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

Lista de ilustrações

Lista de tabelas

Sumário

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: O autor

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

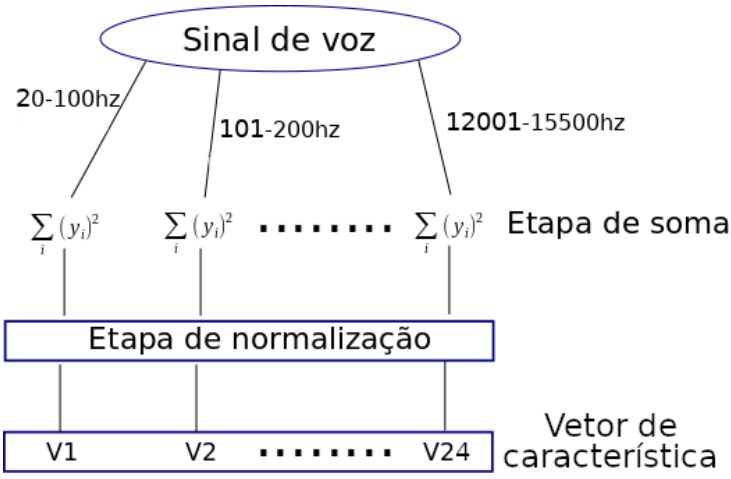
$$E = \sum_{i=0}^{M-1} (s_i)^2 \qquad . \tag{2.1}$$

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: O autor

2.1.3.2 A escala MEL

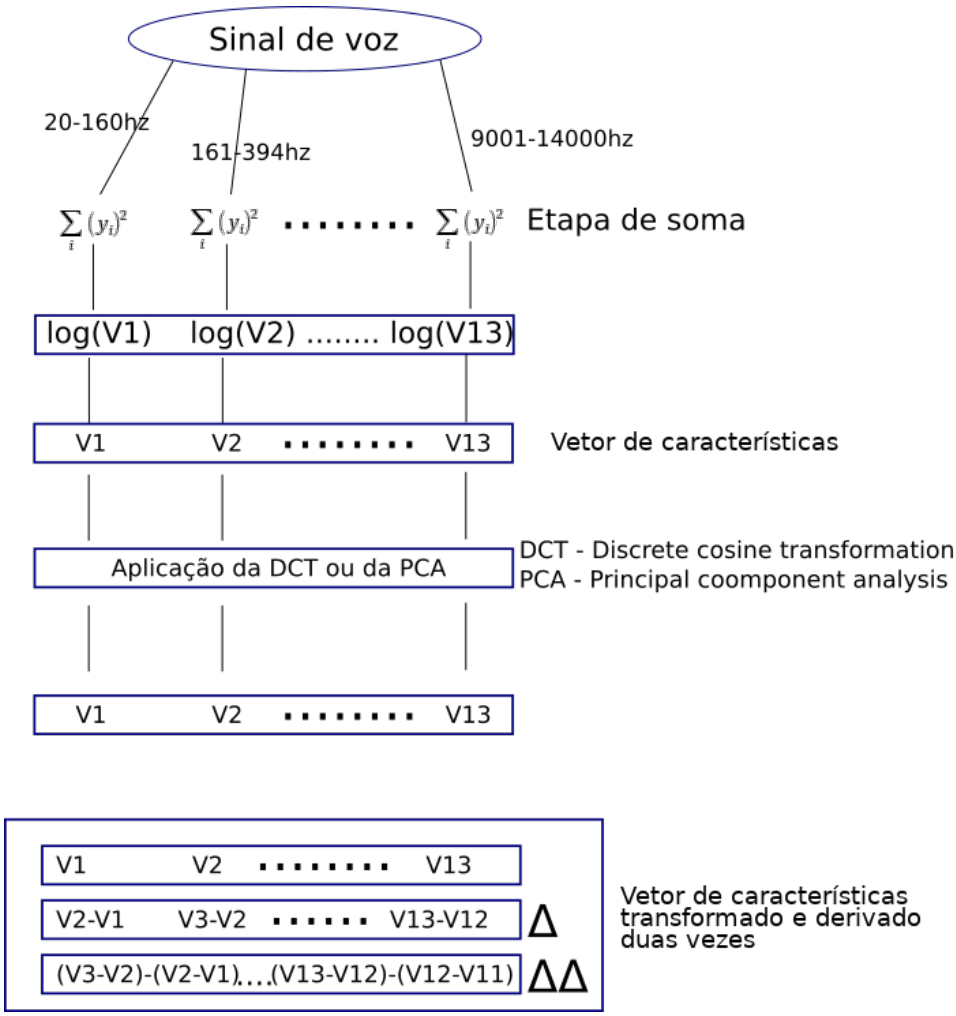
Escala Mel, advinda do termo *melody*, é uma adaptação da escala Bark para sinais de voz. Dentre as várias implementações de bandas críticas a escolhida foi a implementação que contém os valores em Hz: **20, 160, 394, 670, 1000, 1420, 1900, 2450, 3120, 4000, 5100, 6600, 9000, 14000**.

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: O autor

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,

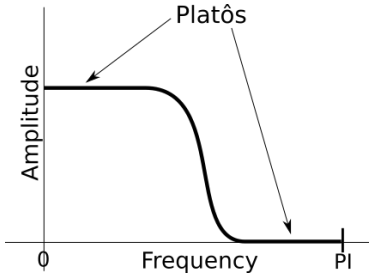
2009). Particularmente, as *wavelets* proporcionam a análise do sinal de forma detalhada tanto no espectro de baixa frequência quanto no de alta contando com diferentes funções-base não periódicas diferentemente da tradicional transformada de Fourrier que utilizam somente as bases periódicas senoidal 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

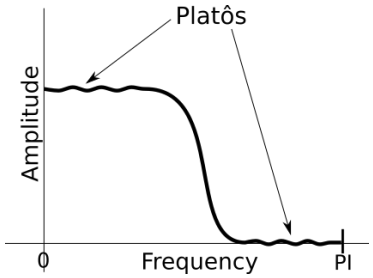


Fonte: O autor

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;

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



Fonte: O autor

- 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 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 Equação 2.2, Equação 2.3 e Equação 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 Tabela 2, Tabela 3 e Tabela 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 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 Tabela 3 e Tabela 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 subbandas 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:

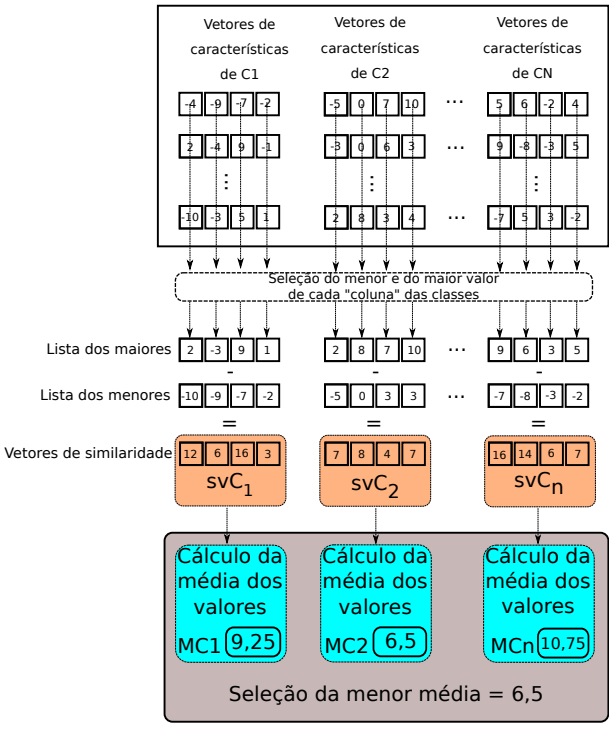
- a menor similaridade intraclasse, α.
- a razão de sobreposição interclasse, β.

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.



