* da imagem de entrada:

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & p_{x,y} & p_{x+1,y} & p_{x+2,y} & \cdots \\ \cdots & p_{x,y+1} & p_{x+1,y+1} & p_{x+2,y+1} & \cdots \\ \cdots & p_{x,y+2} & p_{x+1,y+2} & p_{x+2,y+2} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{3.19}$$

, onde cada neurônio produz um único pixel para o mapa de características.

Frequentemente, camadas convolucionais são seguidas de camadas de Acumulação (Pooling). Estas camadas têm a função de acumular valor de uma vizinhança de *pixels* e produzir uma imagem menor. É interessante destacar que esse tipo de camada não possui pesos treináveis. Em outras palavras, uma função de *pooling* substitui uma região de *pixels* com um resumo estatístico de saídas próximas. Por exemplo, uma operação de *max pooling* aplicada em uma região de 2*x*2 *pixels* produz um único *pixel* com o valor mais alto da região. Outros tipos de operação de *pooling* existem, como o *average pooling*, onde o *pixel* de saída tem como valor da média de todos os *pixels* da região.

## 3.5 Redes Neurais Recorrentes

Redes neurais recorrentes (RNN) são uma classe de redes neurais apropriadas para tratar dados sequenciais, como vídeos, som e texto, e compartilha estados entre cada passo. Isto significa que as RNN possuem uma forma de memória interna, capaz de levar em conta o que aconteceu no passado para uma decisão futura.

Camadas recorrentes são teoricamente um grafo cíclico como na Figura 3.5. Entretanto, considerando uma quantidade de passos finita, podemos ter uma representação desdobrada como na Figura 3.5.

Existem diversos tipos de RNN, sendo a forma mais básica aquela que compartilha sequencialmente apenas o estado $\mathbf{h}_{t-1}$ com o próximo passo $t$. Para as rede neurais recorrentes seus pesos e *biases* são compartilhados com todos os passos.

Formas mais sofisticadas de redes neurais recorrentes possuem células recorrentes mais complexas, como a Long Short-Term Memory (LSTM) e modelos que possuem uma forma de compartilhamento de estado mais complexo como a Tree-LSTM (Figura 3.6,(Tai et al., 2015)).

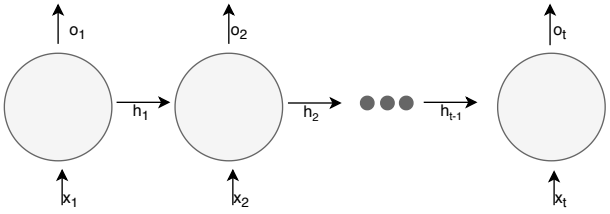


Figura 3.5: Diagrama de uma Rede neural recorrente desdobrada.

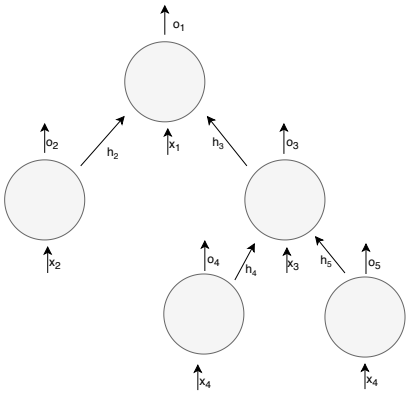


Figura 3.6: Diagrama de uma Tree-LSTM.

Para cada passo de uma camada recorrente existe uma saída atrelada a aquele passo, sendo essa saída construída através do estado atual. Como é construída uma saída para cada passo, temos duas arquiteturas básica de RNN, uma que concatena as saídas dos passos para classificar as sequencias utilizando de camadas densas padrão, e, a outra sendo uma arquitetura que utiliza as saídas de cada passo de forma a predizer a próxima sequencia, um exemplo clássico são os *chatbot*.

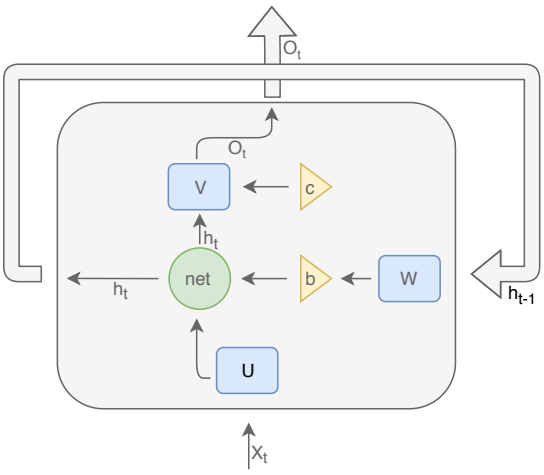


Figura 3.7: Modelo de célula de Rede neural recorrente padrão, com quadrados azuis sendo os pesos, triângulos amarelos os *biases* e o círculo verde a função de ativação

Uma camada de rede neural recorrente padrão segue como na Figura 3.5, possuindo três conjuntos de pesos e dois *biases* para cada passo, o peso **U** são os pesos atrelados as entradas para a ativação, com **U** podendo decidir a força que o passo atual tem para toda célula. Os pesos e bias **W** e **b**, referentes ao estado anterior se remetem a força que o estado anterior possui para todo passo atual e com o estado atual sendo construído a partir da entrada e do estado do passo anterior

$$\mathbf{h}_t = net(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b}) \tag{3.20}$$

e com os pesos e *biases* relativos a saída **V** e **c**, sendo os pesos que controlam a força da decisão para classificação ou o elemento desejado para sequencia que rede fornece de saída.

