UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Introdução à neurociência computacional com a linguagem Python

Weverson Vieira do Nascimento

DM 39/2023

BELÉM - PA

2023

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Weverson Vieira do Nascimento

Introdução à neurociência computacional com a linguagem Python

Dissertação apresentada como exigência parcial para obtenção do grau de Mestre em Engenharia Elétrica pela Universidade Federal do Pará. Área de concentração: Engenharias.

Orientador: Prof. Dr. Antonio Pereira Junior

DM 39/2023

BELÉM - PA
2023

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

N244i Nascimento, Weverson Vieira do.
Introdução à neurociência computacional com a linguagem Python / Weverson Vieira do Nascimento. — 2023.
xviii, 61 f. : il. color.

Orientador(a): Prof. Dr. Antonio Pereira Junior
Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2023.

1. Neurociência computacional. 2. Python. 3. Conexões neuronais. 4. Inteligência artificial. I. Título.

CDD 629

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

“INTRODUÇÃO À NEUROCIÊNCIA COMPUTACIONAL COM A LINGUAGEM PYTHON”

AUTOR: **WEVERSON VIEIRA DO NASCIMENTO**

DISSERTAÇÃO DE MESTRADO SUBMETIDA À BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, SENDO JULGADA ADEQUADA PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE TELECOMUNICAÇÕES.

APROVADA EM: 29/12/2023

BANCA EXAMINADORA:

Prof. Dr. Antonio Pereira Júnior
(Orientador – PPGEE/ITEC/UFPA)

Prof. Dr. Miércio Cardoso de Alcântara Neto
(Avaliador Interno – PPGEE/ITEC/UFPA)

Prof. Dr. Ronaldo de Freitas Zampolo
(Avaliador Externo ao Programa – FCT/ITEC/UFPA)

VISTO:

Prof. Dr. Diego Lisboa Cardoso

(Coordenador do PPGEE/ITEC/UFPA)

Para a comunidade acadêmica brasileira,
que segue com um trabalho exemplar,
apesar das várias adversidades.

AGRADECIMENTOS

Primeiramente, agradeço a Deus, pois sem a força dele não seria possível a conclusão deste trabalho.

Agradeço aos meus familiares, pelo apoio em toda a jornada até aqui, especialmente aos meus pais, Áurea e Raimundo, por sempre apoiarem minhas decisões. Cabe destaque aos tios Ari e Vanise, pelo apoio durante o início do mestrado, antes sequer de eu ir para Florianópolis.

Agradeço, também, à Laís Rocha, por ter me apoiado e compreendido meus momentos de falta.

Faço aqui grandes agradecimentos ao meu orientador, Antonio, por todos os ensinamentos que me proporcionou nesse tempo, e pela paciência comigo, especialmente durante as várias ausências minhas.

Muitos amigos e colegas meus me deram força durante essa jornada. Começa destacando aqui Letícia de Miranda, Flávio Matos, os mais próximos, bem como os demais do bando, que direta ou indiretamente me apoiaram ao longo desse tempo todo. Agradecimentos especiais ao Geandreson Costa, Pedro Nascimento e Wesler Sales, pela companhia de sempre, pelos vários co-workings, e por lembrarem de ligar a câmera quando a gente não via ninguém durante a fase aguda da pandemia. Aos meus amigos do LINC, Lucas Vinícius, Ewerton Christian e Ana Zuila Castro, pela recepção e os vários cafés. Agradeço também aos meus colegas de UFPA, dos mais variados ambientes, em especial de ENGCOMP e do LAPS, que contribuíram com um pouco de conhecimento de mundo, conversas de corredor e cafés, essenciais para espairecer quando era necessário. Aqui cabe também um agradecimento para a secretaria do PPGEE, pois cada visita ao local revigorava meus ânimos, mesmo quando a minha situação não era das melhores.

Não poderia deixar de fazer um agradecimento de destaque ao Matheus Cardoso e Flávia Rocha, por todo o apoio durante o começo do meu mestrado, ainda na UFSC, sendo os amigos que eu não sabia que precisava. Muito obrigado pelo abrigo, pelas maratonas da novela dos congelados, as noites de *Overcooked* e as raivas vendo nossos times de futebol.

Um agradecimento especial para os meus colegas da Azos, pela compreensão durante minhas ausências por atividades acadêmicas, e pela motivação em todas as vezes que falei sobre meu mestrado.

Agradeço ao CNPq, pelo apoio financeiro.

Finalmente, fica aqui um agradecimento para qualquer um que, de alguma maneira que este cérebro não lembra, contribuiu para que este trabalho fosse concluído.

"Tente aprender alguma coisa sobre tudo e tudo sobre alguma coisa."