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:

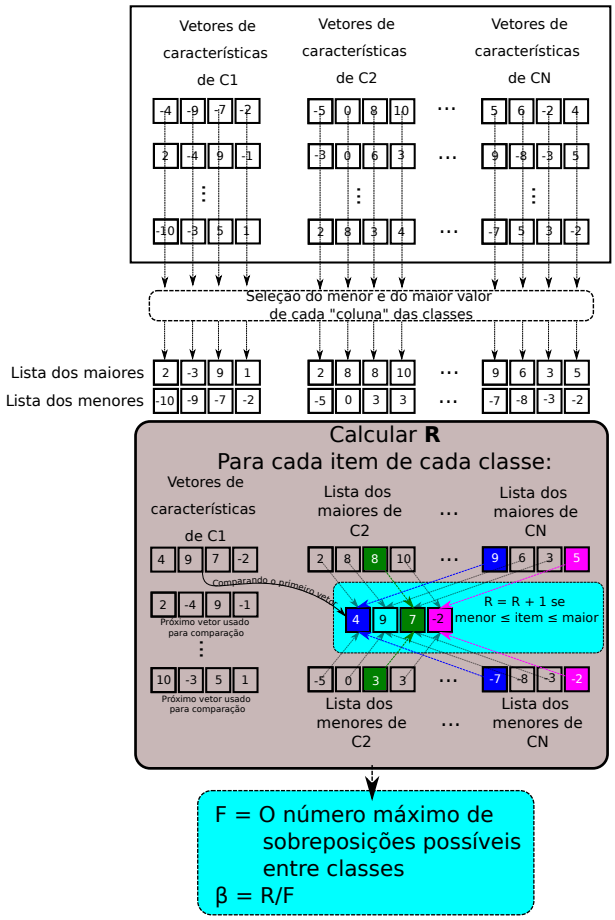
$$G_1 = \alpha - \beta \quad , \tag{2.7}$$

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

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

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.



Adaptado de (GUIDO, 2019)

($G_2 = 1$). 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:

- (-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:

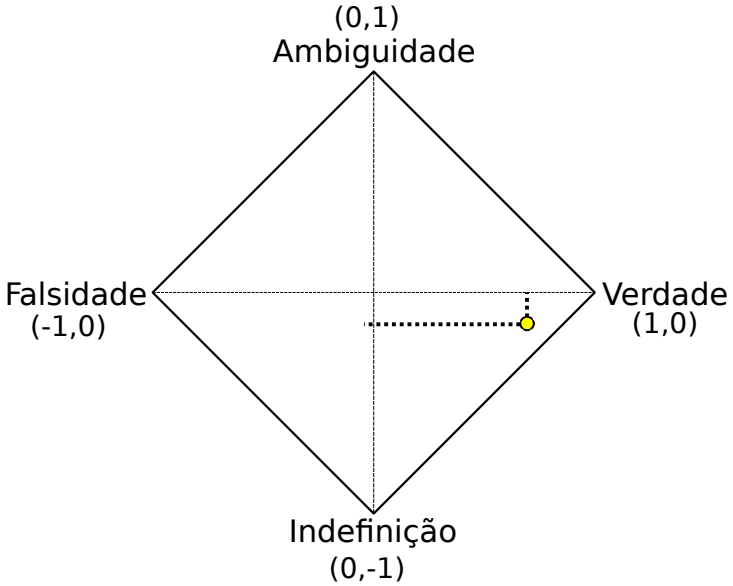
$$D_{-1,0} = \sqrt{(G_1 + 1)^2 + (G_2)^2} \quad , \tag{2.9}$$

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

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

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

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)



Adaptado de (GUIDO, 2019)

Na prática, ou seja, para fins de classificação, geralmente considera-se a distância em relação ao ponto *"(1,0) → 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 Interfaces Humano-Máquina e EEG

Entre os métodos de Interface Cérebro-Computador (BCI), o Eletroencefalograma (EEG) se destaca como o sistema mais econômico e simples de implementar. No entanto, ele possui algumas peculiaridades, como alta sensibilidade a interferência eletromagnética e dificuldade em capturar o sinal devido a posicionamentos subótimos dos eletrodos

no couro cabeludo. Portanto, um dos aspectos fundamentais que qualquer sistema de processamento de EEG deve ter é a tolerância ao ruído (BIDGOLY; BIDGOLY; AREZOUMAND, 2020).

Os eletrodos *úmidos* são colocados usando gel condutivo e são menos propensos a artefatos provocados por movimentos, que são interferências eletromagnéticas causadas, por exemplo, ao piscar dos olhos. Os eletrodos *secos* não precisam do gel, mas são mais sensíveis a tais artefatos.

O EEG registra as atividades elétricas do cérebro, geralmente colocando eletrodos ao longo da superfície do couro cabeludo. Essas atividades elétricas resultam de fluxos de corrente iônica induzidos pela ativação sináptica sincronizada dos neurônios do cérebro. Elas se manifestam como flutuações de voltagem rítmicas com amplitude variando de 5 a $100\mu V$ e frequência entre 0,5 e 40 Hz (BIDGOLY; BIDGOLY; AREZOUMAND, 2020). As bandas de frequência operacionais no cérebro são as seguintes (SANEI; CHAMBERS, 2021):

- **Delta (1–4Hz)**: A onda mais lenta e geralmente a de maior amplitude. A banda Delta é observada em bebês e durante o sono profundo em adultos.

- **Theta (4–8Hz)**: Observada em crianças, adultos sonolentos e durante a recordação de memórias. A amplitude da onda Theta é tipicamente inferior a $100\mu V$.

- **Alpha (8–12Hz)**: Geralmente a banda de frequência dominante, aparecendo durante a consciência relaxada ou quando os olhos estão fechados. A atenção focada ou a relaxamento com os olhos abertos reduzem a amplitude da banda Alpha. Essas ondas tem amplitudes normalmente inferiores a 50 μV.

- **Beta (12–25Hz)**: Associada ao pensamento, concentração ativa e atenção focada. A amplitude da onda Beta é normalmente inferior a 30 μV.

- **Gamma (acima de 25Hz)**: Observada durante o processamento sensorial múltiplo. Os padrões Gamma têm a menor amplitude.

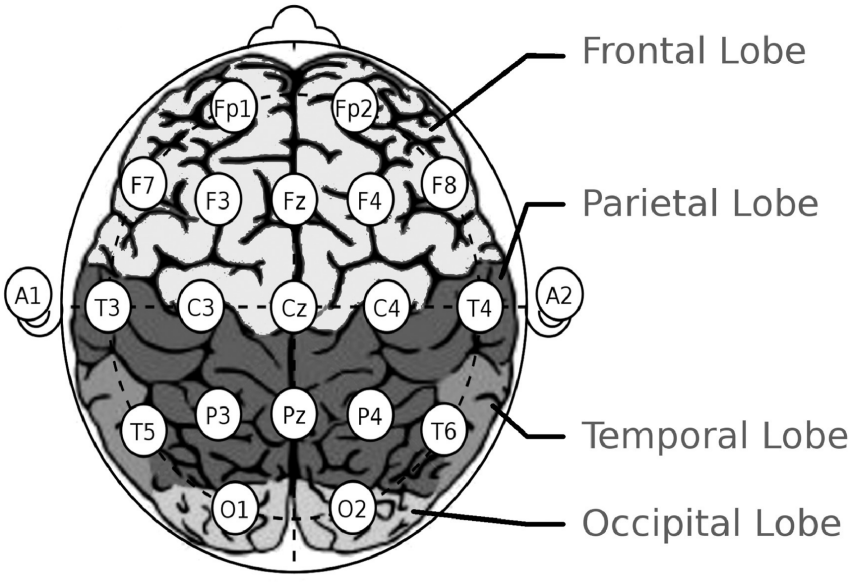
De acordo com (BIDGOLY; BIDGOLY; AREZOUMAND, 2020), para a maioria das tarefas realizadas pelo cérebro, existem regiões associadas a elas, conforme visto na Tabela Tabela 5.

2.1.7 Sistema 10-20 e as áreas do cérebro

Tabela 5 – Tarefas cerebrais e suas regiões correspondentes (VICENTE, 2023), (BIDGOLY; BIDGOLY; AREZOUMAND, 2020)

Região	Canais	Tarefas
Lobo frontal	Fp1, Fp2, Fpz, Pz, F3, F7, F4, F8	Memória, concentração, emoções.
Lobo parietal	P3, P4, Pz	Resolução de problemas, atenção, sentido do tato.
Lobo temporal	T3, T5, T4, T6	Memória, reconhecimento de faces, audição, palavras e percepção social.
Lobo occipital	O1, O2, Oz	Leitura, visão.
Cerebelo		Controle motor, equilíbrio.
Córtex senso-motor	C3, C4, Cz	Atenção, processamento mental, controle motor fino, integração sensorial.