O treino de uma rede neural recorrente é feito pelo algoritmo de retropropagação no tempo, que é parecido com o algoritmo de retropropagação. Porém, este leva em consideração a forma desdobrada 3.5 da rede neural recorrente. Para mais informações, O algoritmo de retropropagação no tempo calcula o vetor gradiente para cada etapa de tempo que da camada de rede neural recorrente $\nabla_{o_t} Loss$ partindo das últimas etapas até as primeiras etapas de tempo. O gradiente dos estados são atualizados por uma matriz jacobiana relacionada a ativação da camada no tempo atual

$$\begin{aligned}
\nabla_{\mathbf{W}} Loss &= \sum_t diag(\frac{\partial Loss}{\partial h_t})(\nabla_{h_t} Loss) h_{(t-1)}^T \\
\nabla_{\mathbf{U}} Loss &= \sum_t diag(\frac{\partial Loss}{\partial h_t})(\nabla_{h_t} Loss) x_t^T \\
\nabla_{\mathbf{V}} Loss &= \sum_t (\nabla_{o_t} Loss) h_t^T \\
\nabla_{\mathbf{c}} Loss &= \sum_t (\nabla_{o_t} Loss) \\
\nabla_{\mathbf{b}} Loss &= \sum_t diag(\frac{\partial Loss}{\partial h_t}) \nabla_{h_t} Loss
\end{aligned} \tag{3.21}$$

os pesos e *biases* são atualizados após obterem a rede em sua forma desdobrada, como os pesos e *biases* são compartilhados para cada passo, na etapa de atualização são adicionados novos pesos e *biases* $\mathbf{W}_t, \mathbf{U}_t, \mathbf{V}_t, \mathbf{b}_t e \mathbf{c}_t$ e são inicializados com os mesmos valores dos pesos e *biases* atuais na rede, com isso o algoritmo de retropropagação através do tempo pode atualizar os novos pesos e *biases* levando em consideração cada etapa da RNN, (Werbos et al., 1990).

### 3.5.1 *Long Short-Term Memory*

Long Short-Term Memory (LSTM) é uma variedade de rede neural recorrente que adiciona conceito de portões nas camadas recorrente. Redes neurais recorrentes padrão possuem um problema relacionado ao gradiente sumir (*vanish gradient*) ou estourar na etapa de ajustes, a LSTM aborda esse problema adicionando portões e ligando esses portões com as entradas,

saídas e os estados da rede, fazendo o gradiente ter um fluxo melhor em toda camada recorrente. Desse modo, a LSTM é uma variedade de uma *Gated RNN*, onde a célula como um todo pode ser associada a uma memória, e os portões sendo unidades que liberam o acesso para leitura ou escrita da informação entre os passos, fortalecendo as dependências de longo prazo em outros passos (Hochreiter and Schmidhuber, 1997).

LSTM continua possuindo as características de uma RNN padrão de compartilhar informações com outros passos. Porém, ela tem como vantagem a capacidade de aprender quando deve esquecer e incorporar informação ao estado. Mais especificamente, a LSTM possibilita a o uso de memória (estados) através de portões, aprendendo quando fechar ou abrir o portão, e assim reduzindo ou incrementando a importância do sinal que chega ao portão. Os mecanismos de portões na LSTM estão ligados à entrada $x_t$, à saída $o_t$, e à unidade de esquecimento $f_t$.

Uma célula de LSTM é construída como na Figura 3.8, onde cada um dos portões de uma célula $i$ possui a mesma função de ativação $\sigma$ com imagem $[0,1]$. A unidade de esquecimento $f_{i,t}$ da célula $i$ no passo $t$, com pesos $\mathbf{W}^f$, $\mathbf{U}^f$ e bias $\mathbf{b}^f$, usa a saída do passo anterior $\mathbf{h}_{t-1}$ para determinar quanto deve ser esquecido do estado anterior $c_{t-1}$, de acordo com a equação

$$f_{i,t} = \sigma(b_i^f + \sum_n^i (W_{i,n}^f) h_{(t-1)_n} + \sum_{n=1}^i (U_{i,n}^f) x_{t_n}). \tag{3.22}$$

Analogamente, o portão de entrada $g_{i,t}$ tem pesos $W^g$ e $U^g$ e bias $b^g$ e determina quanto

