UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

André Furlan

Autenticação Biométrica de Locutores Drasticamente Disfônicos Aprimorada pela *Imagined Speech*

São José do Rio Preto
2022

André Furlan

Autenticação Biométrica de Locutores Drasticamente Disfônicos Aprimorada pela *Imagined Speech*

Tese apresentada como parte dos requisitos para obtenção do título de Doutor em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista 'Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Orientador: Prof. Dr. Rodrigo Capobianco Guido

São José do Rio Preto

2022

ficha catalografica

André Furlan

Autenticação Biométrica de Locutores Drasticamente Disfônicos Aprimorada pela *Imagined Speech*

Tese apresentada como parte dos requisitos para obtenção do título de Doutor em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista 'Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Comissão Examinadora

Prof. Dr. Rodrigo Capobianco Guido
UNESP – Câmpus de São José do Rio Preto
Orientador

Prof. Dr. Exemplo Jr
Universidade – Câmpus

Prof. Dr. Exempl2
Universidade – Câmpus

São José do Rio Preto

06 de Agosto de 2022

Haha…

Agradecimentos

Agradece

"Citação"

-O Autor.

Resumo

Resumo

Abstract

Summary

1 Introdução

1.1 Considerações Iniciais e Objetivos

1.1.1 Objetivos

Comparar o desempenho das estratégias de *feature learnning* baseadas em *auto-encoders* com técnicas de análise como as *Wavelet-Packet* de Tempo Discreto usando a engenharia paraconsistente de características.

O problema-alvo deste projeto é conceber algoritmos biométricos para autenticar, em princípio por meio da fala, indivíduos com locuções severamente degradadas complementando tais informações com aquelas provenientes dos sinais cerebrais extraídos durante a fonação (imagined speech).

Estudar base de dados existentes, criar a própria base de dados. Depois iniciar com a criação de vetores de características usando *autoencoders* e análise com *wavelets packet tranform*, e classificá-los com redes neurais profundas residuais e recorrentes.

Text-dependent: Uma mesma frase é falada e imaginada

Text-independent: Pode ser falada qualquer coisa tanto no treinamento quanto nos testes.

1.2 Estrutura do trabalho

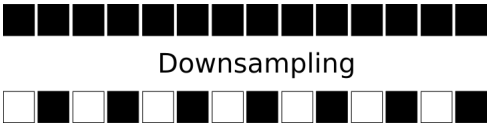
2 Revisão de Bibliográfica

2.1 Conceitos utilizados

2.1.1 Sinais digitais e sub-amostragem (*downsampling*)

Os sinais digitais, tanto de voz quanto aqueles vindos das medições de Eletroencefalograma (ECG), isto é, aqueles que estão amostrados e quantizados (HAYKIN; MOHER, 2011), constituem a base deste trabalho. Além do processo de digitalização, inerente ao ato de armazenar sinais em computadores, os mesmos podem sofrer, a depender da necessidade ou possibilidade, sub-amostragens ou *downsamplings* (POLIKAR et al., 1996). Isso implica em uma estratégia de redução de dimensão e, comumente, ocorre após a conversão de domínio dos sinais com base em filtros digitais do tipo *wavelet*, a serem apresentados adiante. Um exemplo consta na Figura 1, na qual as partes pretas contêm dados e as brancas representam os elementos removidos. Tendo em vista que este trabalho está baseado em sinais digitais de voz e ECG com base em *wavelets*, o processo de sub-amostragem é essencial.

Figura 1 – Sub-amostragem



Fonte: Elaborado pelo autor, 2023.

2.1.2 Caracterização dos processos de produção da voz humana

A fala possui três grandes áreas de estudo: A fisiológica, também conhecida como fonética articulatória, a acústica, referida como fonética acústica, e ainda, a perceptual, que cuida da percepção da fala (KREMER; GOMES, 2014). Neste trabalho, o foco será apenas na questão acústica, pois não serão analisados aspectos da fisiologia relacionada à voz, mas sim os sinais sonoros propriamente ditos.

2.1.2.1 Sinais vozeados *versus* não-vozeados

Quando da análise dos sinais de voz, consideram-se as partes vozeadas e não-vozeadas. Aquelas são produzidas com a ajuda da vibração quase periódica das pregas vocais, enquanto estas praticamente não contam com participação regrada da referida estrutura.

2.1.2.2 Frequência fundamental da voz

Também conhecida como F_0, é o componente periódico resultante da vibração das pregas vocais. Em termos de percepção, se pode interpretar F_0 como o tom da voz, isto é, a frequência de *pitch* (KREMER; GOMES, 2014). Vozes agudas tem uma frequência de *pitch* alto, enquanto vozes mais graves tem baixa. A alteração da frequência (jitter) e/ou intensidade (shimmer) do *pitch* durante a fala é definida como entonação, porém, também pode indicar algum distúrbio ou doença relacionada ao trato vocal (WERTZNER; SCHREIBER; AMARO, 2005).

A frequência fundamental da voz é o número de vezes na qual uma forma de onda característica, que reflete a excitação pulmonar moldada pelas pregas vocais, se repete por unidade de tempo. Sendo assim, as medidas de F_0 geralmente são apresentadas em Hz (FREITAS, 2013).

A medição de F_0 está sujeita a contaminações surgidas das variações naturais de *pitch* típicas da voz humana (FREITAS, 2013). A importância de se medir F_0 corretamente vem do fato de que, além de carregar boa parte da informação da fala, ela é a base para construção das outras frequências que compõe os sinais de voz, que são múltiplas de F_0.

2.1.2.3 Formantes

O sinal de excitação que atravessa as pregas vocais é rico em harmônicas, isto é, frequências múltiplas da fundamental. Tais harmônicas podem ser atenuadas ou amplificadas, em função da estrutura dos tratos vocal e nasal de cada locutor. Particularmente, o primeiro formante (F_1), relaciona-se à amplificação sonora na cavidade oral posterior e à posição da língua no plano vertical; o segundo formante (F_2) à cavidade oral anterior e à posição da língua no plano horizontal; o terceiro formante (F_3) relaciona-se às cavidades à frente e atrás do ápice da língua e, finalmente, o quarto formante (F_4) relaciona-se ao formato da laringe e da faringe na mesma altura (VALENÇA et al., 2014). Formantes caracterizam fortemente os locutores, pois cada indivíduo possui um formato de trato vocal e nasal. Assim, tais frequências, que podem ser capturadas com ferramentas diversas, a exemplo da Transformada *Wavelet*, são de suma importância na área de verificação de locutores.

2.1.3 Escalas e energias dos sinais

A energia de um sinal digital $s[\cdot]$ com M amostras é definida como

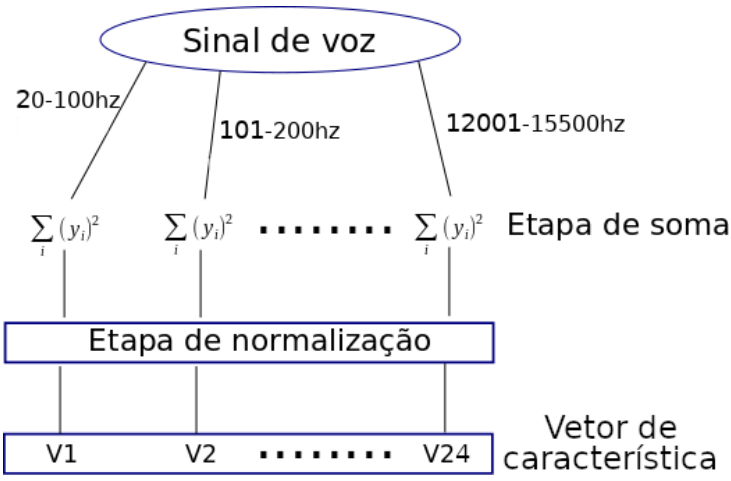
$$E = \sum_{i=0}^{M-1} (s_i)^2 \qquad . \tag{2.1}$$

E pode ainda sofrer normalizações e ter a sua mensuração restrita a uma parte específica do sinal sob análise. Possibilidades para tais restrições podem, por exemplo, envolver a escala BARK (ZWICKER, 1961) e MEL (BERANEK, 1949) que serão utilizadas neste trabalho.

2.1.3.1 A escala BARK

BARK foi definida tendo em mente vários tipos de sinais acústicos. Essa escala corresponde ao conjunto de 25 bandas críticas da audição humana. Suas frequências-base de audiometria são, em Hz: **20, 100, 200, 300, 400, 510, 630, 770, 920, 1080, 1270, 1480, 1720, 2000, 2320, 2700, 3150, 3700, 4400, 5300, 6400, 7700, 9500, 12000, 15500**. Nessa escala,os sinais digitais no domínio temporal atravessam filtros passa-faixas (BOSI; GOLDBERG, 2002) para os quais o início e o final da banda de passagem correspondem à frequências-base consecutivas resultando em um vetor de características com 24 coeficientes e, em seguida, as energias dos sinais filtrados são utilizadas como características descritivas de propriedades do sinal sob análise, como mostrado na Figura 2.

Figura 2 – Cálculo de vetores de características com BARK



Fonte: Elaborado pelo autor, 2023.

2.1.3.2 A escala MEL

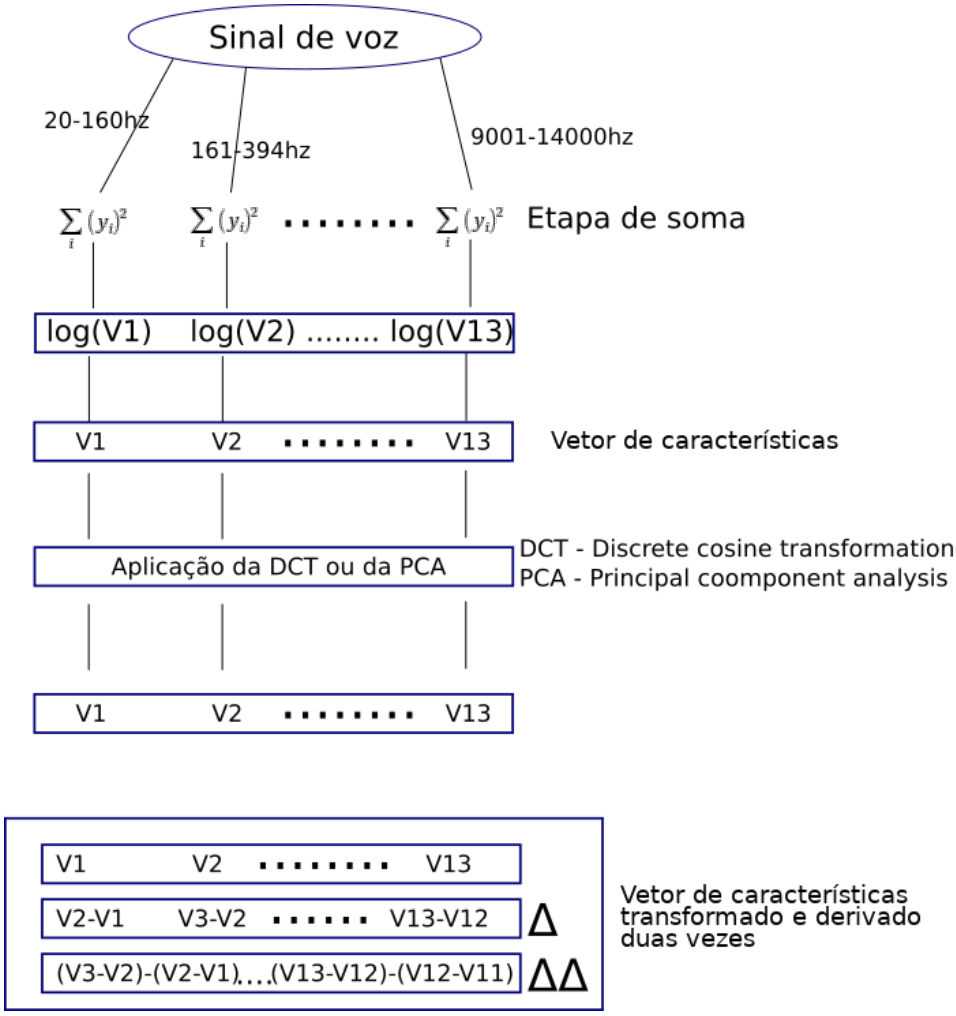
Escala Mel, advinda do termo *melody*, é uma adaptação da escala Bark para sinais de voz. Dentre as várias implementações de bandas críticas a escolhida foi a implementação que contém os valores em Hz: **20, 160, 394, 670, 1000, 1420, 1900, 2450, 3120, 4000, 5100, 6600, 9000, 14000**.