Figura 9 – Posicionamento dos eletrodos de acordo com o padrão 10-20 e lobos cerebrais. Números ímpares são atribuídos aos eletrodos no hemisfério esquerdo, e números pares são atribuídos aos eletrodos no hemisfério direito.



Fonte: (BIDGOLY; BIDGOLY; AREZOUMAND, 2020)

2.1.8 Redes Neurais de Pulso (Spiking Neural Networks)

Redes Neurais (RNs), conforme definido aqui como *uma rede multicamadas, totalmente conectada, com ou sem camadas recorrentes ou convolucionais*, exigem que todos os neurônios sejam ativados tanto na fase de *feed-forward* quanto na de *backpropagation*. Isso

implica que cada unidade na rede deve processar alguns dados, resultando em consumo de energia (ESHRAGHIAN *et al.*, 2023).

O sistema sensorial dos sistemas neurológicos biológicos converte dados externos, como luz, odores, toque, sabores e outros, em pulsos. Um pulso é uma alteração na voltagem que é propagada transmitindo informações (KASABOV, 2019). Esses pulsos são então transmitidos ao longo da cadeia neuronal sendo processados, gerando uma resposta ao ambiente.

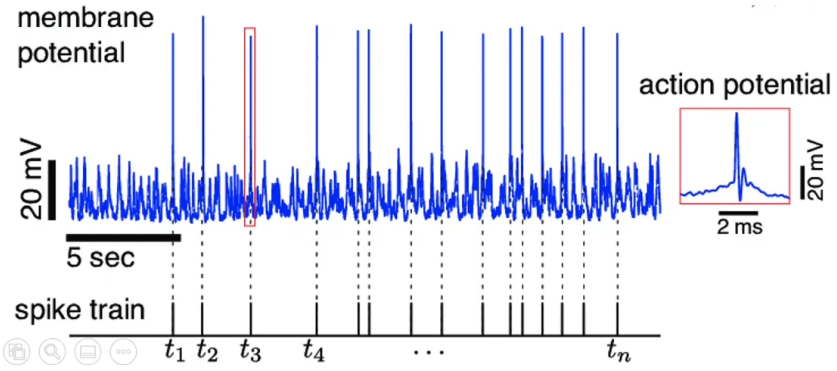
O neurônio biológico dispara apenas quando um certo nível de sinais excitatórios (voltagem) se acumula acima de um limiar em seu citoplasma, permanecendo inativo quando não há sinal, portanto, esse tipo de célula é muito eficiente em termos de consumo de energia e processamento.

Para obter as vantagens mencionadas, as Redes Neurais de Pulso (RNP), em vez de empregar valores de ativação contínuos, como as RNs, utilizam **pulsos** nas camadas de entrada, ocultas e de saída. As RNPs também podem ter entradas contínuas e manter suas propriedades.

Uma RNP **não é** uma simulação um-para-um de neurônios. Em vez disso, ela aproxima certas capacidades computacionais de propriedades biológicas específicas. Alguns estudos, como (JONES; KORDING, 2020), criaram modelos muito mais próximos aos neurônios naturais explorando a não linearidade dos dendritos e outras características neurais, obtendo resultados notáveis na classificação.

Como pode ser visto na Figura 10, os neurônios das RNPs, com os parâmetros corretos, são muito tolerantes a ruídos porque atuam como um **filtro passa-baixa**. Eles geram pulsos mesmo quando um nível considerável de interferência está presente. Tais neurônios são muito sensíveis ao tempo, sendo ótimos para processar fluxos de dados (ESHRAGHIAN *et al.*, 2023).

Figura 10 – Pulsos de um sinal ruidoso.



Fonte: (GOODMAN *et al.*, 2022)

2.1.8.1 Neurônio de Pulso

Embora o foco deste trabalho seja nos *Leaky Integrate and Fire Neurons* (LIF) porque são mais simples, mais eficientes e atualmente generalizam melhor para a maioria dos problemas (GOODMAN *et al.*, 2022), existem modelos mais biologicamente precisos como o **neurônio Hodgkin-Huxley** (GERSTNER *et al.*, 2014) e outros como (JONES; KORDING, 2020) que criaram modelos mais próximos aos neurônios naturais explorando a não linearidade dos dendritos e outras características neurais.

2.1.8.2 Entendendo o LIF

O LIF é um dos modelos de neurônios mais simples em RNPs, ainda assim, pode ser aplicado com sucesso na maioria dos problemas em que as RNPs podem ser usadas.

O LIF, assim como um neurônio de RN, recebe a soma dos inputs ponderados, mas, em vez de passá-lo diretamente para sua função de ativação, algum *vazamento* é aplicado, diminuindo em algum grau a soma.

O LIF se comporta muito como circuitos Resistor-Capacitor, como pode ser visto na Figura 11. Aqui, R é a resistência ao vazamento da corrente, I_{in} é a corrente de entrada, C é a capacitância, U_{mem} representa é o potencial acumulado e v é um interruptor que permite que o capacitor se descarregue (ou seja, emita um pulso) quando um determinado limiar de potencial é alcançado.

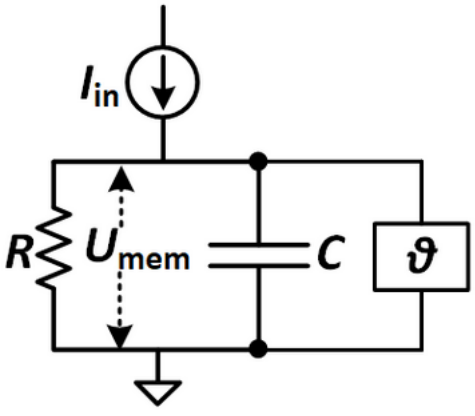


Figura 11 – Modelo RC: Fonte: (ESHRAGHIAN *et al.*, 2023)

Ao contrário do neurônio Hodgkin-Huxley, os pulsos são representados como **uns** dispersamente distribuídos em uma sequência de **zeros**, como ilustrado nas s Figura 12 e Figura 13. Essa abordagem simplifica os modelos e reduz a potência computacional e o armazenamento necessário para executar uma RNP.

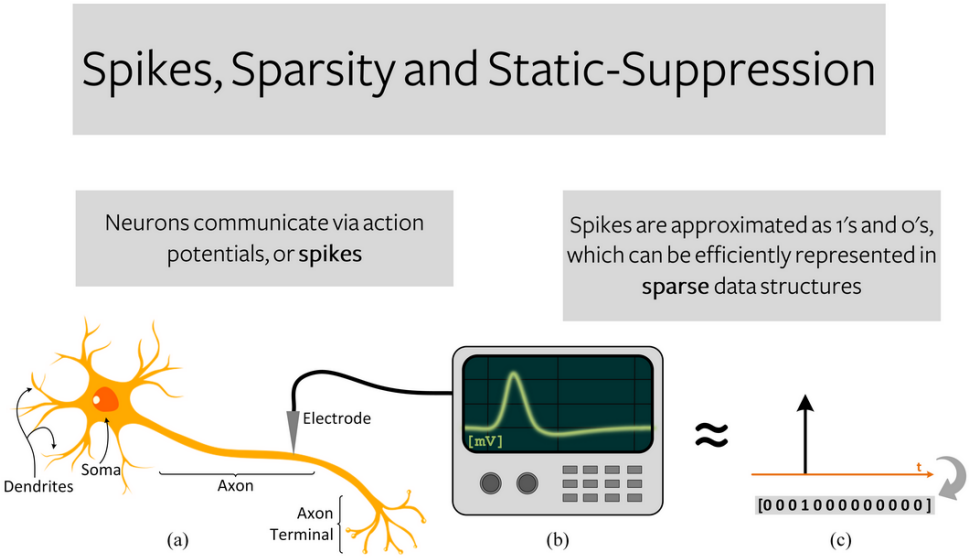


Figura 12 – Dispersão em Redes Neurais de Pulsos. Fonte: (ESHRAGHIAN *et al.*, 2023)

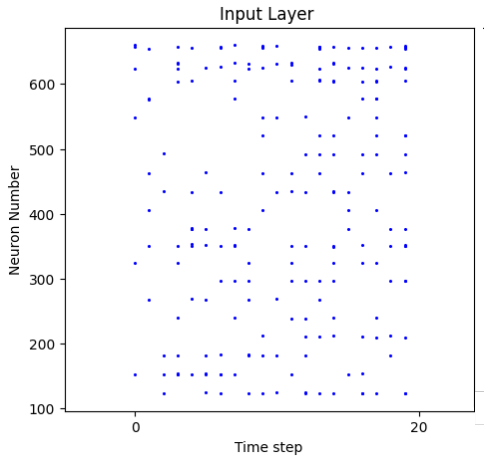


Figura 13 – Atividade dispersa de uma RNP: O eixo horizontal representa o momento no qual os dados estão sendo processados e o vertical representa o número de neurônios da RNP. Note que, na maior parte do tempo, muito poucos neurônios são ativados. Fonte: O autor.