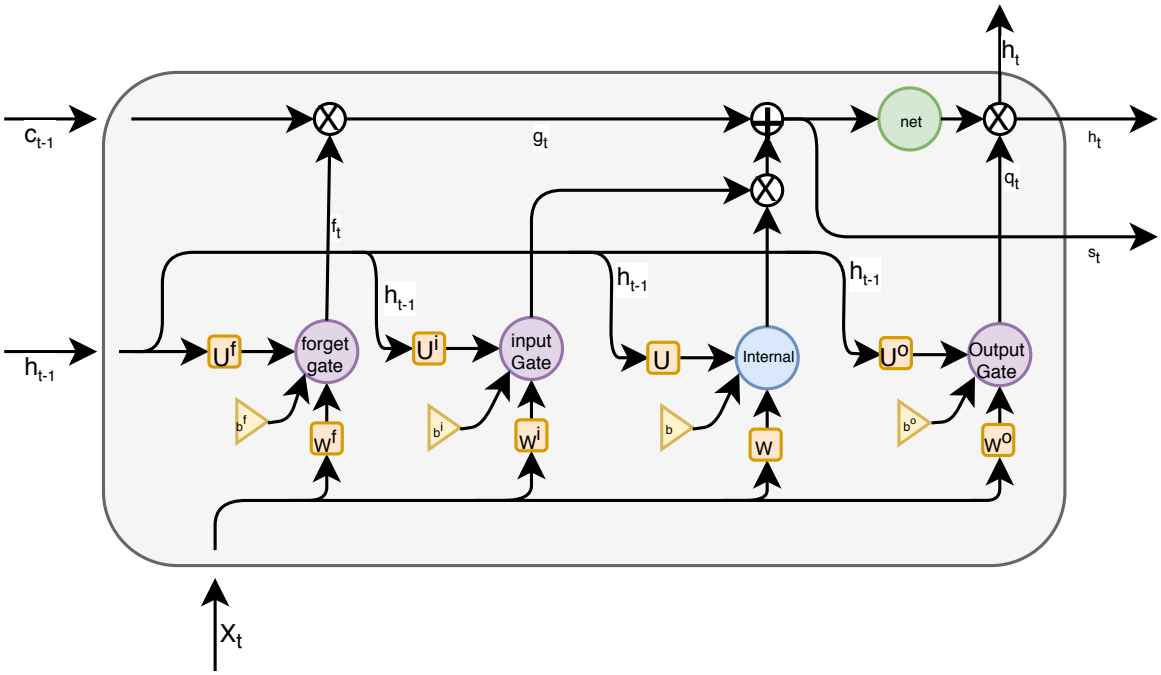


Figura 3.8: Modelo de célula de LSTM com os quadrados amarelos sendo os pesos, triângulos amarelos os *biases* os círculos roxos os portões, círculo azul o estado interno, e o círculo verde a ativação.

deve ser incorporado de nova informação no passo $t$, de acordo com a equação

$$g_{i,t} = \sigma(b_i^g + \sum_n^i (W_{i,n}^g) h_{(t-1)_n} + \sum_n^i (U_{i,n}^g) x_{t_n}). \tag{3.23}$$

Estes dois portões são combinados para atualizar o estado $c_{i,t}$, respeitando a equação

$$c_{i,t} = f_{i,t} c_{i,t-1} + g_{i,t} \sigma(b_i + \sum_n^i (W_{i,n}) h_{(t-1)_n} + \sum_{n=1}^i (U_{i,n}) x_{t_n}), \tag{3.24}$$

onde **U** e **W** são pesos e **b** um bias. O portão de saída controla a saída da unidade LSTM, e tem valor dado por

$$q_{i,t} = \sigma(b_i^q + \sum_n^i (W_{i,n}^q) h_{(t-1)_n} + \sum_n^i (U_{i,n}^q) x_{t_n}), \tag{3.25}$$

onde $W^q$ e $U^q$ são pesos e $b^q$ um bias. Finalmente, a saída da célula LSTM tem ativação *net* e é atrelada ao portão da saída $q_t$ e ao estado $c_{i,t}$, sendo calculada como

$$o_{t_i} = net(c_{t_i}) q_{i_t} \tag{3.26}$$

# Capítulo 4

# Metodologia

Nesse capítulo descreveremos as etapas da metodologia adotada para avaliar arquiteturas de redes neurais profundas para realizar o reconhecimento de sinais dinâmicos de Libras. Em cada instante de tempo, durante o treino ou classificação em tempo real, o usuário realiza uma pose da mão, que é capturada pelo sensor de profundidade, e a imagem de profundidade resultante é enviada para segmentação (Seção 4.1). Esta etapa tem como objetivo extrair a mão do fundo e reescalar a imagem para uma imagem menor de $50x50$ *pixels* (dimensões de entrada da rede neural). A imagem segmentada e reescalada é então enviada para treinamento ou classificação da rede neural, dependendo da fase. Avaliamos duas classes de arquiteturas de redes neurais, descritas nas Seções 4.2 e 4.3, e diversas combinações de seus hiper-parâmetros, através de procedimentos de validação cruzada, descritos na Seção 4.4.