A variante que será usada neste trabalho é conhecida como *Mel-frequency cepstral coefficients*(MFCC) a qual inclui, além dos intervalos definidos, uma diminuição da correlação entre os componentes gerados via aplicação da Transformada Discreta Cosseno (DCT) (SALOMON; MOTTA; BRYANT, 2007) ou da Análise de Componentes Principais

(PCA) (JOLLIFFE, 2002) seguida de duas derivações no vetor de características resultando em um total de 11 coeficientes. Nesse trabalho foi escolhida a DCT, no entanto, PCA poderia também ser escolhida sem prejuízos, o uso de uma ou outra depende da preferência do autor.

Novamente, desconsiderando qualquer etapa intermediária que possa ser adicionada, as energias calculadas nos intervalos definidos na escala MEL podem, por si mesmas, constituir um vetor de características, como mostrado na Figura 2.

Figura 3 – Cálculo de vetores de características com MEL



Fonte: Elaborado pelo autor, 2023.

2.1.4 Filtros digitais *wavelet*

Filtros digitais *wavelet* têm sido utilizados com sucesso para suprir as deficiências de janelamento de sinal apresentadas pelas Transformadas de Fourier e de Fourier de Tempo Reduzido. *Wavelets* contam com variadas funções-filtro e têm tamanho de janela variável, o que permite uma análise multirresolução (ADDISON; WALKER; GUIDO, 2009). Particularmente, as *wavelets* proporcionam a análise do sinal de forma detalhada tanto no

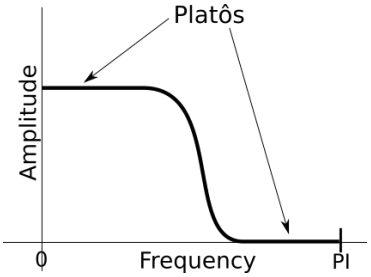
espectro de baixa frequência quanto no de alta contando com diferentes funções-base não periódicas diferentemente da tradicional transformada de Fourrier que utilizam somente as bases periódicas senoidal e cossenoidal.

É importante observar que, quando se trata de Transformadas *Wavelet*, seis elementos estão presentes: dois filtros de análise, dois filtros de síntese e as funções ortogonais *scaling* e *wavelet*. No tocante a sua aplicação, só a transformada direta, e não a inversa, será usada na construção dos vetores de características. Portanto, os filtros de síntese, a função *scaling* e a função *wavelet* não serão elementos abordados aqui: eles somente interessariam caso houvesse a necessidade da transformada inversa.

No contexto dos filtros digitais baseados em *wavelets*, o tamanho da janela recebe o nome de **suporte**. Janelas definem o tamanho do filtro que será aplicado ao sinal. Quando esse é pequeno (limitado), se diz que a janela tem **um suporte compacto** (POLIKAR et al., 1996).

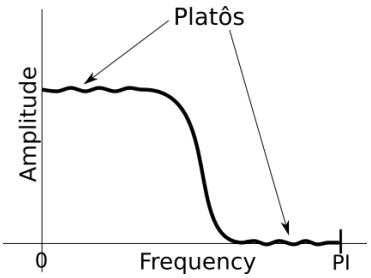
Se diz que uma *wavelet* tem boa **resposta em frequência** quando, na aplicação da mesma para filtragem, não são causadas muitas pertubações indesejadas ao sinal, no domínio da frequência. Os filtros *wavelet* de Daubechies (DAUBECHIES, 1992) se destacam nesse quesito por serem *maximamente planos* (*maximally-flat*) (BUTTERWORTH, 1930) (BIANCHI, 2007) nos platôs de resposta em frequência como indicado na Figura 4 ao contrário do que ocorre na Figura 5.

Figura 4 – Platôs maximamente planos em um filtro digital: característica da família de Daubechies



Fonte: Elaborado pelo autor, 2023.

Figura 5 – Platôs não maximamente planos de um filtro digital: características de outros filtros *wavelet*, distintos da família de Daubechies



Fonte: Elaborado pelo autor, 2023.

Além da resposta em frequência, na aplicação de um filtro digital *wavelet* também é possível considerar a **resposta em fase**, que constitui um atraso ou adiantamento do sinal filtrado em relação ao sinal original, ambos no domínio temporal. Esse deslocamento pode ser **linear**, **quase linear** ou **não linear**:

- na resposta em fase **linear**, há o mesmo deslocamento de fase para todos os componentes do sinal;
- quando a resposta em fase é **quase linear** existe uma pequena diferença no deslocamento dos diferentes componentes do sinal;
- finalmente, quando a resposta é **não linear**, acontece um deslocamento significativamente heterogêneo para as diferentes frequências que compõe o sinal.

Idealmente, é desejável que todo filtro apresente boa resposta em frequência e em fase linear. Características de fase e frequência de algumas famílias de filtros *wavelet* constam na Tabela 1.

Tabela 1 – Algumas das *wavelets* mais usadas e suas propriedades

Wavelet	**Resposta em frequência**	**Resposta em fase**
Haar	Pobre	Linear
Daubechies	mais próxima da ideal à medida que o suporte aumenta; *maximally-flat*	Não linear
Symmlets	mais próxima da ideal à medida que o suporte aumenta; não *maximally-flat*	Quase linear
Coiflets	mais próxima da ideal à medida que o suporte aumenta; não *maximally-flat*	Quase linear

Fonte: Elaborado pelo autor, 2023.

2.1.4.1 O algoritmo de Mallat para a Transformada *Wavelet*

Baseando-se no artigo (GUIDO, 2015), percebe-se que algoritmo de Mallat faz com que aplicação das *wavelets* seja uma simples multiplicação de matrizes. O sinal que deve ser transformado se torna uma matriz linear vertical. Os filtros passa-baixa e passa-alta tornam-se, nessa ordem, linhas de uma matriz quadrada que será completada segundo regras que serão mostradas mais adiante. É importante que essa matriz quadrada tenha a mesma dimensão que o sinal a ser transformado.

Interessantemente, para que seja possível a transformação *wavelet*, basta ter disponível o vetor do filtro passa-baixas calculado a partir da *mother wavelet*, que é a função geradora desse filtro, já que o passa-alta pode ser construído a partindo-se da ortogonalidade do primeiro.

Determinar a ortogonal de um vetor significa construir um vetor, tal que, o produto escalar do vetor original com sua respectiva ortogonal seja nulo.

Considerando $h[\cdot]$ como sendo o vetor do filtro passa-baixas e $g[\cdot]$ seu correspondente ortogonal, tem-se que $h[\cdot] \cdot g[\cdot] = 0$.

Portanto, se $h[\cdot] = [a, b, c, d]$ então seu ortogonal será $g[\cdot] = [d, -c, b, -a]$ pois:

$$h[\cdot]\cdot g[\cdot] = [a, b, c, d]\cdot[d, -c, b, -a] = (a\cdot d)+(b\cdot(-c))+(c\cdot b)+(d\cdot(-a)) = ad-ad+bc-bc = 0 \quad .$$

A título de exemplo, considera-se:

- o filtro passa baixa baseado na *wavelet* Haar: $h[\cdot] = [\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}]$
- o seu respectivo vetor ortogonal: $g[\cdot] = [\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}]$
- e também o seguinte sinal-exemplo de entrada: $s = \{1, 2, 3, 4\}$

Se o tamanho do sinal a ser tratado é quatro e se pretende-se aplicar o filtro Haar, a seguinte matriz de coeficientes é construída:

$$\begin{pmatrix} \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0 \\ \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, 0 \\ 0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \\ 0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \end{pmatrix} \tag{2.2}$$

Tendo em vista que a dimensão do sinal sob análise é diferente da dimensão do filtro, basta completar cada uma das linhas da matriz de coeficientes com zeros. A matriz é montada de forma que ela seja ortogonal.

Montada a matriz de filtros, segue-se com os cálculos da transformada:

$$\begin{pmatrix} \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0 \\ \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, 0 \\ 0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \\ 0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} \frac{3}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \\ \frac{7}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} \end{pmatrix} \tag{2.3}$$

Realizada a multiplicação, é necessário montar o sinal filtrado. Isso é feito escolhendo, dentro do resultado, valores alternadamente de forma que o vetor resultante seja:

$$resultado = \Big[\frac{3}{\sqrt{2}}, \frac{7}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}\Big] \quad . \tag{2.4}$$

Percebe-se que, na transformação descrita nas Equações 2.2, 2.3 e 2.4, a **aplicação dos filtros sobre o vetor de entrada ocorreu apenas uma vez**. Sendo assim, se

diz que o sinal recebeu uma **transformação de nível 1**. A cada transformação, há uma separação do sinal em dois componentes: o de baixa e o de alta frequência.

Embora haja um limite, que será mencionado adiante, é possível aplicar mais de um nível de decomposição ao sinal. Para que se possa fazer isso, a Transformada *Wavelet* nível 2 deve considerar apenas a parte de baixas frequências da primeira transformada; a transformada de nível 3 deve considerar apenas a parte de baixas frequências da transformada nível 2, e assim consecutivamente.

Nos exemplos numéricos mostrados nas Tabelas 2, 3 e 4, usou-se um filtro normalizado cujos coeficientes são $\{\frac{1}{2}, -\frac{1}{2}\}$. Os dados destacados em **verde** correspondem ao **vetor original** que será tratado. Cada uma das linhas são os resultados das transformações nos níveis 1, 2, 3 e 4, respectivamente. As partes em **azul** correspondem à porção de **baixas frequências**, enquanto que as partes em **amarelo** correspondem às porções de **altas frequências**.

Percebe-se que na Tabela 2, a partir da transformação nível 2, apenas as partes de baixa frequência são modificadas. Isso implica que, no momento da implementação do algoritmo de Mallat **para níveis maiores que 1**, a abordagem será **recursiva**. Em outras palavras, a partir do nível 1 se deve aplicar Mallat apenas às porções de baixas-frequências geradas pela transformação anterior.

Tabela 2 – Exemplo numérico da transformação *wavelet* aplicada a um vetor

Sinal	**32**	**10**	**20**	**38**	**37**	**28**	**38**	**34**	**18**	**24**	**24**	**9**	**23**	**24**	**28**	**34**
Nível 01	21	29	32,5	36	21	16,5	23,5	31	11	-9	4,5	2	-3	7,5	-0,5	-3
Nível 02	25	34,25	18,75	27,25	-4	-1,75	2,25	-3,75	11	-9	4,5	2	-3	7,5	-0,5	-3
Nível 03	29,62	23	-4,625	-4,25	-4	-1,75	2,25	-3,75	11	-9	4,5	2	-3	7,5	-0,5	-3
Nível 04	26,3125	3,3125	-4,625	-4,25	-4	-1,75	2,25	-3,75	11	-9	4,5	2	-3	7,5	-0,5	-3

Fonte: Elaborado pelo autor, 2023.

2.1.4.2 O algoritmo de Mallat e a Transformada *Wavelet-Packet*

Na Transformada *Wavelet-Packet*, os filtros aplicados são os mesmos da Transformada *Wavelet* e o procedimento recursivo de cálculo também é o mesmo, no entanto, realizada a transformação de nível 1, a transformada de nível 2 deve ser aplicada aos componentes de baixa e de alta frequência. Sendo assim a Transformada *Wavelet-Packet* obtém um nível de detalhes em todo o espectro de frequência, maior do que uma transformação regular.

Os exemplos mostrados nas Tabelas 3 e 4 permitem perceber como se dão as transformações na porção de **baixa** e de **alta** frequências, respectivamente, após a transformação *wavelet-packet* de nível 1, 2, 3 e 4.

Devido ao *downsampling* aplicado às porções de alta frequência, essas partes acabam por ficar "espelhadas" no espectro (JENSEN; COUR-HARBO, 2001), ou seja, suas sequências ficam invertidas. Para resolver esse problema e preservar a ordem das sub-bandas no sinal transformado, os filtros são aplicados em ordem inversa nas porções de alta frequência. Isso altera como o algoritmo de Mallat deve ser implementado para a Transformada *Wavelet-Packet*, já que dessa vez é preciso se atentar a ordem da aplicação dos filtros passa-alta e passa-baixa.

Tabela 3 – Exemplo numérico de *wavelet-packet* Haar aplicada ao vetor da Tabela 2 (porção das baixas frequências)

Sinal	**32**	**10**	**20**	**38**	**37**	**28**	**38**	**34**
Nivel 01	21	29	32,5	36	21	16,5	23,5	31
Nivel 02	25	34,25	18,75	27,25	-4	-1,75	2,25	-3,75
Nivel 03	29,62	23	-4,625	-4,25	-1,125	3	-2,875	-0,75
Nivel 04	26,3125	3,3125	-0,1875	-4,4375	0,9375	-2,0625	-1,0625	-1,8125

Fonte: Elaborado pelo autor, 2023.