Como resultado do mencionado acima, em RNP, a informação é codificada no formato de *tempo* e/ou *taxa* de pulsos, proporcionando consequentemente grandes capacidades de processamento de fluxos de dados, mas limitando o processamento de dados estáticos.

O modelo LIF é governado pelas equações abaixo (ESHRAGHIAN *et al.*, 2023).

Considerando que Q é uma medida de carga elétrica e $V_{\text{mem}}(t)$ é a diferença de potencial na membrana em um determinado tempo t, então a capacitância do neurônio C é dada pela Equação Equação 2.13.

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

Então, a carga do neurônio pode ser expressa pela Equação Equação 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 Equação 2.15. This expression express the current in the capacitive part of the neuron I_C

Para saber como essa carga muda ao longo do tempo (ou seja, medir a corrente), podemos derivar Q como na Equação Equação 2.15. Essa expressão representa a corrente na parte capacitiva do neurônio I_C.

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

Para calcular a corrente total passando pela parte resistiva do circuito, podemos usar a lei de Ohm:

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

Então, considerando que a corrente total não muda, como visto na Figura 14, temos a corrente total de entrada I_{in} do neurônio como na Equação Equação 2.17.

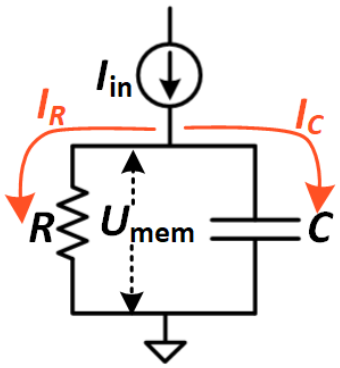


Figura 14 – Modelo RC para correntes: $I_{in} = I_R + I_C$

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

Portanto, para descrever a membrana passiva, temos a Equação linear Equação 2.18.

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

Então, se considerarmos $\tau = R.C$ como a **constante de tempo da membrana**, obtemos tensões em ambos os lados da Equação Equação 2.19, que **descreve o circuito RC**.

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

A partir disso, igualando $I_{in} = 0$ (ou seja, sem entrada) e considerando que $\tau = R.C$ é uma constante e, além disso, atribuindo um potencial inicial $V_{mem}(0)$, o comportamento da voltagem do neurônio pode ser modelado como uma curva exponencial, como visto na Equação Equação 2.20.

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

Então, pode-se dizer que: Na ausência de uma entrada I_{in}, o potencial da membrana decai exponencialmente, como ilustrado na Figura 15 e implementado no Código Algoritmo 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: $I_{in} = 0$

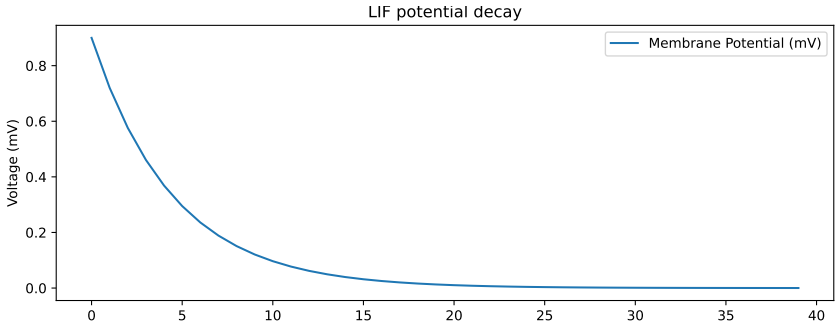


Figura 15 – Decaimento do potencial da membrana. Fonte: O autor

Com os resultados da Equação Equação 2.19, é possível calcular o aumento da voltagem, conforme visto na Equação Equação 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 que quando os potenciais de ação aumentam, ainda há um comportamento exponencial, como visto na Figura 16 e implementado no Código Algoritmo 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: $I_{in} = 1$

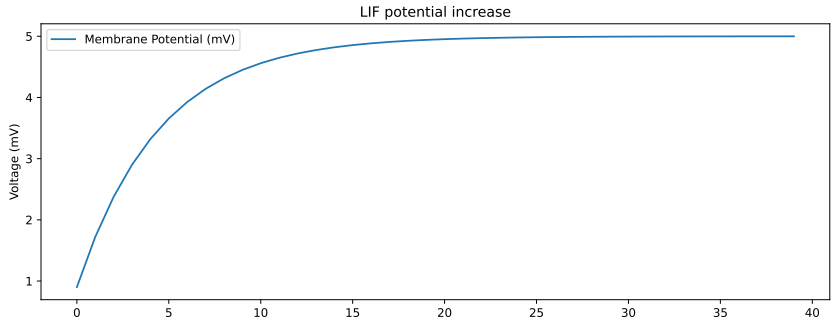


Figura 16 – Aumento do potencial da membrana. Fonte: O autor

Levando em consideração um certo **limiar** que indica um *reset* nvoltagem do neurônio e dois tipos de *resets* (para zero e subtração do limiar), finalmente é possível modelar o comportamento completo do LIF, conforme ilustrado na Figura 17 e implementado no Código Algoritmo 2.3:

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

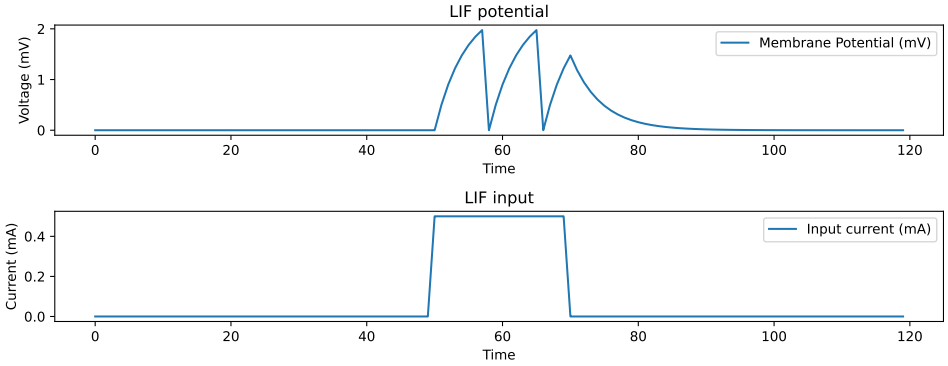


Figura 17 – Gráfico do LIF simulado completo: Foram fornecidos 0.5 mA de corrente no intervalo de tempo de 51 a 70. Fonte: O autor

2.1.8.3 Outra interpretação do LIF

Começando com a Equação Equação 2.19 e usando o Método de Euler para resolver o modelo LIF:

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

Resolvendo a derivada na Equação Equação 2.23, obtemos o potencial da membrana para qualquer tempo $t + \Delta t$ no futuro:

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

2.1.8.4 Treinamento

Como as RNPs são treinadas? Esta ainda é uma questão em aberto. Um neurônio RNP tem o comportamento de sua função de ativação mais parecido com uma **função de passo**. Portanto, em princípio, não podemos usar soluções baseadas em descida de gradiente porque esse tipo de função **não é** diferenciável (KASABOV, 2019).

Mas existem algumas ideias que podem lançar alguma luz sobre este assunto: Enquanto algumas observações *in vivo/in vitro* mostram que os cérebros, em geral, aprendem fortalecendo/enfraquecendo e adicionando/removendo sinapses ou até mesmo criando novos neurônios ou outros métodos complicados como pacotes de RNA, existem algumas mais aceitáveis como as da lista abaixo (KASABOV, 2019):

- **Plasticidade Dependente do Tempo de Pulsos (STDP)**: Se um neurônio pré-sináptico dispara **antes** do pós-sináptico, há um fortalecimento na conexão, mas se o neurônio pós-sináptico disparar antes, então há um enfraquecimento.
- **Descida de Gradiente Emprestada**: Aproxima a função de passo usando outra função matemática, que é diferenciável (como uma sigmoide), para treinar a rede. Essas aproximações são usadas apenas na fase de *backpropagation*, enquanto mantêm a função de passo na fase do *feed-forward* (KASABOV, 2019).
- **Algoritmos Evolutivos**: Usam a seleção dos mais aptos ao longo de muitas gerações de redes.
- **Reservatório/Computação Dinâmica**: **Redes de estado de eco** ou **Máquinas de estado líquido**, respectivamente.