## 4.1 Segmentação

Os vídeos utilizados no projeto são vídeos de profundidade, gravados a 30 quadros por segundo utilizando a câmera de profundidade da Intel a RealSense F200. Os sinais foram adquiridos por um sensor de profundidade capaz de capturar imagens de dimensões $640x480$ *pixels* com único canal de 8-bit (0 a 256) com a informação da profundidade da cena. Desse modo, consideramos uma imagem de profundidade como uma função

$$s_i : \mathbb{U} \subset \mathbb{R}^2 \rightarrow \mathbb{R}. \tag{4.1}$$

As imagens foram filtradas para remoção do fundo, considerando que a mão é o objeto mais próximo da câmera. Desse modo, a imagem $s_i$ é filtrada em relação ao ponto mais próximo do sensor ($p$), mais um limiar $d$, onde o valor do limiar foi escolhido de forma empírica, onde o valor $d = 20$ cm mostrou ser uma boa escolha durante os experimentos.

$$\tilde{s_i}[x,y] = \begin{cases} f_t[x,y], & f_t[x,y] \leq p+d \\ 0, & f_t[x,y] > p+d\,. \end{cases} \tag{4.2}$$

A escolha de não recortar a mão das imagens para incluir apenas ao sinal é feita para preservar translações e evitar redimensionamentos de algum quadro de um video.

Um vídeo de sinais dinâmicos é representado por uma sequência $s = (s_1, s_2, s_3, ..., s_n)$ com cada $s_i$ sendo uma imagem correspondente a um quadro do vídeo. Todos os vídeos foram gravados a 30 quadros por segundo, com o sinal mais longo possuindo 1.3 segundos de duração ou 40 quadros. A gravação dos sinais foi realizada variando a duração e posição para a câmera.

Para entrada das redes neurais, os vídeos precisam ser de uma sequência de tamanho fixo. Por isso escolhemos fornecer uma sequência com tamanho do maior vídeo $T = 40$ quadros. As sequências menores que 40 quadros, são completadas com quadros de imagens de zeros (imagens onde todos os pixels tem profundidade zero).

Todos os vídeos foram re-escalados para resolução de $50x50$ para entrada das redes. A resolução menor foi escolhida empiricamente visto que, com uma resolução de $50x50$, ainda seria possível um humano identificar os sinais, e com isso esperamos que a máquina também fosse capaz.

## 4.2 CNN 3D

Redes neurais convolucionais 3D (CNN 3D) foram desenvolvidas para aplicações de reconhecimento de atividades de humanos em vídeos de segurança (Ji et al., 2013), em utilizações futuras das CNN 3D foi observado que elas são boas para aprender características espaço-temporais melhores que redes neurais convolucionais 2D que perdem as informações relacionadas ao tempo (Tran et al., 2015).

Uma camada de CNN 3D, tem funcionamento semelhante a camada de rede neural convolucional 2D, com a diferença que os *kernels* (filtros) vão ter três dimensões, e a camada de acumulação (*pooling*) também.

Uma camada de CNN 3D tem seus neurônios organizados de forma que as sinapses de um único neurônio são conectadas a uma região com um formato de bloco tridimensional ao vídeo de entrada. Cada conjunto de neurônios atrelados a uma convolução tem como objetivo aprender uma determinada característica espaço-temporal. Um conjunto de neurônios para uma única convolução compartilham os mesmos pesos $\mathbf{W}$ e bias $b$ e possuem a mesma função de ativação *net* para toda a camada convolucional.Para uma camada de rede neural convolucional 3D, que faz $k$convoluções e recebe de entrada $n$ mapas de características, tem como saída $k * n$ mapas de características.

A nossa arquitetura de rede neural convolucional é composta de um conjunto de $h$ blocos com camadas convolucionais 3D e acumulação máxima 3D com ativação *ReLU*, em sequência

temos uma camada de achatamento com pesos não treináveis, convertendo todos os mapas de características da saída do bloco anterior achatando-os em um vetor e fornecendo ele para um bloco com $g$ camadas Densas de ativação *ReLU*, e a saída desse bloco sendo classificada por uma única camada densa com ativação *softmax*.

## 4.3 LRCN

A arquitetura de rede neural que combina redes neurais recorrentes do tipo LSTM junto com redes neurais convolucionais 2D (LRCN) foi proposta por Donahue et al. (2015). Atualmente ela tem aplicações em reconhecimento de atividades, descrição de vídeo e legenda de imagens. Esta arquitetura combina o poder da LSTM para aprender sequências que possuam dependências temporais de longo termo, com o poder das redes neurais convolucionais bidimensionais de aprender características em imagens. A junção das duas arquiteturas promove uma forma de reconhecer característica espaço-temporais dentro das sequências de imagens (vídeo).