Tabela 4 – Exemplo numérico de *wavelet-packet* Haar aplicada ao vetor da Tabela 2 (porção das altas frequências)

Sinal	**18**	**24**	**24**	**9**	**23**	**24**	**28**	**34**
Nivel 01	11	-9	4,5	2	-3	7,5	-0,5	-3
Nivel 02	10	1,25	-5,25	1,25	1	3,25	2,25	-1,75
Nivel 03	5,625	-2	4,375	-3,25	-1,125	2	2,125	0,25
Nivel 04	1,8125	3,8125	3,8125	0,5625	0,4375	-1,5625	0,9375	1,1875

Fonte: Elaborado pelo autor, 2023.

2.1.5 Engenharia Paraconsistente de características

Nos processos de classificação, frequentemente surge a questão: "Os vetores de características criados proporcionam uma boa separação de classes?". A Engenharia Paraconsistente de Características, recém publicada (GUIDO, 2019), que usa a paraconsistência (COSTA; BÉZIAU; BUENO, 1998), (COSTA; ABE, 2000) é, em meio a outras técnicas, uma ferramenta que pode ser usada para responder essa questão.

O processo inicia-se após a aquisição dos vetores de características para cada classe C_n. Se o número de classes presentes for, por exemplo, quatro então estas poderão ser representadas por C_1, C_2, C_3, C_4.

Em seguida é necessário o cálculo de duas grandezas:

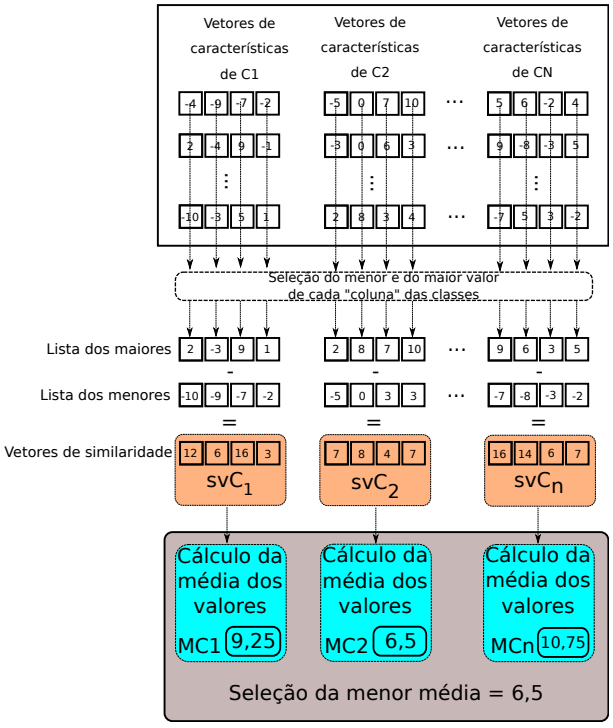
- a menor similaridade intraclasse, α.
- a razão de sobreposição interclasse, β.

α indica o quanto de similaridade os dados têm entre si, dentro de uma mesma classe, enquanto β é a razão de sobreposição entre diferentes classes. Idealmente, α deve ser maximizada e β minimizada para que classificadores extremamente modestos apresentem uma acurácia interessante.

Particularmente, para calcular α e β, é necessária a normalização dos vetores de características de forma que todos os seus componentes estejam no intervalo entre 0 e 1. Em seguida, a obtenção de α se dá selecionando-se os maiores e os menores valores de cada uma das posições de todos os vetores de características para cada classe, gerando assim um vetor para os valores maiores e outro para os menores.

O **vetor de similaridade da classe**(svC_n) é obtido fazendo-se a diferença item-a-item dos maiores em relação aos menores. Finalmente, e para cada classe, é obtida a média dos valores para cada vetor de similaridade, sendo que α é o menor valor dentre essas médias. A Figura 6 contém uma ilustração do processo.

Figura 6 – Cálculo do coeficiente α.



Fonte: Adaptado de (GUIDO, 2019).

A obtenção de β, assim como ilustrado na Figura 7, também se dá selecionando os maiores e os menores valores de cada uma das posições de todos os vetores de características de cada classe, gerando assim um vetor para os valores maiores e outro para os menores.

Na sequência, realiza-se o cálculo de R cujo valor é a quantidade de vezes que um valor do vetor de características de uma classe se encontra entre os valores maiores e menores de outra classe.

Seja:

- N a quantidades de classes;
- X a quantidade de vetores de características por classe;
- T o tamanho do vetor de características.

Então, F, que é o número máximo de sobreposições possíveis entre classes, é dado por:

$$F = N.(N-1).X.T \quad . \tag{2.5}$$

Finalmente, β é calculado da seguinte forma:

$$\beta = \frac{R}{F} \quad . \tag{2.6}$$

Neste ponto, é importante notar que $\alpha = 1$ sugere fortemente que os vetores de características de cada classe são similares e representam suas respectivas classes precisamente. Complementarmente, $\beta = 0$ sugere os vetores de características de classes diferentes não se sobrepõe (GUIDO, 2019).

Considerando-se o plano paraconsistente (GUIDO, 2019), temos:

- Verdade $\rightarrow$ fé total ($\alpha = 1$) e nenhum descrédito ($\beta = 0$)
- Ambiguidade $\rightarrow$ fé total ($\alpha = 1$) e descrédito total ($\beta = 1$)
- Falsidade $\rightarrow$ fé nula ($\alpha = 0$) e descrédito total ($\beta = 1$)
- Indefinição $\rightarrow$ fé nula ($\alpha = 0$) e nenhum descrédito ($\beta = 0$) .

No entanto, raramente α e β terão valores inteiros como os mostrados na listagem acima: Na maioria das ocasiões, $0 \leqslant \alpha \leqslant 1$ e $0 \leqslant \beta \leqslant 1$. Por isso, se torna necessário o cálculo do **grau de certeza**, isto é, G_1, e do **grau de contradição**, isto é, G_2, conforme segue:

$$G_1 = \alpha - \beta \quad , \tag{2.7}$$

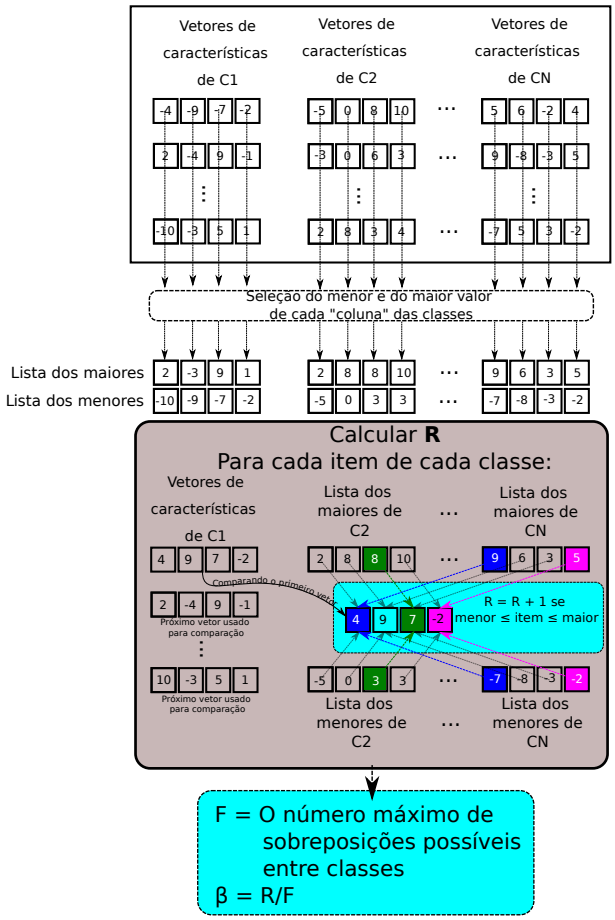
$$G_2 = \alpha + \beta - 1 \quad , \tag{2.8}$$

onde: $-1 \leqslant G_1$ e $1 \geqslant G_2$.

Os valores de G_1 e G_2, em conjunto, definem os graus entre verdade ($G_1 = 1$) e falsidade ($G_1 = -1$) e também os graus entre indefinição ($G_2 = -1$) e ambiguidade ($G_2 = 1$). Novamente, raramente tais valores inteiros serão alcançados já que G_1 e G_2 dependem de α e β.

O Plano Paraconsistente, para fins de visualização e maior rapidez na avaliação dos resultados, encontra-se ilustrado na Figura 8 e tem quatro arestas precisamente definidas:

Figura 7 – Cálculo de β: Os itens destacados em azul e rosa são aqueles pertencentes a classe C1 e CN que se sobrepõe, em verde, a sobreposição é entre C1 e C2. Para cada sobreposição verificada soma-se 1 ao valor R. Essa comparação é feita para todos os vetores de características de cada uma das classes.



Fonte: Adaptado de (GUIDO, 2019).

- (-1,0) → falsidade;
- (1,0) → verdade;
- (0,-1) → indefinição;
- (0,1) → ambiguidade.

A propósito de ilustração na Figura 8, é possível ver um pequeno círculo indicando os graus dos quatro casos listados.

Para se ter ideia em que área exatamente se encontram as classes avaliadas, as distâncias (D) do ponto $P = (G_1, G_2)$ até o limites supracitados podem ser computadas. Tais cálculos podem ser feitos da seguinte forma:

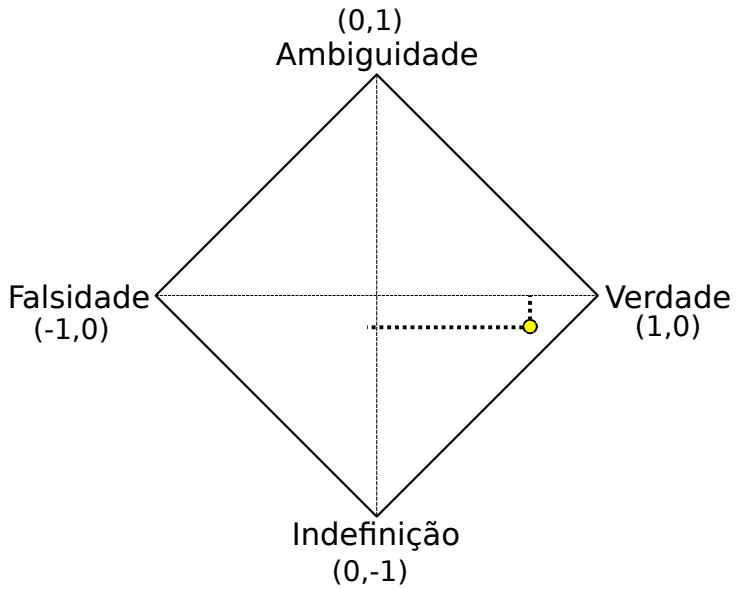
$$D_{-1,0} = \sqrt{(G_1 + 1)^2 + (G_2)^2} \quad , \tag{2.9}$$

$$D_{1,0} = \sqrt{(G_1 - 1)^2 + (G_2)^2} \quad , \tag{2.10}$$

$$D_{0,-1} = \sqrt{(G_1)^2 + (G_2 + 1)^2} \quad , \tag{2.11}$$

$$D_{0,1} = \sqrt{(G_1)^2 + (G_2 - 1)^2} \quad . \tag{2.12}$$

Figura 8 – O plano paraconsistente: O pequeno círculo indica os graus de falsidade(-1,0), verdade(1,0), indefinição(0,-1) e ambiguidade(0,1)



Fonte: Adaptado de (GUIDO, 2019).

Na prática, ou seja, para fins de classificação, geralmente considera-se a distância em relação ao ponto *"(1,0)* $\rightarrow$ *Verdade"*, que é o ponto ótimo: quanto mais próximo o ponto (G_1, G_2) estiver de $(1, 0)$, mais as os vetores de características das diferentes classes estão naturalmente separados. Isso implica, dentro da limitação de cada algoritmo, em resultados melhores sejam quais forem os classificadores usados.

2.1.6 Brain-Computer Interface and EEG

Among the methods of Brain-Computer Interface (BCI), Electroencephalogram (EEG) stands out as the most cost-effective and simple system to implement. However, it does have some quirks, such as high sensitivity to electromagnetic interference and difficulty in capturing the signal due to suboptimal scalp placement of the electrodes. So one of the fundamental aspects that any EEG processing system must have is the tolerance to noise (Jalaly Bidgoly; Jalaly Bidgoly; AREZOUMAND, 2020).