2.1.9 *Autoencoders*

Como ilustrado na Figura 18 *autoencoders* são redes neurais treinadas para reconstruir seus dados de entrada. Elas 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 da entrada que pode ser ou não ser comprimida (GOODFELLOW; BENGIO; COURVILLE, 2016). Tais funções se traduzem em camadas como ilustrado na Figura 19 cujos neurônios são todos unidades ativas com ou sem funções de ativação. É importante destacar que um *autoencoder* pode ser tão raso como o mostrado na figura ou x e r podem ser redes profundas.

O principal objetivo de um *autoencoder* é aprender uma representação dos dados de entrada na camada de **código** e, em seguida, reconstruir os dados de entrada com a maior precisão possível usando o decodificador. Geralmente, os *autoencoders* são projetados para serem incapazes de copiar perfeitamente os dados de entrada pois isso os tornaria demasiadamente complexos e, em alguns casos, inúteis como será explicado adiante.

Figura 18 – Representação funcional de um *autoencoder*: Sendo o vetor x uma entrada, então o vetor h é resultado da aplicação de uma função codificadora f sobre x: $h = f(x)$, portanto r é a reconstrução de x a partir h através de uma função decodificadora g: $r = g(h)$. Tal processo pode ou não implicar em uma perda de informação.

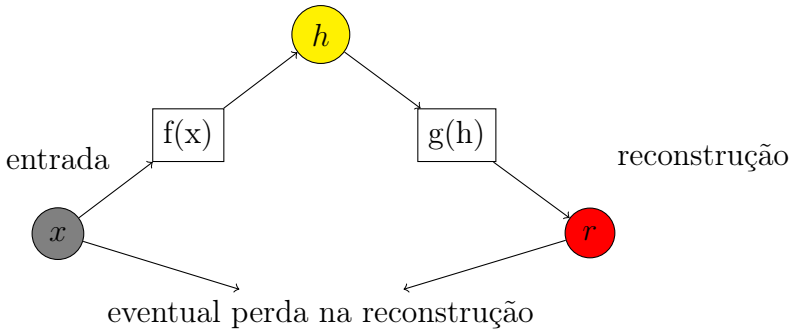
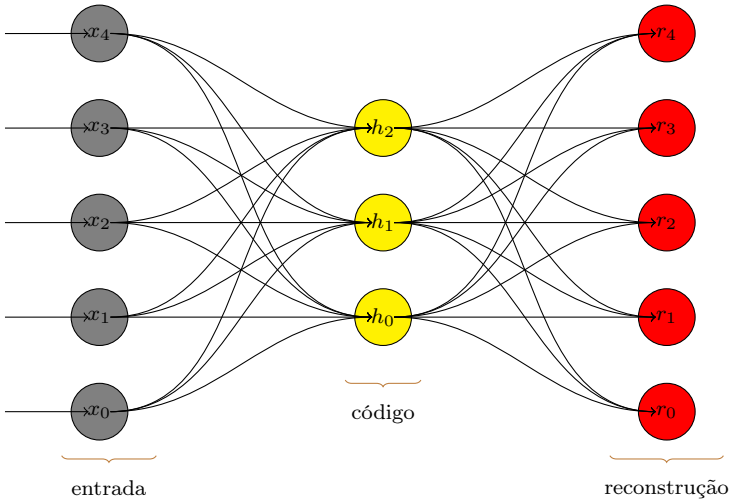


Figura 19 – Exemplo esquemático de um *autoencoder*: Os nós de entrada estão em cinza, a camada de código em amarelo contêm as características codificadas e finalmente a camada de reconstrução em vermelho contêm um cópia aproximada da entrada.



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, *autoencoders* são ótimos produtores de vetores de características. Hipoteticamente é possível criar *autoencoders* cuja a camada de código seja apenas um número inteiro caso tanto o codificador quanto o decodificador sejam grandes e complexos o suficiente, mas isso se mostraria inútil caso se necessitasse algum tipo de representação da informação original que fosse significativa, ademais, excluindo-se casos triviais, segundo GOODFELLOW; BENGIO; COURVILLE isso ainda não se verifica na prática.

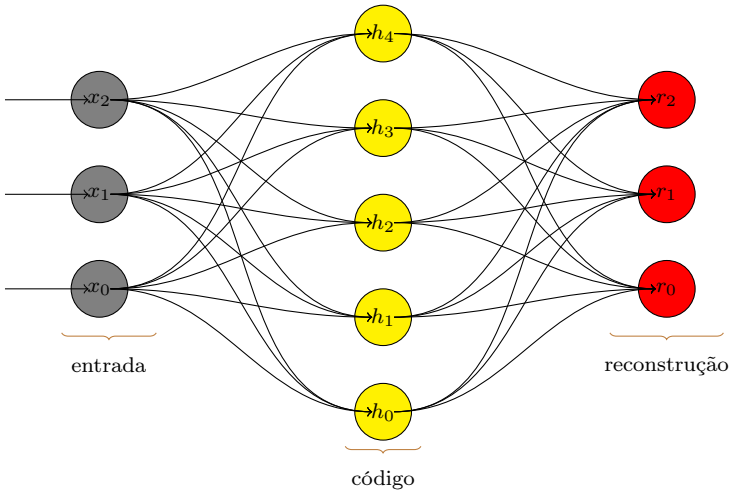
Em se tratando de treinamento as técnicas já conhecidas e testadas para redes neurais em geral (como, por exemplo, *gradient descent* com *minibatch* ou estocástico) podem ser usadas (GOODFELLOW; BENGIO; COURVILLE, 2016) incluindo as formas de regularização com será exposto mais adiante.

2.1.9.1 Autoencoders sub-completos ou clássicos

Esse tipo de rede neural, como ilustrado na Figura 19, tem sua camada de código limitada em dimensionalidade para apenas aproximar a entrada e priorizar certos aspectos dos dados extraindo dessa forma os componentes mais cruciais e significativos de um conjunto de dados (GOODFELLOW; BENGIO; COURVILLE, 2016). Um autoencoder clássico age de forma muito semelhante ao algoritmo *Principal Component Analysis* (PCA) (BENGIO; COURVILLE; VINCENT, 2014).

2.1.9.2 Autoencoders supra-completos

Figura 20 – Exemplo esquemático de um *autoencoder* supra-completo.



Como mostrado na Figura 20, ao contrário dos autoencoders sub-completos, os supra-completos tem sua camada de código com uma **dimensão maior** do que os dados de entrada. Dessa forma as características dos dados de entrada podem ser codificadas

mais de uma vez e uma reconstrução perfeita é atingida. Estruturas assim são inúteis já que não conseguem extrair características significativas a não ser que sejam aplicadas técnicas de regularização (BENGIO; COURVILLE; VINCENT, 2014).

2.1.9.3 *Autoencoders* regularizados

Segundo (BENGIO; COURVILLE; VINCENT, 2014) uma das formas de regularização é justamente a limitação da dimensão da camada de código, a mesma aplicada aos *autoencoders* sub-completos, como estes já foram explicados apresentar-se-ão a seguir outras formas de regularização.