Diferentemente da CNN 3D, a LRCN recebe para cada quadro do vídeo a imagem correspondente ao quadros. Os quadros são processados independentemente por $C$ blocos compostos pelas camadas convolucionais e acumulação máxima, onde a saída do último bloco passa por uma camada de achatamento transformando-se num vetor para entrada nas camadas LSTM. Esse processo é feito para cada quadro da sequência, onde as camadas LSTM levam em conta o estado retornado pela camada LSTM correspondente do passo anterior da sequência. A saídas da LSTM para cada quadro são fornecidas para uma camada densa (com ativação *softmax*) para serem classificadas.

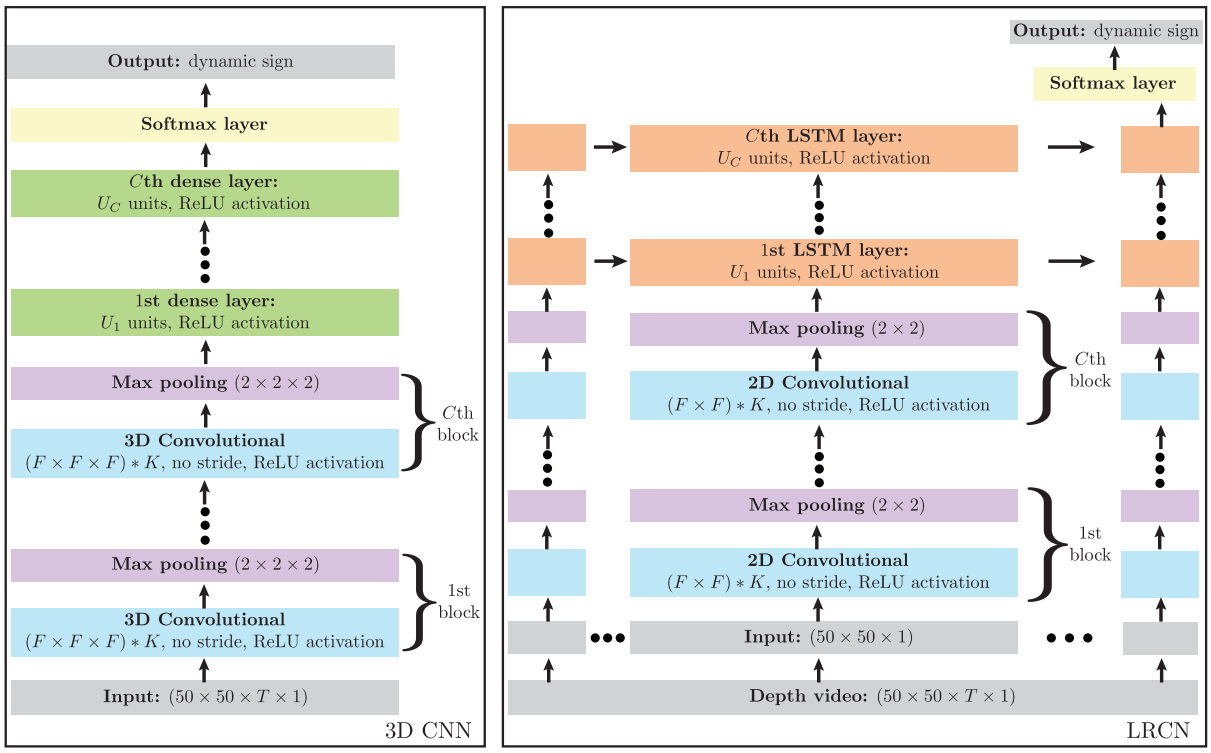


Figura 4.1: Arquitetura da CNN 3D (esquerda) e arquitetura da rede LRCN (direita).

## 4.4 Otimização de hiper-parâmetros

Cada camada de rede neural possui uma certa quantidade de parâmetros, tais como quantidade de neurônios, função de ativação, quantidade de filtros e tamanho dos filtros. Aplicamos algoritmos de busca para achar a combinação de hiper-parâmetros que melhor se aplica ao problema proposto.

Para as arquiteturas da da classe das CNN 3D, o espaço de busca é composto por todas as combinações entre os hiperparâmetros abaixo:

- Número de filtros $K_{CNN3D} = \{16, 32, 64\}$
- Tamanho dos filtros $F_{CNN3D} = \{3, 5\}$
- Quantidade de camadas convolucionais $C_{CNN3D} = \{1, 2\}$
- Quantidade de camadas densas $D_{CNN3D} = \{1, 2, 3\}$
- Número de neurônios nas camadas densas $N_{CNN3D} = \{100, 200, 300\}$.

Para as arquiteturas da classe LRCN, consideramos um espaço de busca parecido:

- Número de filtros $K_{LRCN} = \{16, 32, 64\}$
- Tamanho dos filtros $F_{LRCN} = \{3, 5\}$

- Quantidade de camadas convolucionais $C_{LRCN} = \{1,2,3\}$
- Quantidade de camadas LSTM $D_{LRCN} = \{1,2,3\}$
- Número de neurônios nas camadas LSTM $N_{LRCN} = \{100,200,300\}$.