The *wet* electrodes are placed using conductive gel and are less prone to movement artifacts which are electromagnetic interferences caused by some motion like blinking. *Dry* electrodes do not need the gel but is more sensible to artifacts.

EEG records the electrical activities of the brain, typically placing along the scalp surface electrodes. These electrical activities result from ionic current flows induced by the synchronized synaptic activation of the brain's neurons. They manifest as rhythmic voltage fluctuations ranging from 5 to $100\mu V$ in amplitude and between 0.5 and 40 Hz in frequency(Jalaly Bidgoly; Jalaly Bidgoly; AREZOUMAND, 2020). The operational frequencies bands in the brain are following (SANEI; CHAMBERS, 2021):

- **Delta (1–4Hz)**: The slowest and usually the highest amplitude waveform. The Delta band is observed in babies and during deep sleep in adults.
- **Theta (4–8Hz)**: Observed in children, drowsy adults, and during memory recall. Theta wave amplitude is typically less than $100\mu V$.
- **Alpha (8–12Hz)**: Usually the dominant frequency band, appearing during relaxed awareness or when eyes are closed. Focused attention or relaxation with open eyes reduces the amplitude of the Alpha band. These waves are normally less than 50 μV.
- **Beta (12–25Hz)**: Associated with thinking, active concentration, and focused attention. Beta wave amplitude is normally less than 30 μV.
- **Gamma (over 25Hz)**: Observed during multiple sensory processing. Gamma patterns have the lowest amplitude.

According to (Jalaly Bidgoly; Jalaly Bidgoly; AREZOUMAND, 2020), for most of the tasks brain do, there are regions associated with it as seen in Table 5.

2.1.7 10-20 system and the brain's areas

Tabela 5 – Brain tasks and its corresponding regions. See Figure 9 for more information.

Region	Channels	Tasks
Frontal lobe	Fp1, Fp2, Fpz, Pz, F3, F7, F4, F8	Memory, concentration, emotions.
Parietal lobe	P3, P4, Pz	Problem Solving, attention, grammar, sense of touch.
Temporal lobe	T3, T5, T4, T6	Memory, recognition of faces, hearing, word and social clues.
Occipital lobe	O1, O2, Oz	Reading, vision.
Cerebellum		Motor control, balance.
Sensorimotor Cortex	C3, C4, Cz	Attention, mental processing, fine motor control, sensory integration.

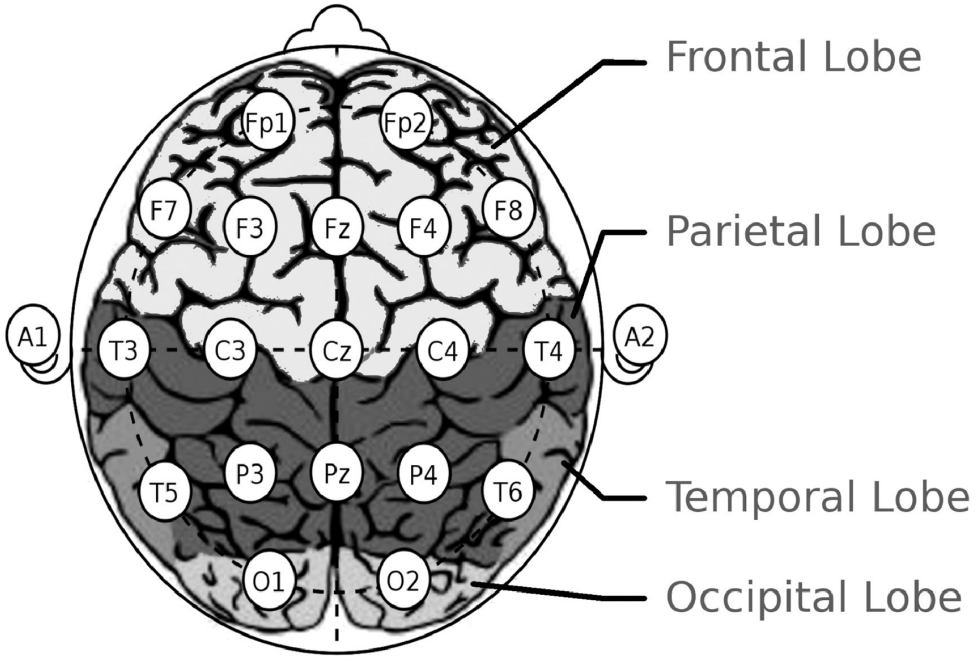


Figura 9 – Electrodes placement according to 10-20 standard (VICENTE, 2023) and brain's lobes. Odd numbers are assigned to the electrodes on the left hemisphere, and even numbers are assigned to the electrodes on the right. Source (Jalaly Bidgoly; Jalaly Bidgoly; AREZOUMAND, 2020)

2.1.8 Spiking Neural Networks

Neural Networks (NNs), as defined here as *a multilayered, fully connected, with or without recurrent or convolutional layers network*, require that all neurons are activated in both the forward and backward passes. This implies that every unit in the network must process some data, leading to power consumption (ESHRAGHIAN et al., 2023).

The sensory system of biological neurological systems converts external data, such as light, odors, touch, flavors, and others, into **spikes**. A spike is a voltage that convey information (KASABOV, 2019). These ones are then transmitted along the neuronal chain to be processed, generating a response to the environment.

The biological neuron only spikes when a certain level of excitatory signals get accumulated above some threshold in its soma staying inactive when there is no signal, therefore, this type of cell is very efficient in terms of energy consumption and processing.

In order to get the aforementioned advantages the Spiking Neural Networks (SNNs) instead of employing continuous activation values, like NNs, SNNs utilize **spikes** at the input, hidden and outputs layers. SNNs can have continuous inputs as well and keep its properties.

A SNN **is not** a one-to-one simulation of neurons. Instead, it approximates certain computational capabilities of specific biological properties. Some studies like (JONES;

KORDING, 2020) created models way closer to natural neurons exploring the nonlinearity of dendrites and another neuron features yielding remarkable results in the classification.

As can be seen in Figure 10 SNNs neurons, given the correct parameters, are very noise tolerant because it acts as a **lowpass filter**. Generating spikes even when a considerably level of interference is present. This units are very time-oriented too, being great when it comes to process streams of data (ESHRAGHIAN et al., 2023).

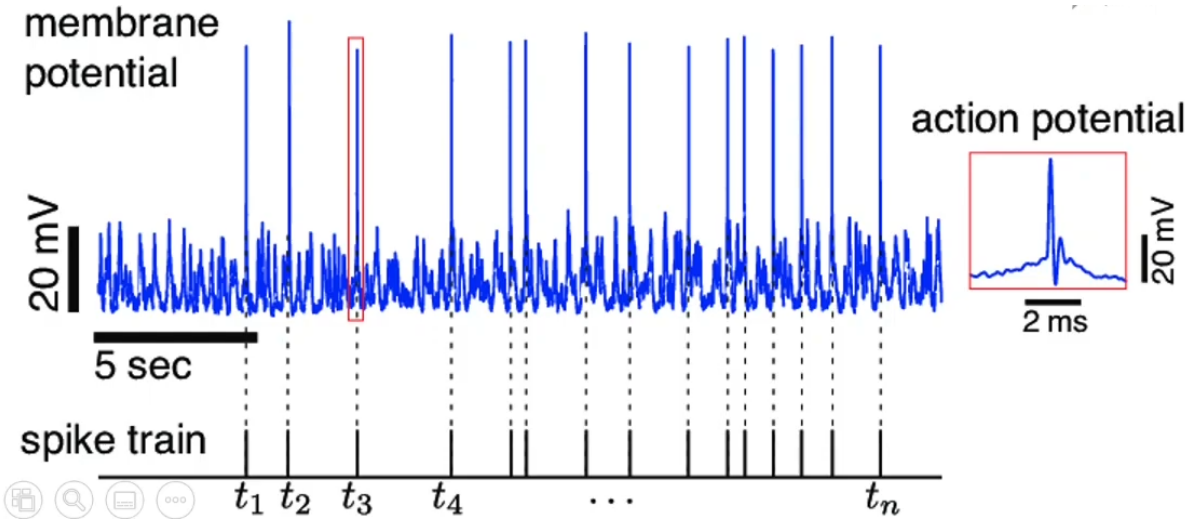


Figura 10 – Spikes from a noisy signal. Source (GOODMAN et al., 2022)

2.1.9 Spiking Neuron

Although the focus of this work is the *Leaky Integrate and Fire Neurons* (LIF) because is simpler, more efficient and currently generalize better for most of the problems (GOODMAN et al., 2022), there are more biological accurate model like **Hodgkin-Huxley neuron** (GERSTNER et al., 2014) and ones like (JONES; KORDING, 2020) that created models closer to natural neurons exploring the nonlinearity of dendrites and other neuron features.

2.1.9.1 Leaky Integrate and Fire Neuron

The Leaky Integrate and Fire Neuron (LIF) is one of the simplest neuron models in SNNs, still, it can be applied successfully to most of the problems in with SNNs can be used.

LIF, like a NN neuron takes the sum of weighted inputs but, rather than pass it directly to its activation function, some *leakage* is applied, which decreases in some degree the sum.

LIF behave much like Resistor-Capacitor circuits as can be seen in Figure 11. Here R is resistance to the leakage, I_{in} the input current, C the capacitance, U_{mem} means the

accumulated action potential and v is a switch that lets the capacitor discharge (i.e. emit a spike) when some potential threshold is reached.

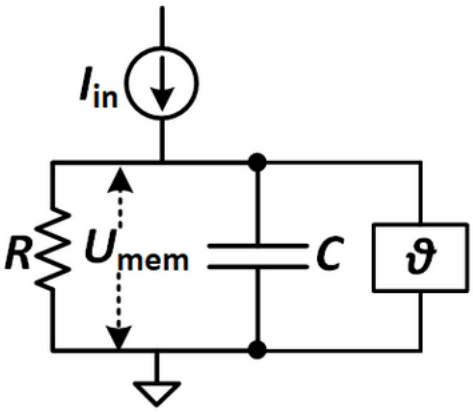


Figura 11 – RC model: Source: (ESHRAGHIAN et al., 2023)

Unlike the Hodgkin-Huxley neuron, spikes are represented as **sparsely** distributed **ones** in a train of **zeros**, as illustrated in Figure 12 and 13. This approach simplify the models and reduces the computational power and needed storage to run a SNN.

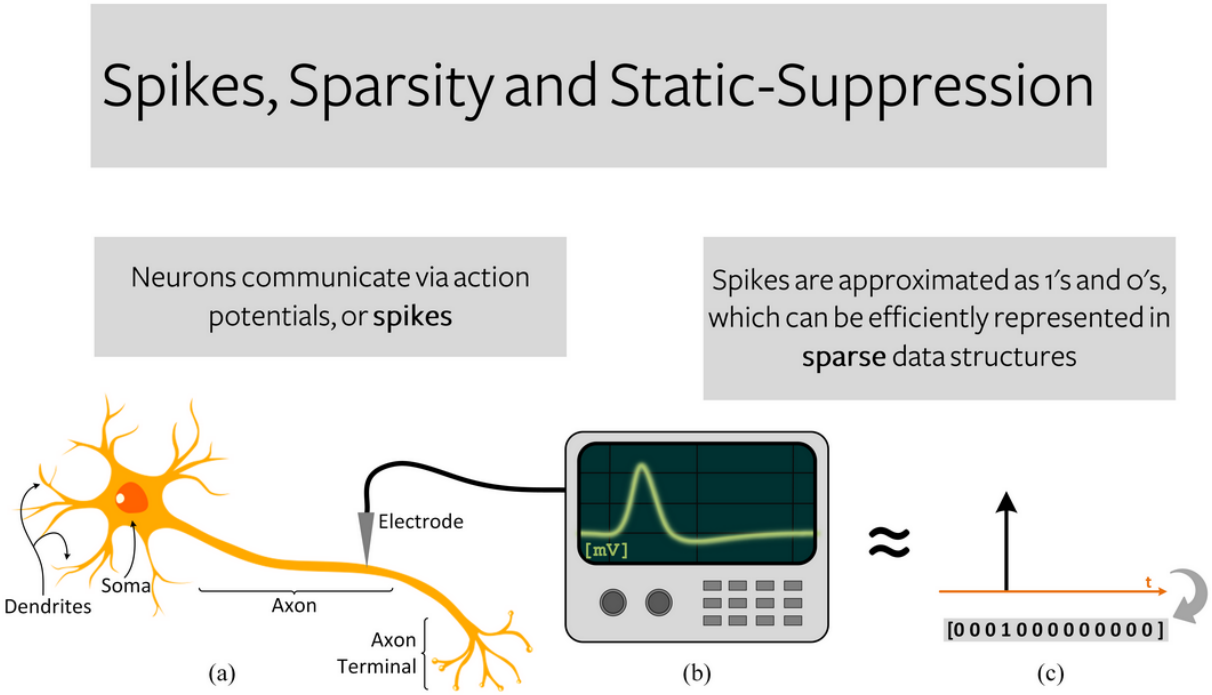


Figura 12 – Sparsity on Spike Neuron Networks. Source: (ESHRAGHIAN et al., 2023)