Considerando que *autoencoders* devem ser treinados para minimizar uma função de erro E que compara a entrada original x com a reconstrução r então um autoencoder **regularizado** deve ter adicionado a essa função um termo de regularizador ϕ como a mostrado na Equação Equação 2.24 que é um termo que pode corresponder a uma regularização L1, L2, $\Omega(h)$ (discutidas no apêndice seção E.1).

$$E(x, r) + \phi \tag{2.24}$$

Segundo (GOODFELLOW; BENGIO; COURVILLE, 2016) usando técnicas de regularização é possível até mesmo para um *autoencoder* supra-completo que o mesmo extraia características úteis para uso. Tais técnicas devem prover a seguintes propriedades para a rede:

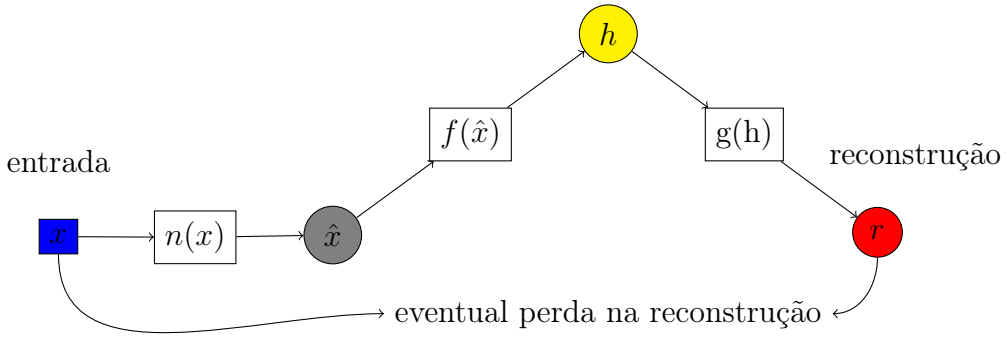
- dispersão: É a tendência de ativar apenas um pequena quantidade das unidades (também conhecidas como dimensões) da camada de código para qualquer entrada dada criando uma representação mais compacta.

- pequena variação no gradiente: Garantir pouca amplitude nos valores dos gradientes em relação as variações ou ruídos de uma entrada evita que grandes flutuações ocorram na camada de código proporcionando um comportamento estável.

2.1.9.4 *Denoising autoencoders*

Tais redes podem ser caracterizadas também como *autoencoders* regularizados. Porém há uma diferença fundamental: Aqui o foco não é criar funções de regularização e sim adicionar ruídos **apenas** na informação de entrada mas mantendo a intactos os dados que serão comparados na função de erro como mostra a Figura 21. Um *autoencoder* treinado dessa forma, além de ser mais resiliente a ruídos, também adquirire uma funcionalide de preencher falhas de informação nos dados de entrada quando o componente adicionador de ruído é removido e os dados são diretamente apresentados a rede.

Figura 21 – Representação funcional de um *denoising autoencoder*: Sendo o vetor x uma entrada e n um função que adiciona um ruído aleatório então a informação com ruído $\hat{x}$ é definida como $\hat{x} = n(x)$, portanto o vetor h é resultado da aplicação de uma função codificadora f sobre x: $h = f(x)$, finalmente r é a reconstrução de x a partir h através de uma função decodificadora g: $r = g(h)$. É importante notar que r é comparada com x e não com $\hat{x}$.

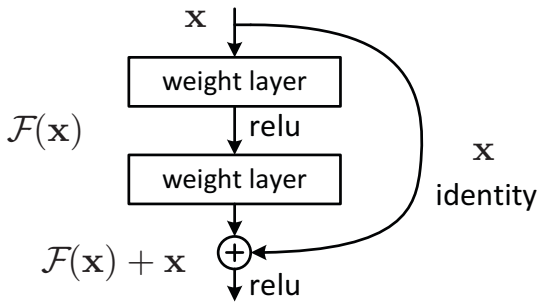


Fonte: O autor

2.1.10 Redes neurais residuais (ResNets)

Segundo (HE *et al.*, 2015) a ideia-chave por trás das *ResNets* é a inclusão de conexões de salto como ilustrado na Figura 22, também conhecidas como mapeamentos de identidade, permitindo que a saída de uma camada seja adicionada elemento-a-elemento diretamente à entrada da camada subsequente. Para que isso aconteça é importante que tanto a saída quanto a entrada tenham a mesma dimensão.

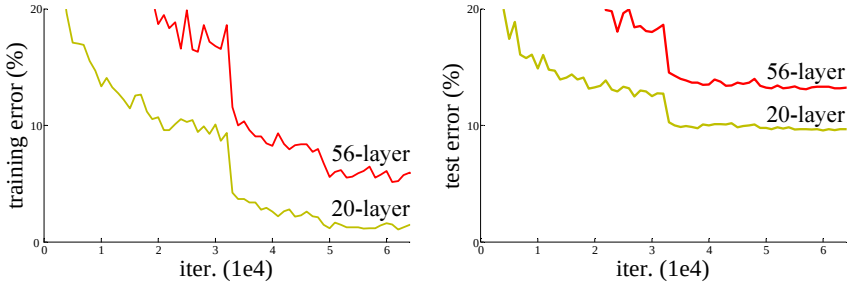
Figura 22 – Bloco de uma Rede Neural Residual: x contorna as camada intermediárias $F(x)$ via uma função identidade somando-se a $F(x)$ ao final do bloco.



Fonte: (HE *et al.*, 2015)

Quando uma rede chega a um certo limite de acurácia, intuitivamente se pode pensar que adicionando mais camadas o desempenho da mesma melhore, no entanto, na prática isso não se verifica. O que se observa em redes mais profundas pode ser exatamente o contrário como mostrado na Figura 23.

Figura 23 – Desempenho de duas redes neurais não residuais com 56 e 20 camadas. A esquerda a taxa de erro durante o treinamento, a direita a taxa de erro em relação ao conjunto de testes.



Fonte: (HE *et al.*, 2015)

Redes neurais muito profundas podem sofrer de problemas como o desaparecimento ou explosão do gradiente, onde os gradientes (ou seja, as direções para ajustar os pesos da rede) se tornam muito pequenos ou grandes à medida que são retro-propagados durante o treinamento. Isso pode dificultar o treinamento eficaz das camadas mais profundas.

Então já que, empiricamente, as redes com menos camadas estão expostas a menos problemas de otimização, se pode juntar várias dessas formando assim uma rede mais profunda e com mais hiperparâmetros que podem ser ajustados. Isso, aliado as técnicas de regularização (ver Apêndice E) diminui a ocorrência de *vanishing/exploding gradients* um problema comum em redes com muitas camadas que pode piorar, impossibilitar ou diminuir a níveis impraticáveis o aprendizado da rede (HE *et al.*, 2015).

Em vez de aprender a mapear diretamente de x para y, as camadas da rede aprendem a mapear de x para a diferença entre x e y, ou seja, para o **resíduo**. Isso simplifica o treinamento, pois a rede precisa apenas ajustar os resíduos, em vez de aprender a mapear completamente as entradas para as saídas.

Além das características já expostas as redes residuais não adicionam complexidade computacional além da negligenciável adição elemento-a-elemento. Isso facilita a comparação com redes não residuais com o mesmo número de hiperparâmetros.

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. *et al.* Time–frequency analysis of biosignals. *IEEE Engineering in Medicine and Biology Magazine*, v. 28, n. 5, p. 14–29, Sep 2009.

BENGIO, Y. *et al. Representation Learning: A Review and New Perspectives*. 2014.

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

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

BIDGOLY, A. J. *et al.* 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>.

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 *et al. 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. *et al. 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>.

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. *et al. 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. *et al.* 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 – A inferência bayesiana

A inferência bayesiana é um método estatístico usado para atualizar crenças ou quantificar a incerteza sobre os parâmetros de um modelo com base em dados observados. É nomeado em homenagem ao Reverendo Thomas Bayes, um matemático do século XVIII que introduziu o teorema de Bayes, o princípio fundamental subjacente à inferência bayesiana.

Na inferência bayesiana:

- **Prior**: Começa com uma crença inicial ou distribuição de probabilidade priori sobre os parâmetros de interesse. Isso representa o que se sabe sobre os parâmetros antes de observar quaisquer dados.
- **Verosimilhança**: Dados os dados observados, há uma função de verossimilhança que descreve quão prováveis são os dados para diferentes valores dos parâmetros no modelo.
- **Posterior**: Usando o teorema de Bayes, a distribuição prior é atualizada com a função de verossimilhança para obter a distribuição posterior. Esta distribuição posterior representa as crenças atualizadas sobre os parâmetros após observar os dados.

Matematicamente, o teorema de Bayes afirma: $P(\theta|D) = \frac{P(D|\theta) \times P(\theta)}{P(D)}$

Onde:

- $P(\theta|D)$ é a distribuição posterior dos parâmetros (θ) dado os dados (D).
- $P(D|\theta)$ é a verossimilhança de observar os dados dados os parâmetros.
- $P(\theta)$ é a distribuição prior dos parâmetros.
- $P(D)$ é a probabilidade de observar os dados, também conhecida como verossimilhança marginal.