Para ambas as classes de arquiteturas, construímos os blocos de LSTM ou densas a partir de uma combinação feita com os parâmetros $N_{CNN3D}$ para CNN3D e $N_{LRCN}$ LRCN. O resultado da combinação desses parâmetros fornece um conjunto de vetores onde cada elemento contém a quantidade de neurônios de uma dessas camadas. Caso não possua uma camada o vetor terá 0 na posição. Por exemplo, se $N_{LRCN} = \{200,300,0\}$, teremos 2 camadas consecutivas de LSTM com 200 e 300 neurônios, respectivamente. O mesmo acontece para as camadas de CNN 3D convolucionais e para as convoluções 2D na LRCN, com os parâmetros dessa vez utilizando $K_{CNN3D}$ e $K_{LRCN}$ para os filtros e $F_{CNN3D}$ e $F_{LRCN}$ para combinatória do tamanho dos filtros. Após cada camada convolucional, em ambas as classes de arquiteturas, são incluídas camadas de acumulação máxima com tamanho do bloco sendo (2x2) para LRCN e (2x2x2) para CNN 3D. Para todas as camadas convolucionais, a função de ativação escolhida foi a $ReLU(x) = max(0,x)$. Para as camadas LSTM, as ativações são a tangente hiperbólica, dada por

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

Para as camadas densas da CNN 3D a ativação escolhida é *ReLU* e em ambas arquiteturas a ativação da camada densa para classificação é a função *softmax*, que retorna uma distribuição das probabilidades do sinal pertencer a uma determinada classe de sinais, e é dada por

$$\begin{aligned} softmax &: \mathbb{R}^n \to [0,1]^n \\ softmax(x) &= [s(x,1), s(x,2), \ldots, s(x,n)]^T, \end{aligned} \tag{4.3}$$

onde

$$\begin{aligned} s &: \mathbb{R}^n \times \mathbb{R} \to [0,1] \\ s(x,i) &= \frac{e^{x_i}}{\sum^n e^{x_n}}. \end{aligned} \tag{4.4}$$

Como a quantidade de combinações entre estes hiper-parâmetros não é tão alta, resolvemos aplicar a estratégia de busca força-bruta, ou seja, avaliamos todas as combinações possíveis entre estes valores. Para cada combinação de hiper-parâmetros, foi realizada uma validação cruzada, separando a base de dados aleatoriamente com 30% das amostras para teste e os 70% restante para treino. A validação da média da acurácia após 10 repetições é o valor considerado para representar a robustez do modelo.

O algoritmo de otimização utilizado para treino das arquiteturas foi o Adam (Kingma and Ba, 2014), com a implementação da biblioteca Keras (Chollet et al., 2015), utilizando uma taxa

de aprendizado inicial de $0,001$.

# Capítulo 5

# Experimentos

## 5.1 Base de dados

Para nossos experimentos construímos uma base de dados de sinais dinâmicos de Libras [1]. Escolhemos dez classes de sinais, com metade para sinais de uma única mão e outra metade para duas mãos, Sendo as seguintes classes de sinais escolhidas: letra "H", letra "J", as palavras "dia", "noite", "entrar", "entrar", "tudo", "novamente", "início"e "curso".

A escolha dos sinais foi realizada com ajuda de especialista em Libras, com critério de escolher sinais que tivessem alguma semelhança por pares. Os sinais "noite"e "entrar", por exemplo, possuem semelhança na posição da mão não-dominante e movimentação da mão dominante. Para cada classe de sinal foram coletados 300 amostras de vídeos de profundidade utilizando a câmera RealSense.

[1] A base é pública e pode ser acessada em: http://im.ufal.br/professor/thales/libras/database.rar

Figura 5.1: Sinais dinâmicos de Libras da base de dados. Respectivamente a parte superior são os primeiros quadros dos sinais, e a parte inferior o último quadro dos sinais. Da esquerda para direita os sinais são, letra “H”, “dia”, “novamente”, “seu”, letra “J”, “noite”, “entrar”, “tudo”, “início”, “curso”.

Tabela 5.1: Acurácia dos melhores modelos avaliados de CNNs 3D e LRCNs.

| class | $C_*$ | $F_*$ | $K_*$ | $L_*$ | $U_*$ | mean accuracy | weights |
|---|---|---|---|---|---|---|---|
| LRCN | 2 | (3, 3, 0) | 64 | 2 | (100, 100) | 99.8% | 3,256,978 |
| CNN 3D | 2 | (3, 3, 0) | 64 | 3 | (300, 200, 100) | 99.8% | 18,779,658 |
| CNN 3D | 2 | (5, 3, 0) | 32 | 3 | (300, 200, 100) | 99.6% | 5,656,874 |
| LRCN | 1 | (3, 0, 0) | 64 | 2 | (100, 100, 0) | 99.7% | 14,868,050 |
| LRCN | 1 | (5, 0, 0) | 64 | 2 | (200, 200, 0) | 99.7% | 27,570,074 |
| LRCN | 1 | (3, 0, 0) | 64 | 3 | (300, 200, 200) | 99.7% | 45,322,250 |
| LRCN | 1 | (5, 0, 0) | 64 | 3 | (300, 200, 200) | 99.6% | 41,713,674 |
| CNN 3D | 2 | (3, 3, 0) | 32 | 2 | (200, 300, 0) | 99.5% | 6,287,286 |
| CNN 3D | 2 | (5, 3, 0) | 64 | 2 | (300, 100, 0) | 99.5% | 11,437,938 |
| CNN 3D | 2 | (3, 3, 0) | 16 | 3 | (100, 100, 200) | 99.5% | 6,287,286 |