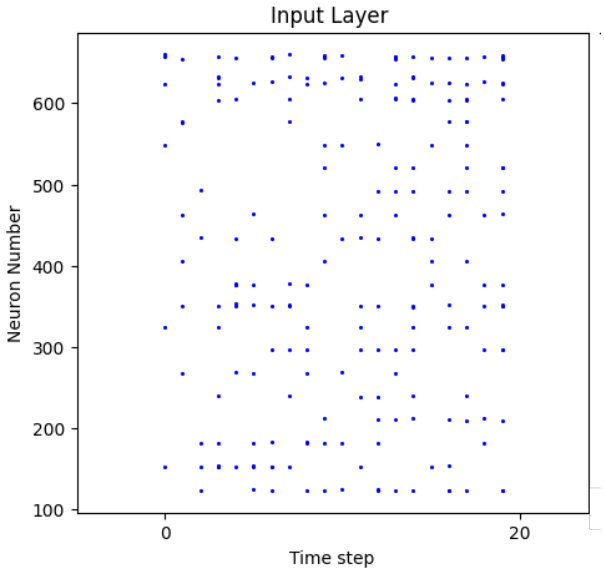


Figura 13 – Sparse activity of a SNN: The horizontal axis represents the time step of some data being processed the vertical are the SNN neuron number. Note that, most of the time, very few neurons gets activated. Source: The author.

As result of the mentioned above, in SNN information is coded in format of *timing* and/or *rate* of spikes giving consequently great capabilities of processing streams of data but limiting the processing of static data.

LIF model is governed by the Equations bellow (ESHRAGHIAN et al., 2023).

Considering that Q is a measurement of electrical charge and $V_{mem}(t)$ is the potential difference at the membrane in a certain time t than the neuron capacitance C is given by the Equation 2.13.

$$C = \frac{Q}{V_{mem}(t)} \tag{2.13}$$

Than the neuron charge can be expressed as Equation 2.14.

$$Q = C.V_{mem}(t) \tag{2.14}$$

To know how these charge changes according to the time (aka current) we can derivate Q as in Equation 2.15. This expression express the current in the capacitive part of the neuron I_C

$$I_C = \frac{dQ}{dt} = C.\frac{dV_{mem}(t)}{dt} \tag{2.15}$$

To calculate the total current passing by the resistive part of the circuit we may use the Ohm's law:

$$V_{mem}(t) = R.I_R \implies I_R = \frac{V_{mem}(t)}{R} \tag{2.16}$$

Than considering that the total current do not change, as seen in Figure 14, we have the total input current I_{in} of the neuron as in Equation 2.17.

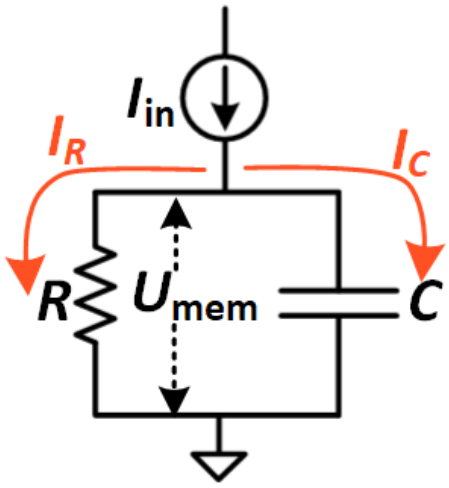


Figura 14 – RC model for currents: $I_{in} = I_R + I_C$

$$I_{in}(t) = I_R + I_C \implies I_{in}(t) = \frac{V_{mem}(t)}{R} + C.\frac{dV_{mem}(t)}{dt} \tag{2.17}$$

Therefore, to describe the passive membrane we got a linear Equation 2.18.

$$\begin{aligned} I_{in}(t) &= \frac{V_{mem}(t)}{R} + C.\frac{dV_{mem}(t)}{dt} \implies \\ I_{in}(t) - \frac{V_{mem}(t)}{R} &= C.\frac{dV_{mem}(t)}{dt} \implies \\ \boxed{R.I_{in}(t) - V_{mem}(t) &= R.C.\frac{dV_{mem}(t)}{dt}} \end{aligned} \tag{2.18}$$

Than, if we consider $\tau = R.C$ as the **membrane time constant** we get voltages on both sides of Equation 2.19 which **describes the RC circuit**.

$$\begin{aligned} R.I_{in}(t) - V_{mem}(t) &= R.C.\frac{dV_{mem}(t)}{dt} \implies \\ R.I_{in}(t) - V_{mem}(t) &= \tau.\frac{dV_{mem}(t)}{dt} \implies \\ \boxed{\tau.\frac{dV_{mem}(t)}{dt} &= R.I_{in}(t) - V_{mem}(t)} \end{aligned} \tag{2.19}$$

From that, and setting $I_{in} = 0$ (i.e. no input) and considering $\tau = R.C$ is a constant and a starting voltage $V_{mem}(0)$ the neuron's voltage behavior can be modeled as an exponential curve as can be seen in Equation 2.20.

$$\begin{aligned} \tau.\frac{dV_{mem}(t)}{dt} &= R.I_{in}(t) - V_{mem}(t) \implies \\ \tau.\frac{dV_{mem}(t)}{dt} &= -V_{mem}(t) = \\ e^{\ln(V_{mem}(t))} &= e^{-\frac{t}{\tau}} = \\ \boxed{V_{mem}(t) &= V_{mem}(0).e^{-\frac{t}{\tau}}} \end{aligned} \tag{2.20}$$

Then one can say that: In the absence of an input I_{in}, the membrane potential decays exponentially as illustrated in Figure 15 and implemented in Listing 2.1.

```
def lif(V_mem, dt=1, I_in=0, R=5, C=1):
  tau = R*C
  V_mem = V_mem + (dt/tau)*(-V_mem + I_in*R)
  return V_mem
```

Algoritmo 2.1 – Python implementation of the action potential decaying of a LIF: $I_{in} = 0$

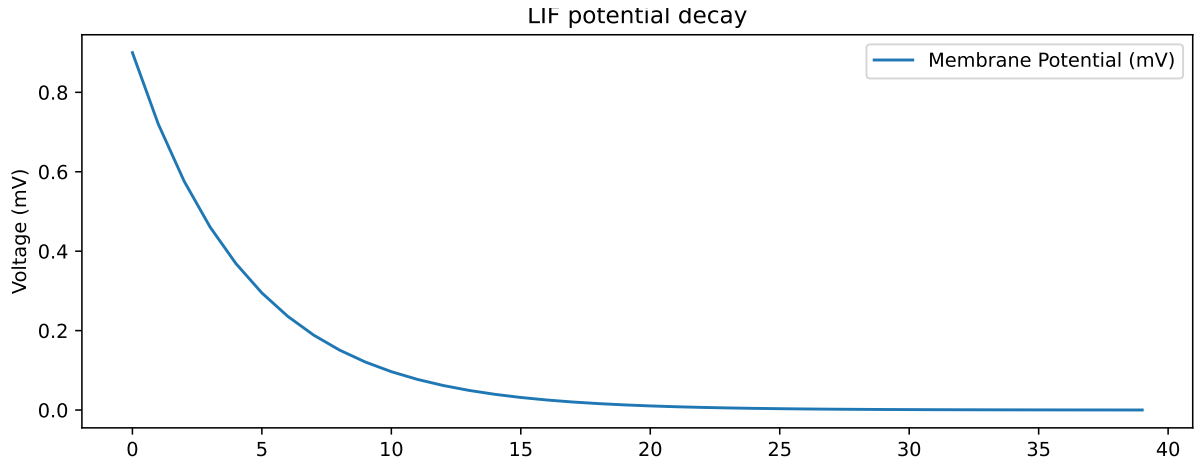


Figura 15 – Membrane potential decaying. Source: The author

With the results from Equation 2.19 it is possible to calculate the action potential increasing as seen in Equation 2.21.

$$
\begin{aligned}
&\tau.\frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t) = \\
&\frac{dV_{mem}(t)}{dt} + \frac{V_{mem}(t)}{\tau} = \frac{R.I_{in}(t)}{\tau} \implies \\
&\text{Integrating factor: } e^{\int \frac{1}{\tau} dt} = e^{\frac{1}{\tau}.t} \implies \\
&(e^{\frac{1}{\tau}.t}.V_{mem}(t))' = \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} = \\
&\int (e^{\frac{1}{\tau}.t}.V_{mem}(t))' = \int \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} dt = \\
&e^{\frac{1}{\tau}.t}.V_{mem}(t) = \int \frac{R.I_{in}(t)}{\tau}.e^{\frac{1}{\tau}.t} dt \therefore \\
&\text{Considering: } V_{mem}(t=0) = 0 \implies \\
&\boxed{V_{mem}(t) = I_{in}(t).R(1 - e^{\frac{1}{\tau}})}
\end{aligned}
\tag{2.21}
$$

Note that when action potentials increases there is still an exponential behavior as seen in Figure 16 and implemented in Listing 2.2.

```
1
2 def lif(V_mem, dt=1, I_in=1, R=5, C=1):
3   tau = R*C
4   V_mem = V_mem + (dt/tau)*(-V_mem + I_in*R)
5   return V_mem
```

Algoritmo 2.2 – Python implementation of the action potential decreasing of a LIF: $I_{in} = 1$

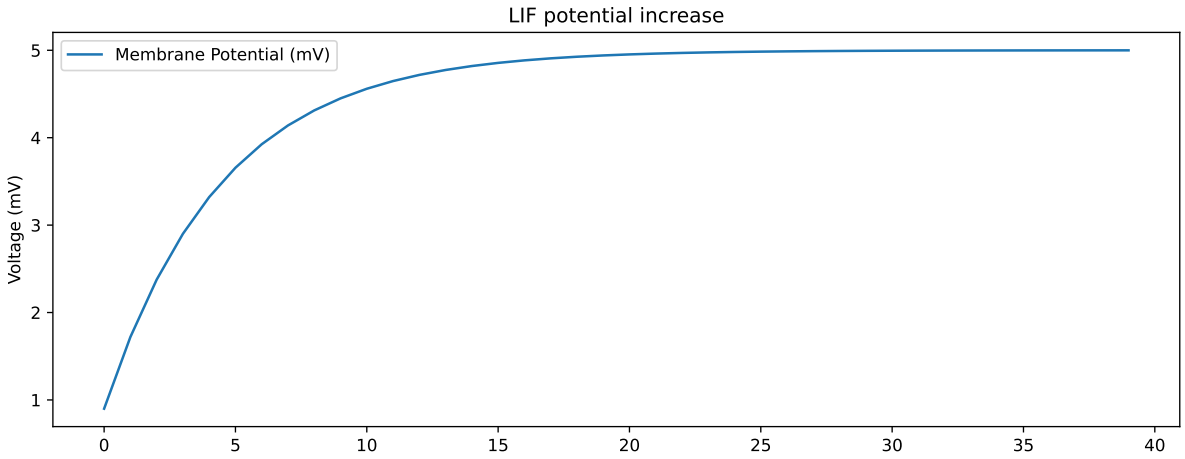


Figura 16 – Membrane potential increasing. Source: The author

Then taking into account some **threshold** which indicates a reset into the neuron potential and two type of resets (to zero and threshold subtraction) finally is possible to model the full LIF behavior as depicted in Figure 17 and implemented in Listing 2.3:

```
1 def lif(V_mem, dt=1, I_in=1, R=5, C=1, V_thresh = 2, reset_zero = True):
2   tau = R*C
```

```
  V_mem = V_mem + (dt/tau)*(-V_mem + I_in*R)
  if V_mem > V_thresh:
    if reset_zero:
      V_mem = 0
    else:
      V_mem = V_mem - V_thresh
  return V_mem
```

Algoritmo 2.3 – Python implementation of the action potential full simulation of a LIF: $I_{in} = 1$, $V_{thresh} = 2$ is threshold