Thomas Henry Huxley

RESUMO

Este trabalho propõe um curso introdutório de neurociências computacionais, utilizando a linguagem de programação Python. O cérebro é um órgão complexo, despertando grande interesse na compreensão dos mecanismos biológicos subjacentes ao seu funcionamento. A neurociência computacional é um campo de estudo que busca contribuir para essa compreensão. O curso introdutório é destinado a alunos de graduação interessados em adquirir conhecimentos básicos em Neurociência Computacional. Inicialmente, o curso fornece uma base teórica, abrangendo tanto aspectos neurofisiológicos quanto matemáticos e algorítmicos, permitindo que estudantes de diversas áreas aproveitem o conteúdo com o mínimo de pré-requisitos. Em seguida, o curso apresenta modelos de neurônios, desde os mais simples até os mais elaborados, explorando como esses neurônios se conectam, incluindo circuitos conhecidos de conexões neuronais e a implementação do aprendizado nessas redes. Além disso, o curso aborda temas de inteligência artificial, como redes neurais e neuromórficas, estas últimas utilizando os modelos mencionados anteriormente. Utiliza códigos interativos em linguagem Python, de natureza livre e código aberto, para as simulações do conteúdo apresentado.

Palavras-chave: Neurociência computacional. Python. Conexões neuronais. Inteligência artificial.

ABSTRACT

This work presents a proposal for an introductory course on computational neuroscience, using the Python programming language. The brain is a complex organ, and there is significant interest in understanding the biological mechanisms underlying its functioning. Computational neuroscience is one of the fields of study that seeks to contribute to this understanding. The introductory course is aimed at undergraduate students interested in acquiring basic knowledge in Computational Neuroscience. The course initially provides a theoretical foundation in both neurophysiology and mathematics, as well as algorithmic concepts, to enable students from diverse backgrounds to benefit from its content with minimal prerequisites. The course then introduces models of neurons, ranging from simple to more elaborate ones, and explores how these neurons connect with each other, including some well-known neural connection circuits and how learning is implemented in these neuron networks. It also includes content on artificial intelligence, such as neural and neuromorphic networks, the latter using the models mentioned initially. The course utilizes interactive Python code, which is free and open-source, for simulating the presented content.

Keywords: Computational neuroscience. Python. Neural conections. Artificial intelligence

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AELIF	*Adaptative exponential leaky integrate-and-fire*
ANN	*Artificial neural network*
CNN	*Convolutional neural network*
DNN	*Deep neural network*
EEG	Eletroencefalograma
ELIF	*Exponential leaky integrate-and-fire*
GABA	Ácido γ-aminobutírico
GPG	*Central pattern generator*
LIF	*Leaky integrate-and-fire*
LSTM	*Long short-term memory*
LTD	*Long-term depression*
LTP	*Long-term potentiation*
MLP	*Multilayer perceptron*
PCP	Plasticidade de curto prazo
RNN	*Recurrent neural network*
SNN	*Spiking neural network*
STDP	*Spike-timing dependent plasticity*

LISTA DE SÍMBOLOS

V_m	Potencial de membrana
E_A	Concentração do íon A
z_A	Carga do íon A
k_B	Constante de Boltzmann
q_e	Carga fundamental
Na^+	Íon sódio
Ca^{2+}	Íon cálcio
K^+	Íon potássio
Cl^-	Íon cloreto
C_m	Capacitância da membrana neuronal
E_l	Potencial de vazamento
G_l	Condutância de vazamento
$\frac{\mathrm{d}}{\mathrm{d}t}$	Representação da operação de derivada na notação de Leibniz
y'	Representação da operação de derivada na notação de Lagrange
λ	Constante de decaimento radioativo
G_A	Condutância do íon A
V_{th}	Potencial de limiar
I_ap	Corrente aplicada no(s) neurônio(s)
V_{reset}	Potencial de reset
$\rightarrow$	Atualização do valor de uma variável
G_{ref}	Condutância refratária
τ_{ref}	Constante de tempo do período refratário

ΔG	Incremento de condutância
V_{th}^{0}	Potencial de limiar de referência
ΔV	Incremento de potencial
V_{max}	Potencial de membrana máximo do modelo ELIF
Δ_{th}	Intervalo de limiar
w	Variável de adaptação do modelo AELIF
m	Variável de portão de ativação de sódio
h	Variável de portão de inativação de sódio
n	Variável de portão de ativação de potássio
α_m	Constante de crescimento da variável de portão de ativação de sódio
β_m	Constante de decrescimento da variável de portão de ativação de sódio
α_h	Constante de crescimento da variável de portão de inativação de sódio
β_h	Constante de decrescimento da variável de portão de inativação de sódio
α_n	Constante de crescimento da variável de portão de ativação de potássio
β_n	Constante de decrescimento da variável de portão de ativação de potássio
m_∞	Variável de portão de ativação de sódio em regime permanente
h_∞	Variável de portão de inativação de sódio em regime permanente
n_∞	Variável de portão de ativação de potássio em regime permanente
τ_m	Constante de tempo da variável de portão de ativação de sódio
τ_h	Constante de tempo da variável de portão de inativação de sódio
τ_n	Constante de tempo da variável de portão de ativação de potássio
$G_{Na}^{(max)}$	Condutância máxima de sódio
$G_{K}^{(max)}$	Condutância máxima de potássio