## 5.2 Acurácia das Arquiteturas

Nossos experimentos de otimização de hiper-parâmetros revelaram que muitas arquiteturas apresentaram excelente robustez, com as melhores arquiteturas superando o patamar de 99% de acurácia. A Tabela 5.1 exibe as arquiteturas com melhor acurácia.

Para analisar a importância da otimização de hiper-parâmetros, construímos um histograma para medir a frequência na qual as arquiteturas experimentadas apresentaram acurácia em determinados intervalos de acurácia. Verificamos que muitos modelos resultaram em acurácia acima dos 90% para ambas arquiteturas, como pode ser visto pelos histogramas na Figura 5.2. Como muitas delas apresentam resultados acima dos 99%, escolher uma arquitetura que possua menos pesos treináveis torna-se um segundo critério relevante para selecionar o melhor modelo. Neste caso, a melhor arquitetura de LRCNs exibida na Tabela 5.1 seria a mais recomendada.

Nós concluímos que ambas as classes de arquiteturas (LRCN e CNN3D) são robustas o suficiente para serem aplicadas para reconhecimento de sinais dinâmicos de Libras. Entretanto, em contraste da CNN 3D em relação a LRCN tem poder para fornecer uma saída de probabilidades parciais para cada passo no tempo, isso tornando a LRCN mais útil para algumas situações.

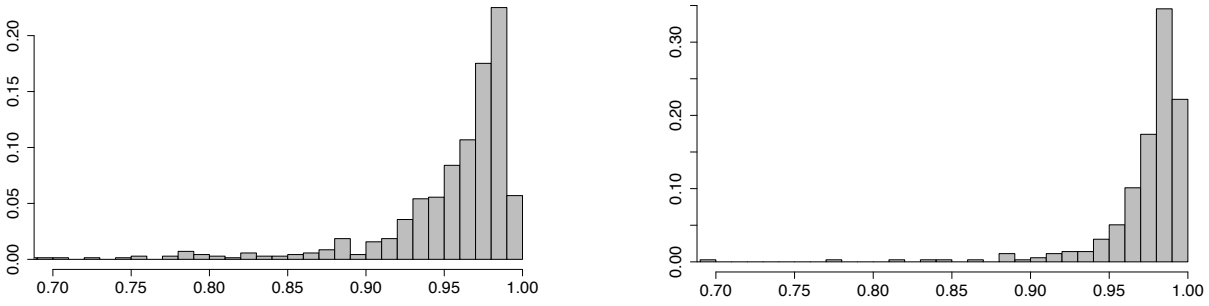


Figura 5.2: Histogramas de acurácias para os modelos profundos avaliados. O eixo horizontal representa as acurácias, e o eixo vertical a frequência das acurácias. A imagem a esquerda indicando histograma para as CNNs 3D e a imagem a direita para LRCN.

## 5.3 Comparação com SVM

Realizamos experimentos com um classificador SVM*baseline* para comparar com as arquiteturas avaliadas, com relação à acurácia alcançada pelas arquiteturas utilizadas com um classificador padrão. Para o SVM, a entrada foi fornecida como o vídeo inteiro de forma achatada, com cada quadro achatado e concatenado com o próximo quadro.

Como o SVM também possui hiper-parâmetros relevantes, em nosso experimento consideramos os hiper-parâmetros de regularização C e o raio $\lambda$. Nosso espaço de busca para otimização de hiper-parâmetros foi construído a partir de todas as combinações entre os seguintes valores de hiper-parâmetros:

- $C_{svm} = \{0.1, 0.3, 0.5, 0.7, 0, 9, 1\}$
- $\gamma_{svm} = \{0.1, 0.5, 1, 10, 100, 1000\}$

Foram realizados procedimentos de validação cruzada semelhantes aos adotados para otimização de hiper-parâmetros das arquiteturas de redes neurais. Repartimos a base de dados em 30% para teste e o restante para treino. Obtivemos os melhores hiper-parâmetros para o SVM como $C = 1$ e $\gamma = 0.1$, alcançando apenas 63% de acurácia, o qual é bastante inferior aos mais de 99% obtidos pelas melhores redes neurais profundas.