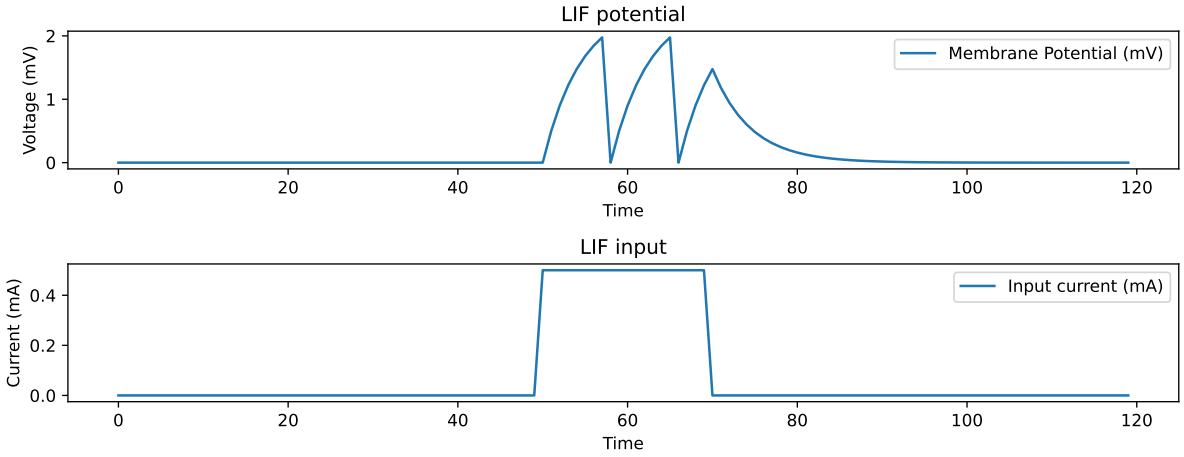


Figura 17 – Plot of the full simulated LIF: A 0.5 mA was provided from time 51 to 70.

2.1.10 Another interpretation of LIF

Starting with the Equation 2.19 and using Forward Euler Method to solve the LIF model:

$$\tau . \frac{dV_{mem}(t)}{dt} = R.I_{in}(t) - V_{mem}(t) \tag{2.22}$$

Solving the derivative in Equation 2.23 results in the membrane potential for any time $t + \Delta t$ in future:

$$\begin{aligned} \tau . \frac{V_{mem}(t+\Delta t) - V_{mem}(t)}{\Delta t} &= R.I_{in}(t) - V_{mem}(t) = \\ V_{mem}(t+\Delta t) - V_{mem}(t) &= \frac{\Delta t}{\tau} . (R.I_{in}(t) - V_{mem}(t)) = \\ \boxed{V_{mem}(t+\Delta t) &= V_{mem}(t) + \frac{\Delta t}{\tau} . (R.I_{in}(t) - V_{mem}(t))} \end{aligned} \tag{2.23}$$

2.1.11 Training

How do SNNs get trained? Well, this is still an open question. A SNN neuron has an activation-function behavior that is more relatable to a **step function**. Therefore, in principle, we can't use gradient descent-based solutions because this kind of function **is not** differentiable (KASABOV, 2019).

But there are some insights out there that may shed some light on this subject: While some *in vivo/in vitro* observations show that brains, in general, learn by strengthening/weakening and adding/removing synapses or even by creating new neurons or other cumbersome methods like RNA packets, there are some more acceptable ones like the ones in the list bellow (KASABOV, 2019):

- **Spike Timing-Dependent Plasticity (STDP)**: If a pre-synaptic neuron fires **before** the post-synaptic one, there is a strengthening in connection, but if the post-synaptic neuron fires before, then there is a weakening.
- **Surrogate Gradient Descent**: Approximates the step function by using another mathematical function, which is differentiable (like a sigmoid), in order to train the network. These approximations are used only in the backward pass, while keeping the step function in the forward pass (KASABOV, 2019).
- **Evolving Algorithms**: Use the selection of the fittest throughout many generations of networks.
- **Reservoir/Dynamic Computing**: **Echo state networks** or **Liquid state machines** respectively.

The one used in the SNN made in this work were **Surrogate Gradient Descent**.

2.1.12 Autoencoders

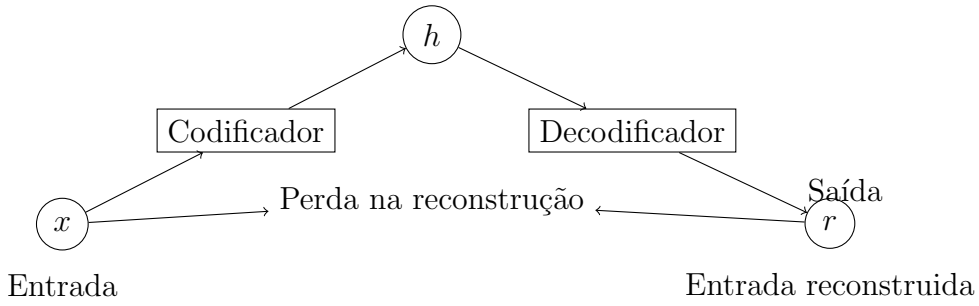
Como ilustrado na Figura 18 *Autoencoders* são redes neurais treinadas para reconstruir seus dados de entrada. Eles consistem em uma função codificadora, denotada como $h = f(x)$, e uma função decodificadora que produz uma reconstrução, denotada como $r = g(h)$. A camada oculta h representa um código ou representação comprimida da entrada (GOODFELLOW; BENGIO; COURVILLE, 2016).

O principal objetivo de um *autoencoder* é aprender uma representação compactada dos dados de entrada na camada oculta e, em seguida, reconstruir os dados de entrada com a maior precisão possível usando o decodificador. No entanto, os *autoencoders* são projetados para serem incapazes de copiar perfeitamente os dados de entrada. Eles

geralmente são limitados de alguma forma para apenas aproximar a entrada e priorizar certos aspectos dos dados.

Autoencoders podem ser treinados usando várias técnicas, como *gradient descent* com *minibatch* ou estocástico (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figura 18 – *Autoencoder*: x é codificado para uma dimensão menor h e, em seguida, é reconstruído em r, tal processo pode implicar ou não em uma perda na reconstrução

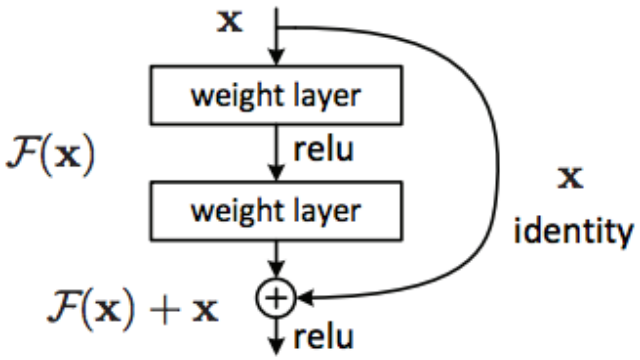


Considerando-se que a reconstrução r seja razoável, isso significa que a região h contém dados suficientes para representar a informação em sua essência, sendo assim, dentro do contexto das redes neurais, autoencoders são ótimos produtores de vetores de características.

2.1.13 Redes neurais residuais (ResNets)

Segundo (HE et al., 2015) a ideia-chave por trás do *ResNets* é a inclusão de conexões de salto como ilustrado na Figura 19, também conhecidas como mapeamentos de identidade, que permitem que a saída de uma camada seja adicionada diretamente à entrada da camada subsequente. Isso contorna as camadas intermediárias e garante que redes mais profundas possam aprender. Outra vantagem no uso de conexões de salto é que essa prática diminui a ocorrência de *vanishing gradients* um problema comum em redes com muitas camadas que pode impossibilitar ou diminuir a níveis impraticáveis o aprendizado da rede.

Figura 19 – Bloco de uma Rede Neural Residual, X é uma função identidade que contorna as camada intermediárias criando "*highway connections*"



Fonte: (HE et al., 2015)

3 Abordagem proposta

3.1 A Base de sinais

3.1.1 Coleta dos sinais

3.1.2 Organização da base de sinais

3.2 Estrutura da estratégia proposta

3.3 Procedimentos

3.3.1 Procedimento 01

4 Testes e Resultados

4.1 Procedimento 01

5 Conclusões e Trabalhos Futuros

Referências

ADDISON, P. S.; WALKER, J.; GUIDO, R. C. Time–frequency analysis of biosignals. *IEEE Engineering in Medicine and Biology Magazine*, v. 28, n. 5, p. 14–29, Sep 2009.

BERANEK, L. L. *Acoustic Measurements*. [S.l.]: J. Wiley, 1949.

BIANCHI, G. *Electronic Filter Simulation & Design*. [S.l.]: McGraw-Hill Education, 2007. ISBN 9780071712620.

BOSI, M.; GOLDBERG, R. E. *Introduction to digital audio coding and standards*. [S.l.]: Springer Science & Business Media, 2002. v. 721.

BUTTERWORTH, S. On the theory of filters amplifiers. 1930.

COSTA, N. C. d.; ABE, J. M. Paraconsistência em informática e inteligência artificial. *Estudos Avançados*, scielo, v. 14, p. 161 – 174, 08 2000.

COSTA, N. da; BÉZIAU, J.; BUENO, O. *Elementos de teoria paraconsistente de conjuntos*. Centro de Lógica, Epistemologia e História da Ciência, UNICAMP, 1998. (Coleção CLE). Disponível em: ⟨https://books.google.com.br/books?id=MGCKGwAACAAJ⟩.

DAUBECHIES, I. *Ten Lectures on Wavelets*. [S.l.]: Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1992. (CBMS-NSF Regional Conference Series in Applied Mathematics).

ESHRAGHIAN, J. K. et al. Training spiking neural networks using lessons from deep learning. *Proceedings of the IEEE*, v. 111, n. 9, p. 1016–1054, 2023.

FREITAS, S. Avaliação acústica e áudio percetiva na caracterização da voz humana. Faculdade de Engenharia da Universidade do Porto, 2013.

GERSTNER, W. et al. *Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition*. Cambridge University Press, 2014. ISBN 9781107060838. Disponível em: ⟨https://neuronaldynamics.epfl.ch/online/Ch2.S2.html⟩.

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

GOODMAN, D. et al. *Spiking Neural Network Models in Neuroscience - Cosyne Tutorial 2022*. Zenodo, 2022. Disponível em: ⟨https://doi.org/10.5281/zenodo.7044500⟩.

GUIDO, R. C. Practical and useful tips on discrete wavelet transforms [sp tips tricks]. *IEEE Signal Processing Magazine*, v. 32, n. 3, p. 162–166, May 2015.

GUIDO, R. C. Paraconsistent feature engineering [lecture notes]. *IEEE Signal Processing Magazine*, v. 36, n. 1, p. 154–158, Jan 2019.

HAYKIN, S. *Redes Neurais: Princípios e Prática*. [S.l.]: Bookman Editora, 2001. ISBN 9788577800865.

HAYKIN, S.; MOHER, M. *Sistemas de Comunicação-5*. [S.l.]: Bookman Editora, 2011.

HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: ⟨http://arxiv.org/abs/1512.03385⟩.

Jalaly Bidgoly, A.; Jalaly Bidgoly, H.; AREZOUMAND, Z. A survey on methods and challenges in eeg based authentication. *Computers and Security*, v. 93, p. 101788, 2020. ISSN 0167-4048. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0167404820300730⟩.

JENSEN, A.; COUR-HARBO, A. la. *Ripples in Mathematics*. [S.l.]: Springer Berlin Heidelberg, 2001.

JOLLIFFE, I. *Principal Component Analysis*. Springer, 2002. (Springer Series in Statistics). ISBN 9780387954424. Disponível em: ⟨https://books.google.com.br/books?id=_olByCrhjwIC⟩.

JONES, I. S.; KORDING, K. P. *Can Single Neurons Solve MNIST? The Computational Power of Biological Dendritic Trees*. 2020. Disponível em: ⟨https://arxiv.org/abs/2009.01269⟩.

KASABOV, N. K. *Time-space, spiking neural networks and brain-inspired artificial intelligence*. [S.l.]: Springer, 2019.

KREMER, R. L.; GOMES, M. d. C. A eficiência do disfarce em vozes femininas: uma análise da frequência fundamental. *ReVEL*, v. 12, p. 23, 2014.

POLIKAR, R. et al. *The wavelet tutorial*. 1996.