A inferência bayesiana fornece um framework coerente para atualizar crenças à medida que novas evidências se tornam disponíveis, permitindo a incorporação de conhecimento prévio e incerteza. É amplamente utilizado em diversos campos, incluindo estatística, aprendizado de máquina, econometria e inteligência artificial.

Uma distribuição de probabilidade é uma função matemática que descreve a probabilidade de observar cada possível resultado de uma variável aleatória em um espaço

amostral dado. Em outras palavras, ela nos diz como a probabilidade total está distribuída entre todos os possíveis valores que a variável aleatória pode assumir.

Existem dois tipos principais de distribuições de probabilidade:

- **Distribuição de Probabilidade Discreta**: Esse tipo de distribuição está associado a variáveis aleatórias discretas, que só podem assumir um número contável de valores distintos. Exemplos incluem a distribuição binomial, distribuição de Poisson e distribuição geométrica. Em uma distribuição de probabilidade discreta, as probabilidades são atribuídas a resultados individuais, e a soma de todas as probabilidades é igual a 1.

- **Distribuição de Probabilidade Contínua**: Esse tipo de distribuição está associado a variáveis aleatórias contínuas, que podem assumir qualquer valor dentro de um intervalo especificado. Exemplos incluem a distribuição normal (Gaussiana), distribuição exponencial e distribuição uniforme. Em uma distribuição de probabilidade contínua, as probabilidades são atribuídas a intervalos de valores, e a área total sob a curva de densidade de probabilidade (PDF) é igual a 1.

As distribuições de probabilidade são fundamentais na estatística e na teoria da probabilidade, pois nos permitem modelar incerteza e aleatoriedade em diversos fenômenos do mundo real. Elas são usadas para fazer previsões, estimar parâmetros e realizar inferência estatística em uma ampla gama de áreas, incluindo ciência, engenharia, finanças e ciências sociais.

A.1 Exemplo

Vamos considerar um exemplo de uma distribuição de probabilidade discreta: a distribuição binomial.

Imagine que você tem uma moeda viciada que tem uma probabilidade p de cair cara quando lançada para cima. Você quer saber a probabilidade de obter um certo número de caras, digamos k, em n lançamentos da moeda.

A função de massa de probabilidade (PMF) da distribuição binomial, denotada como $P(X = k)$, dá a probabilidade de obter exatamente k caras em n lançamentos.

A fórmula para a PMF da distribuição binomial é: $P(X = k) = \binom{n}{k} \times p^k \times (1-p)^{n-k}$

Onde:

- n é o número de tentativas (lançamentos da moeda).

- k é o número de sucessos (caras).

- p é a probabilidade de sucesso (probabilidade de obter cara em cada lançamento).
- $(1-p)$ é a probabilidade de falha (probabilidade de obter coroa em cada lançamento).
- $\binom{n}{k}$ é o coeficiente binomial, representando o número de maneiras de escolher k sucessos em n tentativas.

Por exemplo, se você quiser encontrar a probabilidade de obter exatamente 3 caras em 5 lançamentos da moeda, com uma probabilidade $p = 0.5$ de obter caras, você substituiria $n = 5$, $k = 3$ e $p = 0.5$ na fórmula:

$$\begin{aligned} P(X = 3) &= \binom{5}{3} \times (0.5)^3 \times (1 - 0.5)^{5-3} \\ P(X = 3) &= \frac{5!}{3!(5-3)!} \times (0.5)^3 \times (0.5)^2 \\ P(X = 3) &= \frac{5 \times 4}{2 \times 1} \times 0.125 \times 0.25 \\ P(X = 3) &= 0.3125 \end{aligned} \tag{A.1}$$

Portanto, a probabilidade de obter exatamente 3 caras em 5 lançamentos da moeda é aproximadamente 0.3125, ou 31.25

APÊNDICE B – Como fazer uma rede neural

B.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:

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

(RASHID, 2016)

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

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

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*:

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

- *sigmoid function*:

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

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:

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

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 B.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.

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

B.2 Junção dos aproximadores: Redes neurais

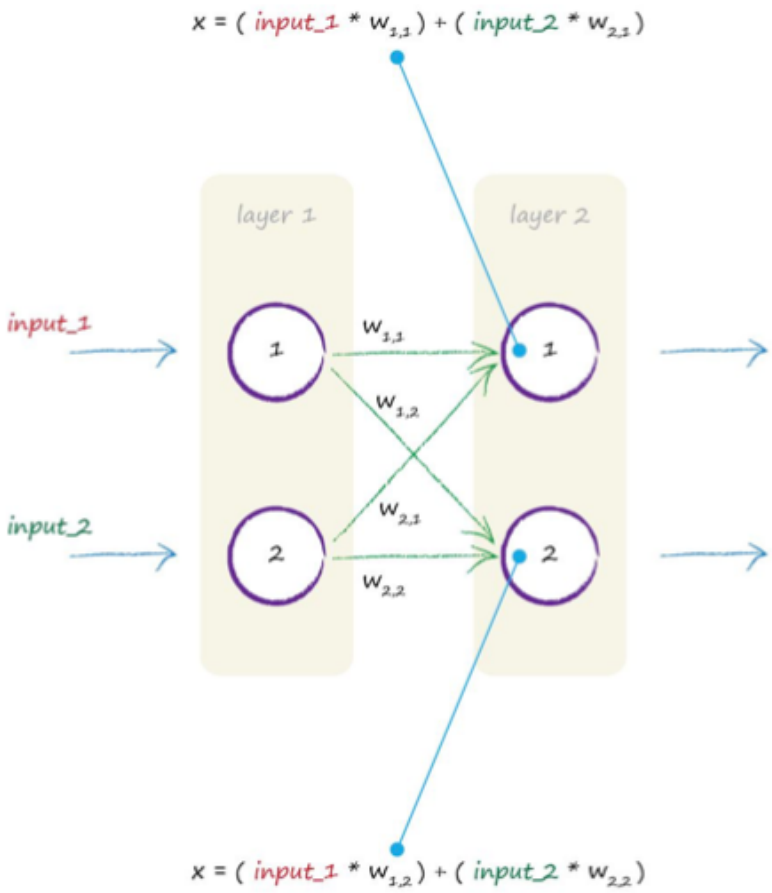
Já que a sessão anterior definiu o que é um aproximador de funções, nesta sessão, usando as definições já vistas, será apresentada a definição de uma rede neural que é a junção de vários aproximadores que, dessa forma, conseguem delimitar espaços de resultados mais complexos além daqueles que podem ser 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 24.

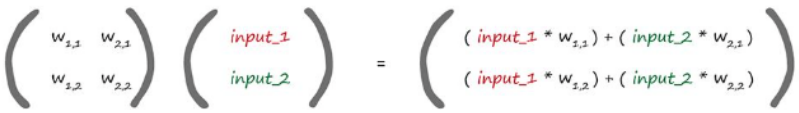
Figura 24 – Uma rede neural simples Fonte: O autor



B.3 Uma rede neural simples

Como visto na figura 24 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 25, 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 25 – Representação de uma rede neural usando matrizes



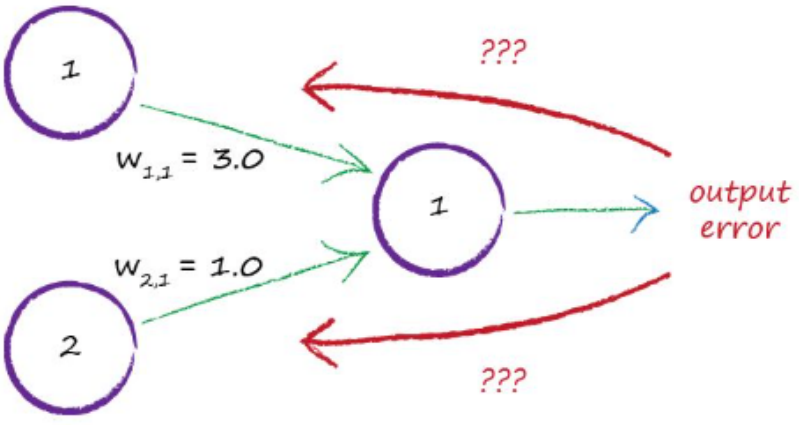
B.4 Redes neurais multicamadas

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 24 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.