# Capítulo 6

# Considerações Finais

Nesse trabalho foi proposto um estudo de arquiteturas de redes neurais para reconhecimento de sinais dinâmicos de Libras. Utilizando arquiteturas que tivessem capacidade de trabalhar com dados temporais, foram escolhidas arquiteturas inspiradas nas redes neurais recorrentes e redes convolucionais tridimensionais. Para esse trabalho coletamos sinais dinâmicos de Libras através de uma câmera de profundidade, cada video coletado foi coletado a 30 quadros por segundo. Com os sinais coletados foi construída uma base de dados pública, que possui 10 classes de sinais, contendo 300 amostras para cada classe, totalizando 3000 amostras coletadas, com o video de maior duração possuindo 40 quadros.

Com as arquiteturas das redes neurais escolhidas, foram realizados um conjunto de experimentos com intuito da otimização dos hiper-parâmetros para as arquiteturas escolhidas, LRCN e CNN 3D. Consistindo uma busca em *grid* no espaço dos hiper-parâmetros, o experimento revelou quais hiper-parâmetros conseguiam melhor se ajustarem para ambas as arquiteturas, provando que são arquiteturas robustas.

Os resultados foram ótimos, conseguimos alcançar uma taxa de 99% de acurácia com ambas arquiteturas. Com um realce para LRCN que pode ter uma classificação previa para cada passo de tempo não sendo necessário fornecer todo video e a LRCN sendo a arquitetura que apresentou menor numero de parâmetros treinvaveis.

## 6.1 Trabalhos Futuros

Temos a intenção de cada vez utilizar sinais mais complexos que possam incluir expressões corporais e faciais além das configurações das mãos.

Onde a nova arquitetura descoberta seja uma tenha viabilidade a para tradução simultânea, seja tratável fornecer transições entre execução de sinais e a própria rede conseguir as classes sendo executadas.

Além que para um trabalho com sinais mais complexos também será necessário uma base de dados mais robusta, incluindo mais pessoas e podendo inclusive ter sinais que possuam sotaque.

# Referências bibliográficas


Lucas Amaral, Givanildo LN Júnior, Tiago Vieira, and Thales Vieira. Evaluating deep models for dynamic brazilian sign language recognition. In *Iberoamerican Congress on Pattern Recognition*, pages 930–937. Springer, 2018.

I. L. O. Bastos, M. F. Angelo, and A. C. Loula. Recognition of static gestures applied to brazilian sign language (libras). In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 305–312, Aug 2015. **DOI** 10.1109/SIBGRAPI.2015.26.

E. J. E. Cardenas and G. C. Chávez. Finger spelling recognition from depth data using direction cosines and histogram of cumulative magnitudes. In *2015 28th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 173–179, Aug 2015. **DOI** 10.1109/SIBGRAPI.2015.49.

François Chollet et al. Keras. https://keras.io, 2015.

Gabriel de Souza Pereira Moreira, Gustavo Ravanhani Matuck, Osamu Saotome, and Adilson Marques da Cunha. Recognizing the brazilian signs language alphabet with neural networks over visual 3d data sensor. In Ana L.C. Bazzan and Karim Pichara, editors, *Advances in Artificial Intelligence – IBERAMIA 2014*, pages 637–648, Cham, 2014. Springer International Publishing. ISBN 978-3-319-12027-0.

Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634. IEEE, 2015.

Edwin Escobedo Cardenas and Guillermo Camara-Chavez. Fusion of deep learning descriptors for gesture recognition. In Marcelo Mendoza and Sergio Velastín, editors, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 212–219, Cham, 2018. Springer International Publishing. ISBN 978-3-319-75193-1.

Myrna S. Felipe, Tanya A. Libras em contexto: curso básico: livro do professor, 2007.

Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. ISBN 013168728X.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

Simon S. Haykin. *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning – with Applications in R*, volume 103 of *Springer Texts in Statistics*. Springer, New York, 2013. ISBN 978-1-4614-7137-0. **DOI** 10.1007/DOI.

Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1): 221–231, 2013.

Jong-Sung Kim, Won Jang, and Zeungnam Bien. A dynamic gesture recognition system for the korean sign language (ksl). *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(2):354–359, 1996.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. Kurakin, Z. Zhang, and Z. Liu. A real time system for dynamic hand gesture recognition with a depth sensor. In *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, pages 1975–1979, Aug 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943. ISSN 1522-9602. **DOI** 10.1007/BF02478259. URL https://doi.org/10.1007/BF02478259.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X, 9780262018258.

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

Ednaldo B. Pizzolato, Mauro dos Santos Anjo, and Guilherme C. Pedroso. Automatic recognition of finger spelling for libras based on a two-layer architecture. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 969–973, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-639-7. **DOI** 10.1145/1774088.1774290. URL http://doi.acm.org/10.1145/1774088.1774290.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 0136042597, 9780136042594.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, CA, USA, 1997. ISBN 0-9660176-3-3.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015. URL http://arxiv.org/abs/1503.00075.

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 4489–4497. IEEE, 2015.

Y. Wang, C. Yang, X. Wu, S. Xu, and H. Li. Kinect based dynamic hand gesture recognition algorithm research. In *2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 1, pages 274–279, Aug 2012.
**DOI** 10.1109/IHMSC.2012.76.

Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.