RASHID, T. *Make Your Own Neural Network: A Gentle Journey Through the Mathematics of Neural Networks, and Making Your Own Using the Python Computer Language*. [S.l.]: CreateSpace Independent Publishing, 2016. ISBN 9781530826605.

SALOMON, D.; MOTTA, G.; BRYANT, D. *Data Compression: The Complete Reference*. Springer London, 2007. (Molecular biology intelligence unit). ISBN 9781846286032. Disponível em: ⟨https://books.google.com.br/books?id=ujnQogzx_2EC⟩.

SANEI, S.; CHAMBERS, J. A. *EEG signal processing and machine learning*. [S.l.]: John Wiley & Sons, 2021.

VALENÇA, E. H. O. et al. Análise acústica dos formantes em indivíduos com deficiência isolada do hormônio do crescimento. Universidade Federal de Sergipe, 2014.

VICENTE, E. *Sistema 10-20 para localizar alvos terapêuticos em EMT*. 2023. Disponível em: ⟨https://www.kandel.com.br/post/como-localizar-os-pontos-de-estimulacao-para-estimulacao-magnetica-transcraniana⟩.

WERTZNER, H. F.; SCHREIBER, S.; AMARO, L. Análise da freqüência fundamental, jitter, shimmer e intensidade vocal em crianças com transtorno fonológico. *Revista Brasileira de Otorrinolaringologia*, SciELO Brasil, v. 71, p. 582–588, 2005.

ZWICKER, E. Subdivision of the audible frequency range into critical bands (frequenzgruppen). *The Journal of the Acoustical Society of America*, v. 33, n. 2, p. 248–248, 1961.

APÊNDICE A – Como fazer uma rede neural

A.1 Entendo aproximadores de função

- Analisar o contexto do problema: O mesmo é linear, não linear (RASHID, 2016).
- A partir do item anterior, definir qual a função da ativação será usada (RASHID, 2016).

É importante também, dependendo do tipo de problema, saber escolher uma função de cálculo de erro, essa função, pode ser uma função quadrática, linear ou outra dependendo da importância que o erro tem na solução do problema. As funções de cálculo de erro mais usadas são mostradas abaixo:

- $erro = valorDesejado - valorCalculado$
-

Calculado o erro é necessário agora, de alguma forma, fazer com que esse erro seja corrigido utilizando uma abordagem iterativa que se aproxima cada vez mais do valor desejado usando pequenos passos. Considerando que Δ represente uma pequena variação em um valor *peso*, e sendo *peso* um dos parâmetros da função de ativação de um aproximador ($y = peso.x$ para uma função de ativação linear) então temos que a correção do valor deve ser dado como na equação abaixo:

$$valorDesejado = (peso + \Delta peso).entrada \quad , \tag{A.1}$$

(RASHID, 2016)

Então, sabendo que o $valorDesejado = peso + \Delta peso.entrada$ e que $erro = valorDesejado - valorCalculado$ se pode concluir que:

$$\Delta peso = \frac{erro}{entrada} \quad , \tag{A.2}$$

No tocante as funções de ativação se pode imaginar quantas forem necessárias, no entanto, algumas se destacam dentro do campo das redes neurais:

- *step function*:

$$y = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases} \tag{A.3}$$

- *sigmoid function*:

$$y = \frac{1}{1 + e^{-x}} \tag{A.4}$$

No entanto, existe um problema quanto a esta abordagem, a função de ativação que recebe o parâmetro *peso* Se adaptará ao último exemplo mostrado a ela criando uma situação conhecida como "overfitting", invalidando assim todos os os outros exemplos anteriormente usados, portanto, como se pode notar, é necessário a criação de um elemento que impeça esse ajustamento extremo, chamado de taxa de aprendizado:

$$\Delta peso = taxaDeAprendizado.\left(\frac{erro}{entrada}\right) \quad , \tag{A.5}$$

Usualmente a taxa de aprendizado é um valor suficientemente pequeno cujo o objetivo é garantir que o *ovefitting* não aconteça mas, suficientemente grande para que a rede aprenda em um tempo razoável. Sendo $0,1$ um dos valores normalmente usados.

Já outros autores preferem usar uma função de erro "acumulada", desta forma, o erro é calculado em lote segundo várias entradas e saídas processadas pela rede de forma que se compute o erro acumulado segundo a expressão A.6 onde E é o erro M é o tamanho do vetor de saída da rede, N é a quantidade de amostras fornecidas, y_{ik} é o vetor de resultante da rede e t_{ik} é o vetor esperado.

$$E = \frac{1}{2MN}.\sum_{k}^{N}\sum_{i}^{M}(y_{ik} - t_{ik})^2 \quad , \tag{A.6}$$

A.2 Junção dos aproximadores: Redes neurais

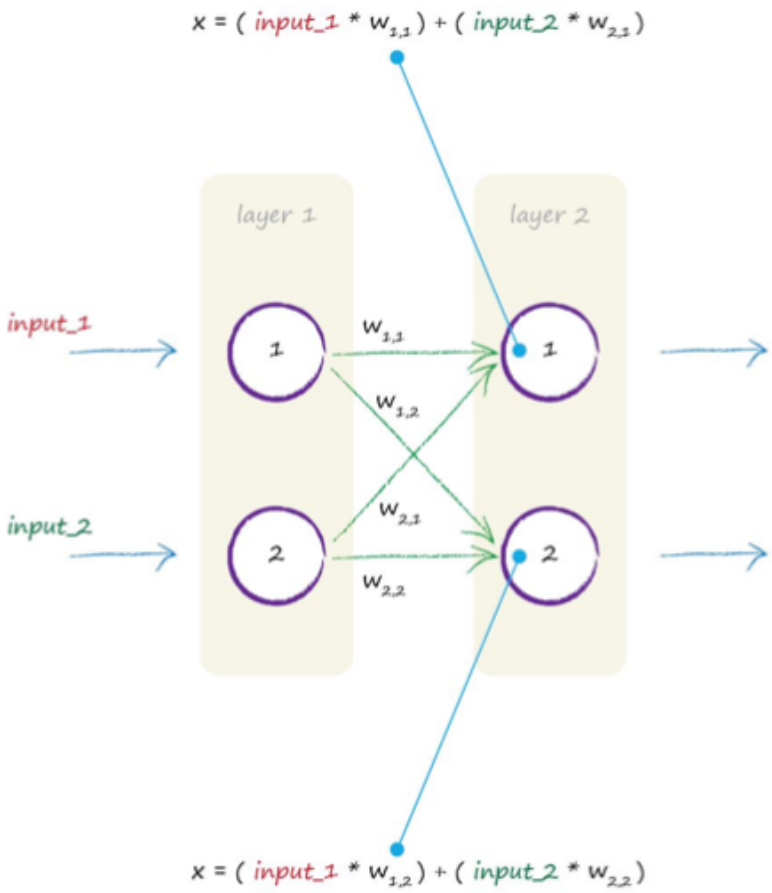
Já que a sessão anterior definiu o que é um aproximador de funções, nesta sessão, usando as definições já vistas, será apresentada a definição de uma rede neural que é a junção de vários aproximadores que, dessa forma, conseguem delimitar espaços de resultados mais complexos além daqueles que podem ser separados por apenas uma função de ativação.

Um dos exemplos clássicos que necessitam dessa abordagem mais complexa é o exemplo da porta lógica *xor*. Quando se tenta usar apenas um aproximador de funções para conseguir os resultados dessa porta é possível notar que o mesmo não é capaz de criar um espaço de resultados suficientemente complexo afim de separar os pontos fornecidos, sendo assim, Torna-se necessário o uso de múltiplos aproximadores (ou, nesse caso, 2).

Pode-se considerar tal junção como uma a rede neural. É possível perceber que a mesma passa ter múltiplas entradas tornando-se necessário, antes que a função de ativação seja aplicada, que a soma dessas entradas seja feita. A este procedimento se dá o nome de *feedforward* (HAYKIN, 2001):

Via de regra uma rede neural tem, no mínimo, duas camadas: A camada de entrada que é usada apenas para representar os valores que serão processados e a camada de saída, cujo papel é fazer a soma ponderada das entradas passando então o resultado dessa operação para uma função de ativação que finalmente gerará as saídas de nossa rede neural como ilustrado na figura 20.

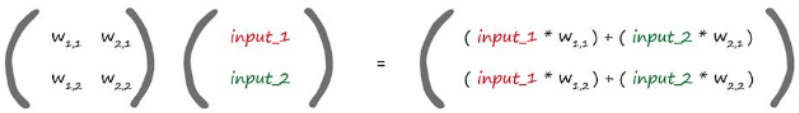
Figura 20 – Uma rede neural simples Fonte: O autor



A.3 Uma rede neural simples

Como visto na figura 20 da sessão anterior uma rede neural conecta suas camadas usando pesos que servirão como moderadores das entradas da próxima camada com o fim desse evitar o *overfitting*. É possível representar uma rede neural de várias formas, no entanto, como ilustrado na figura 21, a forma mais eficiente computacionalmente será a matricial, usando esta abordagem será possível representar facilmente as camadas e os respectivos pesos que as ligam, melhorando também o desempenho computacional já que tais operações são altamente paralelizáveis.

Figura 21 – Representação de uma rede neural usando matrizes



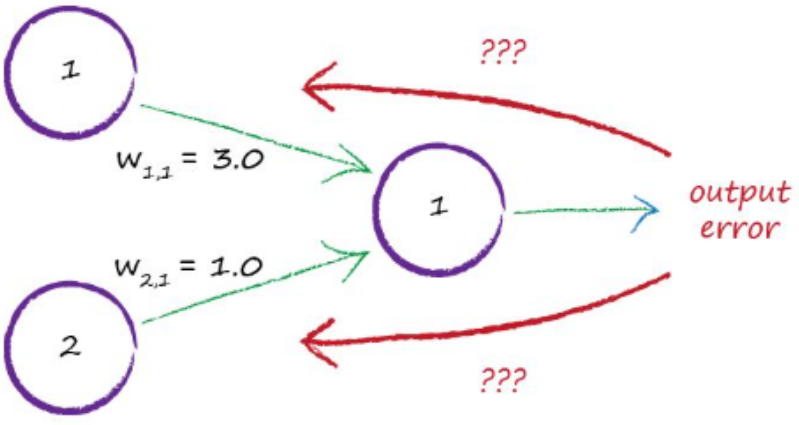
A.4 Redes neurais multicamadas

A depender da complexidade do problema apenas duas camadas não são suficientes para que este aproximados universal de funções, mais conhecido como rede neural, consiga obter resultados satisfatórios. Assim, se torna necessário adição de uma ou mais camadas que servirão para aumentar a complexidade do espaço das soluções de nossa rede.

Então, considerando o contexto, as tais camadas que devem ser adicionadas serão chamadas de ocultas ou *hidden layers*(RASHID, 2016).

Em redes neurais multicamadas surge um novo problema: Como atualizar os valores dos pesos entre as camadas já que o erro produzido depende das várias entradas da camada anterior? Como mostrado na figura 20 o valor da próxima camada depende da soma ponderada pelos pesos dos valores da anterior sendo que tal soma deve passar por uma função de ativação escolhida, portanto, após o cálculo do erro na resposta da rede é preciso recalcular os pesos de forma a distribuir esses erros usando um algoritmo ao qual daremos o nome de retro-propagação ou *backpropagation* (HAYKIN, 2001). É importante frisar que este é apenas um dos vários métodos de treinamento da rede neural, é possível, por exemplo, usar uma abordagem bio-inspirada e/ou evolutiva.

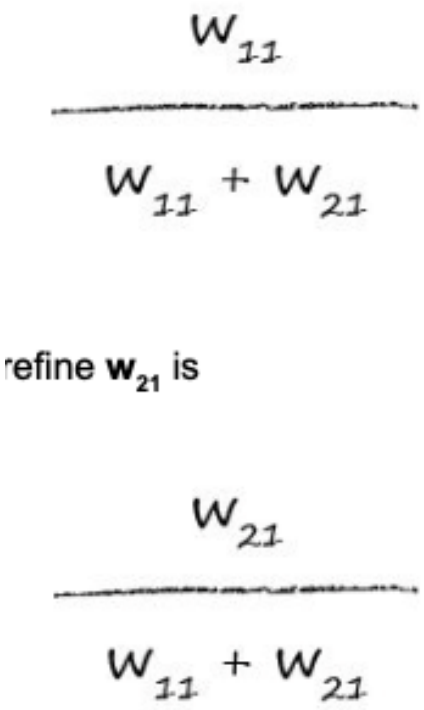
Figura 22 – Retro-propagação



A.5 Retro-propagação

Primeiramente é importante notar que os pesos associados a cada item da camada (neurônio) Contribuem com o erro resultante de forma proporcional aos seus respectivos valores, sendo assim, é necessário, ao fazer a retro-propagação, que esses erros sejam distribuídos proporcionalmente como indicado na figura 23.