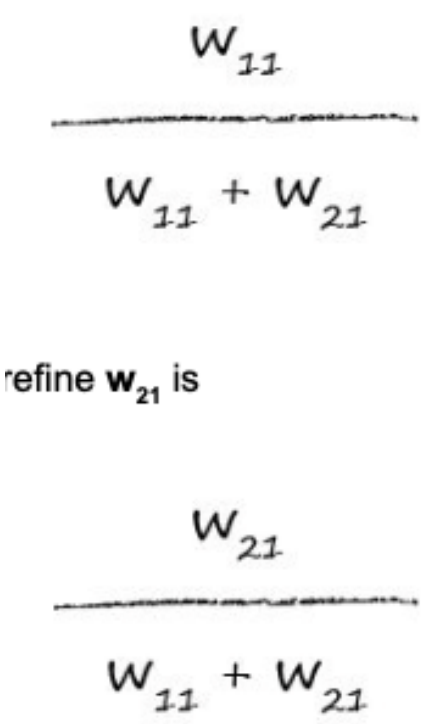
Figura 26 – Retro-propagação



B.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 27.

Figura 27 – 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 B.7.

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

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 28 e 29 mostram em mais detalhes o funcionamento do algoritmo.

Figura 28 – Retro-propagação da camada de saída para a oculta

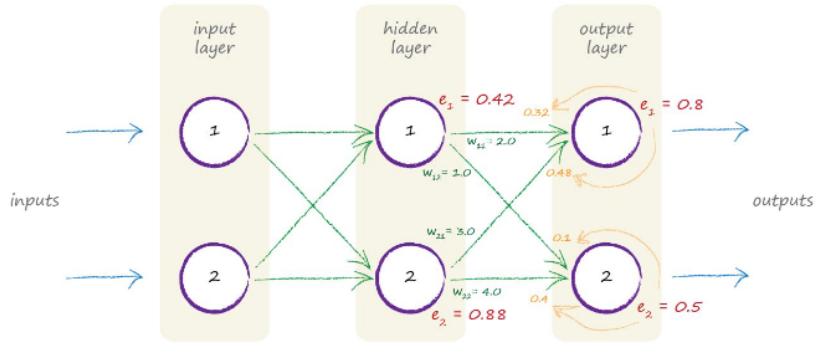
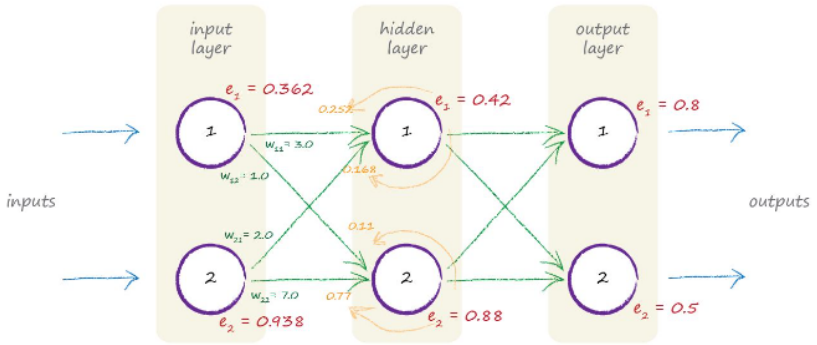


Figura 29 – Retro-propagação da camada oculta para a entrada



B.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 B.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 B.9.

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

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

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 B.10 .

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

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 B.9 e B.10 se tem a expressão B.11.

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

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 30 – Rede neural de três camadas escondidas, os pesos representados pelas variáveis w_1, w_2, w_3 na figura 30 são alguns dos parâmetros da rede que devem ser ajustados para que hajam resultados satisfatórios.

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

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

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

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

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 B.15.

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

Resultando nos valores:

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

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.

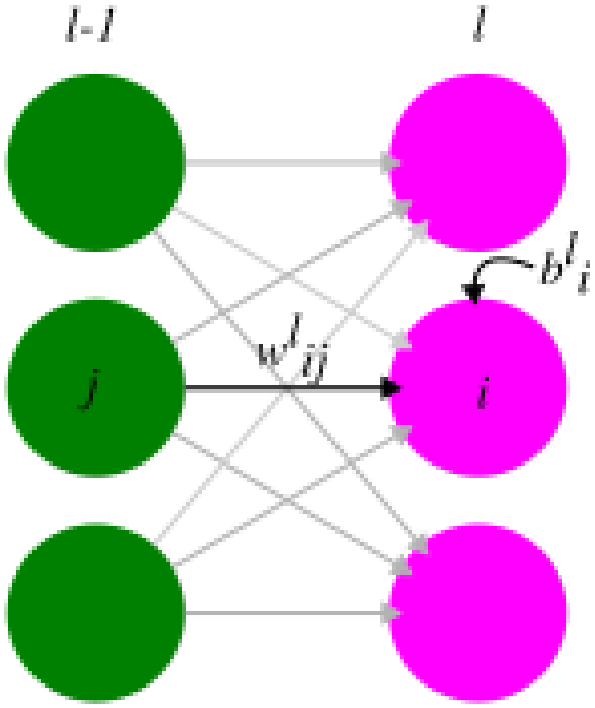
$$E = \frac{1}{2.n}.\sum_{i=0}^{n}(E_i)^2 = \frac{1}{2n}.(-2^2 + 3^2) = \frac{1}{2.2}.(4 + 9) = \frac{1}{4}.13 = \frac{13}{4} \tag{B.17}$$

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 31.

Figura 31 – 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 31 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 B.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 B.19.

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

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

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 32 – 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 32 é representada na equação B.20.

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

APÊNDICE C – Paralelismo

APÊNDICE D – Medidas dos tempos de execução do software

APÊNDICE E – Regularização em redes neurais

E.1 Decaimento de Peso (Regularização L2)

Suponha que tenhamos uma rede neural simples com pesos W. Durante o treinamento, adicionamos um termo de regularização à função de perda, que penaliza o quadrado dos pesos:

$$L_{\text{total}} = L_{\text{dados}} + \lambda \|W\|_2^2 \tag{E.1}$$

Aqui, L_{dados} representa a perda de dados (por exemplo, erro quadrático médio para tarefas de regressão), λ é o parâmetro de regularização, e $\|W\|_2^2$ é a norma L2 dos pesos. Vamos supor que nossa matriz de pesos W seja:

$$W = \begin{bmatrix} 1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

Se definirmos $\lambda = 0.1$, o termo de regularização seria $0.1 \times (1^2 + 0.5^2 + (-0.3)^2 + 0.8^2) = 0.1 \times (1 + 0.25 + 0.09 + 0.64) = 0.1 \times 1.98 = 0.198$. Durante o treinamento, o otimizador atualiza os pesos considerando tanto a perda de dados quanto o termo de regularização. O termo de regularização incentiva pesos menores, prevenindo o sobreajuste.

E.2 Penalidade de Esparsidade

:

Vamos considerar um autoencoder esparsificado com uma penalidade de esparsidade adicionada à função de perda para encorajar a esparsidade nas ativações das unidades ocultas.

Suponha que tenhamos uma camada oculta com valores de ativação $h = [0.9, 0.1, 0.8, 0.05]$.

O termo de penalidade de esparsidade pode ser definido como a divergência Kullback-Leibler (KL) entre o nível de esparsidade desejado ρ e a média da ativação de cada unidade oculta:

$$\Omega(h) = \sum_i (\rho \log(\rho/\hat{\rho}) + (1-\rho) \log((1-\rho)/(1-\hat{\rho}))) \tag{E.2}$$

Aqui, $\hat{\rho}$ é a média da ativação da unidade oculta i.

Vamos supor $\rho = 0.2$. Se a média da ativação da unidade oculta i for $\hat{\rho}_i = 0.5$, então a penalidade de esparsidade para essa unidade seria:

$$\begin{aligned} \Omega(h_i) &= (0.2 \log(0.2/0.5) + 0.8 \log(0.8/0.5)) \\ &= (0.2 \times -0.3219 + 0.8 \times 0.3219) \\ &= -0.0644 + 0.2575 \\ &= 0.1931 \end{aligned} \tag{E.3}$$

Durante o treinamento, a penalidade de esparsidade incentiva a rede a ter ativações esparsas, focando em características importantes dos dados.

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