Figura 23 – Parcela da distribuição do erro na retro-propagação para o $erro_1$, o mesmo deve ser feito para todos os outros erros.



Calculada a parcela da participação de cada peso nos erros produzidos possibilta calcular o erro resultante na camada oculta, usando o mesmo raciocínio de cálculo da participação no erro, é possível continuar com algoritmo de retro-propagação. Assim, o cálculo do erro da camada oculta se dá como mostrado na equação A.7.

$$erro_{oculto_1} = \begin{cases} erro_{resultado_1} * \dfrac{peso_{1,1}}{peso_{1,1} + peso_{2,1} + ... + peso_{n,1}} + \\ erro_{resultado_2} * \dfrac{peso_{1,2}}{peso_{1,2} + peso_{2,2} + ... + peso_{n,2}} + \\ \vdots \\ + erro_{resultado_k} * \dfrac{peso_{1,k}}{peso_{1,k} + peso_{2,k} + ... + peso_{n,k}} \end{cases} \tag{A.7}$$

Onde, em relação ao neurônio atual da camada oculta($oculto_1$), n representa a

quantidade de pesos vinculados e k o número de neurônios na camada de saída . As figuras 24 e 25 mostram em mais detalhes o funcionamento do algoritmo.

Figura 24 – Retro-propagação da camada de saída para a oculta

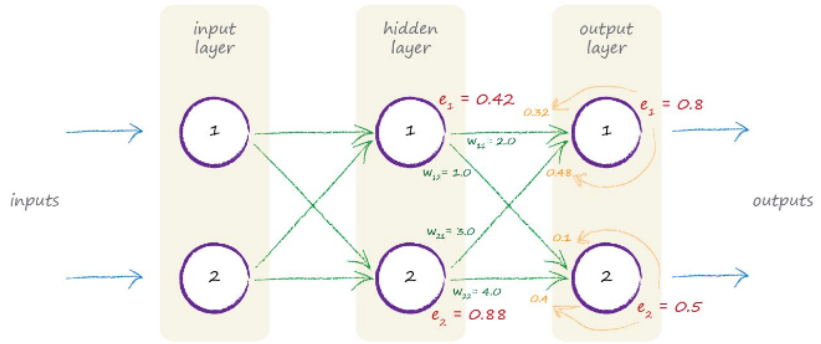
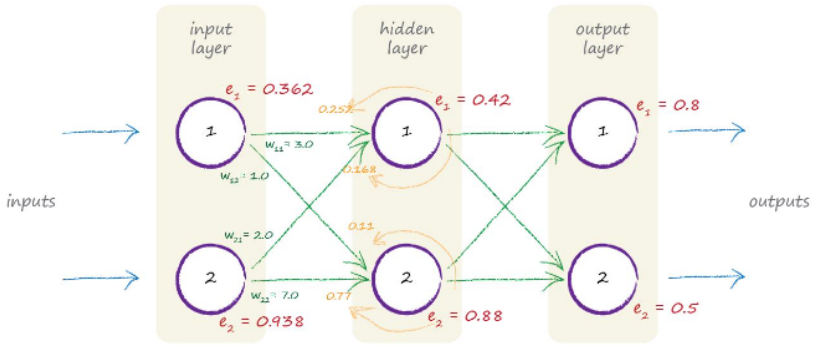


Figura 25 – Retro-propagação da camada oculta para a entrada



A.6 Retro-propagação II

Sendo uma rede neural um aproximador universal de funções, o algoritmo de retro-propagação será feito de forma que a taxa de variação dessa função (derivada) seja levado em conta. Especificamente, a função em questão, é o que se convencionou chamar função de custo ou de erro, que é um método para calcular o erro total que a rede produziu dada uma certa entrada, sendo assim, a função de custo também engloba a função de ativação ja que esta contribui para o erro, esse fato será importante mais a frente quando a derivada da mesma será necessária graças à regra da cadeia.

Para melhor entender como a derivada da função de erro para uma rede neural deve ser tomada, é necessário primeiro entender o que é o vetor gradiente: O vetor gradiente é aquele que aponta para o maior crescimento da função dadas as derivadas parciais dos respectivos parâmetros da mesma. No caso de uma rede neural os parâmetros são os pesos que interliga cada neurônio que a compõem.

Portanto se f é uma função A.8 que contém os parâmetros x_1, x_2, x_3, por exemplo. Então o vetor gradiente da mesma será compostos pelas derivadas parciais de cada parâmetro A.9.

$$f(x_1, x_2, x_3) \tag{A.8}$$

$$\nabla f(x_1, x_2, x_3) = \begin{bmatrix} \frac{\delta f}{\delta x_1} \\ \\ \frac{\delta f}{\delta x_2} \\ \\ \frac{\delta f}{\delta x_3} \end{bmatrix} \tag{A.9}$$

Como já foi dito, o gradiente da função aponta para a direção dentro do domínio da função onde a mesma cresce mais rápido, no entanto, a intenção não é maximizar a função e sim minimiza-las, portanto, para que seja possível encontrar um mínimo da função, considera-se o valor oposto (negativo) do gradiente. Sendo assim, considerando que X_t denota o valor atual do parâmetro X (que é um vetor) e X_{t+1} corresponde ao próximo valor que será fornecido a função f. O cálculo de X_{t+1} se dará como mostrado na equação A.10 .

$$X_{t+1} = X_t - \eta.\nabla f(X_t) \tag{A.10}$$

Onde η é um coeficiente escalar **bastante pequeno** conhecido como **coeficiente de aprendizado** cujo valor geralmente está abaixo de 1 como 0.1, 0,3, etc.

Ou, de forma mais explícita, combinando as equações A.9 e A.10 se tem a expressão A.11.

$$X_{t+1} = \begin{bmatrix} \frac{\delta f}{\delta x[0]_t} \\ \\ \frac{\delta f}{\delta x[1]_t} \\ \vdots \\ \frac{\delta f}{\delta x[n]_t} \end{bmatrix} - \eta. \begin{bmatrix} \frac{\delta f}{\delta x[0]_t} \\ \\ \frac{\delta f}{\delta x[1]_t} \\ \vdots \\ \frac{\delta f}{\delta x[n]_t} \end{bmatrix} \tag{A.11}$$

O conteúdo acima apresentado ajudará no algoritmo de retro-propagação que, como já foi dito, considerará os pesos das sinapses da rede neural. É importante frisar que as

entradas da rede neural e as respectivas saídas esperadas também serão importantes, já que, o valor da função de custo só pode ser calculado na existência desses dois valores.

Considere a representação de uma rede neural.

Figura 26 – Rede neural de três camadas escondidas, os pesos representados pelas variáveis w_1, w_2, w_3 na figura 26 são alguns dos parâmetros da rede que devem ser ajustados para que hajam resultados satisfatórios.

Considere para fins de exemplo o vetor de entrada A.12, o vetor de saída A.13 e o vetor da saída desejada A.14.

$$X = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix} \tag{A.12}$$

$$O = \begin{bmatrix} -1 \\ 3 \end{bmatrix} \tag{A.13}$$

$$T = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{A.14}$$

O vetor de entrada X será processado pela rede produzindo um vetor de saída O, A função de custo usará os valores de O e T para calcular o custo total da rede para, dessa forma, ajustar seus parâmetros. A função de custo de uma rede pode ser das mais variadas, no entanto, para fins de entendimento usaremos o erro quadrática médio que, além de ser uma boa função de custo, se mostrará conveniente quando houver a necessidade de derivá-la.

Para começar calcula-se o erro que é a diferença entre a saída desejada e a saída obtida pela rede como mostrado na equação A.15.

$$E0 = O - T \tag{A.15}$$

Resultando nos valores:

$$E0 = \begin{bmatrix} -1 \\ 3 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ 3 \end{bmatrix} \tag{A.16}$$

Calculado o erro $E0$, passa-se ao cálculo do custo E aplicando-se a soma dos quadrados de cada item do vetor de erros, dividindo o resultado por dois, como já foi dito essa divisão é apenas uma conveniência pois facilitará os cálculos no momento da derivação, n é a quantidade de elementos no vetor de saída.

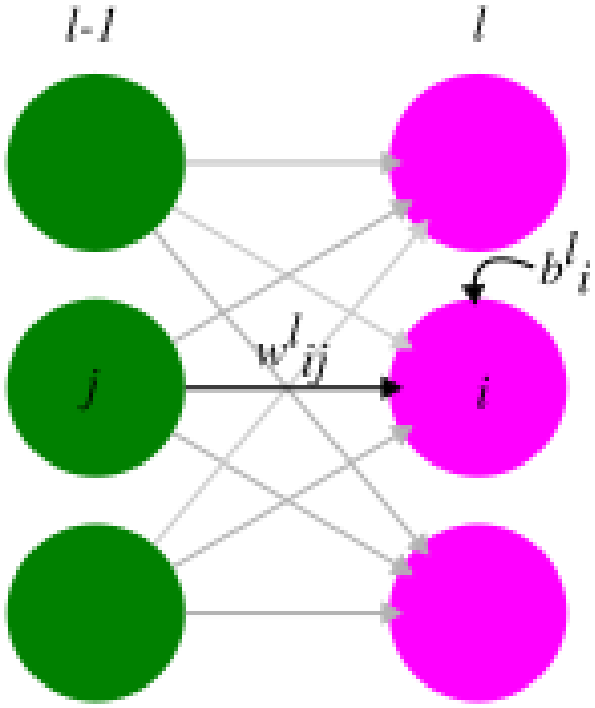
$$E = \frac{1}{2.n}.\sum_{i=0}^{n}(E_i)^2 = \frac{1}{2n}.(-2^2 + 3^2) = \frac{1}{2.2}.(4 + 9) = \frac{1}{4}.13 = \frac{13}{4} \tag{A.17}$$

Agora que o processo de cálculo de custo da rede foi explicado é necessário ter uma visão geral de como a rede será treinada:

- Preparar as entradas e as respectivas saídas esperadas para rede neural
- A partir do custo calculado calcular o gradiente dessa função de custo
- Ajustar os pesos e os *bias* da rede em direção oposta ao gradiente calculado
- Repetir o processo até que uma condição de parada seja encontrada, muita das vezes essa será o custo máximo.

Tendo em mente essa visão geral, considere agora duas camadas $l-1$ e l de uma rede qualquer como mostrado na figura 27.

Figura 27 – Duas camadas quaisquer de uma rede neural totalmente ligada, o neurônio j pertence a camada $l-1$ e um neurônio i pertence a camada l, dessa forma, o peso (ou peso sináptico) pertencente a camada l que liga o neurônio j ao neurônio i é escrito como w^l_{ij}



Representados como mostrado na figura 27 e considerando W^l como a representação de todos os pesos sinápticos pertencentes a camada l essa notação proporciona uma representação matricial descrita na equação A.18 onde i é a linha e j a coluna da matriz. De outra forma: Cada linha i representa o correspondente neurônio da camada l e cada coluna representa o correspondente neurônio da camada $l-1$ e, as intersecções, representam os pesos sinápticos que ligam um neurônimo ao outro. Os *bias* B^l da camada l são representados em A.19.

$$W^l = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ \cdots & w^l_{ij} & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{A.18}$$

$$B^l = \begin{bmatrix} \vdots \\ b^l_i \\ \vdots \\ \vdots \end{bmatrix} \tag{A.19}$$

Para que a rede neural tenha seus pesos sináptica os atualizados é necessário que a operação de *feedforward* seja realizada afim de poderem ser calculados os erros da rede e consequentemente o seu custo.

Figura 28 – Representação do *feedforward*: S_i^l é a soma das saídas vindas da camada $l-1$ ponderadas pelos respectivos pesos w_{ij}^l mais os *bias* B^l da camada l, Z_i^l é um valor escalar resultante da aplicação de uma função de ativação sobre S_i^l

A função de ativação, também conhecida como função de transferência, citada na figura 28 é representada na equação A.20.

$$Z_i^l = \sigma(S_i^l) \tag{A.20}$$

APÊNDICE B – Paralelismo

APÊNDICE C – Medidas dos tempos de execução do software

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de São José do Rio Preto

TERMO DE REPRODUÇÃO XEROGRÁFICA

Autorizo a reprodução xerográfica do presente Trabalho de Conclusão, na íntegra ou em partes, para fins de pesquisa.

São José do Rio Preto, 06 de Agosto de 2022

André Furlan