G_{sin}	Condutância sináptica
τ_{sin}	Constante de tempo da sinapse
τ_{cresce}	Constante de tempo de crescimento da condutância sináptica
τ_{decai}	Constante de tempo de decaimento da condutância sináptica
$t_{disparo}$	Instante de tempo do disparo das célula pré-sináptica
K	Fator de controle da condutância sináptica
G_{max}	Valor máximo da condutância sináptica
p_0	Probabilidade de liberação do neurotransmissor
F	Variável de facilitação sináptica
D	Variável de depressão sináptica
τ_D	Constante de tempo da depressão sináptica
τ_F	Constante de tempo da facilitação sináptica
F_{fat}	Grau de facilitação sináptica
F_{max}	Valor máximo de facilitação sináptica
r_n	Taxa de disparo da unidade n
W^{EE}	Força de conexões excitatórias
W^{IE}	Força de conexões inibitórias
s^E	Fração de canais sinápticos excitatórios abertos
s^I	Fração de canais sinápticos inibitórios abertos
r_e	Taxa de disparos de subpopulações excitatórias no modelo de Wilson-Cowan
r_i	Taxa de disparos de subpopulações inibitórias no modelo de Wilson-Cowan
$T_e(t)$	Entrada da unidade de subpopulação excitatória no modelo de Wilson-Cowan

$T_i(t)$	Entrada da unidade de subpopulação inibitória no modelo de Wilson-Cowan
τ_e	Constante de tempo da subpopulação excitatória no modelo de Wilson-Cowan
τ_i	Constante de tempo da subpopulação inibitória no modelo de Wilson-Cowan
F_e	Função de ativação para a subpopulação excitatória no modelo de Wilson-Cowan
F_i	Função de ativação para a subpopulação inibitória no modelo de Wilson-Cowan
τ_w	Constante de tempo de atualização de pesos sinápticos
$\mathbf{w}$	Vetor de pesos sinápticos
$\mathbf{u}$	Vetor de atividades dos neurônios pré-sinápticos
A_+	Valor máximo de modificação sináptica para potencialização de longa duração
A_-	Valor máximo de modificação sináptica para depressão de longa duração
τ_+	Faixa de intervalo entre disparo das células pré para pós-sinápticas na potencialização de longa duração
τ_-	Faixa de intervalo entre disparo das células pré para pós-sinápticas na depressão de longa duração
x_k	Resposta da sinapse k no modelo do *Perceptron*
w_k	Peso da sinapse no *Perceptron*
b	*Bias*
σ	Função de ativação não-linear do *Perceptron*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto

Estudar e entender o funcionamento do cérebro é de grande interesse, devido à sua complexidade. Além das técnicas tradicionais de estudos em neurociência, simulações computacionais dos neurônios e suas conexões são de grande importância, daí o surgimento da neurociência computacional, voltado a auxiliar na compreensão dos mecanismos cerebrais utilizando o computador. Por isso, este trabalho propõe a criação de um roteiro para um curso que possa introduzir a área de neurociência computacional para alunos de graduação. Além da introdução teórica, a parte prática também é apresentada, utilizando uma linguagem de programação livre para a simulação de diversos comportamentos observados no cérebro.

Alguns trabalhos com introduções semelhantes são encontrados na literatura. Dayan e Abbott (2001) apresentam uma extensa revisão teórica dos conteúdos-chave de neurociências. Ermentrout e Terman (2010) apresentam uma base voltada para os fundamentos matemáticos. Miller (2018) apresenta uma introdução à neurociência computacional, porém utilizando o Matlab como ferramenta de programação, que é um *software* pago. Um diferencial deste trabalho é o uso de ferramentas livres, podendo ser executadas gratuitamente, inclusive online.

1.2 Objetivos

1.2.1 Objetivo Geral

Elaborar um roteiro para execução de um curso introdutório em neurociências computacionais usando linguagens de programação livres.

1.2.2 Objetivos Específicos

- Obter uma bibliografia robusta para servir de base na elaboração do roteiro;
- Criar um conjunto de códigos contendo exemplos dos conceitos apresentados ao longo do roteiro;

- Consolidar o material criado em uma estrutura de fácil uso por interessados no tema em questão.

1.3 Metodologia

Um extenso referencial teórico é utilizado para apresentação dos diversos tópicos do curso. Na parte prática, códigos em Python, que implementam as teorias apresentadas, são disponibilizados e apresentados, com ênfase na relação entre os trechos de código e as partes teóricas associadas. A execução dos códigos é feita utilizando uma ferramente *online* e gratuita, possibilitando a execução do curso sem qualquer tipo de instalação associada.

1.4 Estrutura do Trabalho

O trabalho está estruturado da seguinte maneira: o Capítulo 2 mostra os elementos da base teórica apresentada. Uma breve apresentação de definições sobre neurobiologia, equações diferenciais ordinárias, probabilidade, noções sobre algoritmos e linguagem de programação são mostradas. O Capítulo 3 fala dos modelos de neurônios de disparo mais conhecidos. O Capítulo 4 fala sobre as conexões entre neurônios, incluindo aprendizado e plasticidade de longa duração. O Capítulo 5 mostra as redes neurais e neuromórficas. Finalmente, o Capítulo 6 contém as conclusões do trabalho e seus possíveis desdobramentos.

2 BASE TEÓRICA

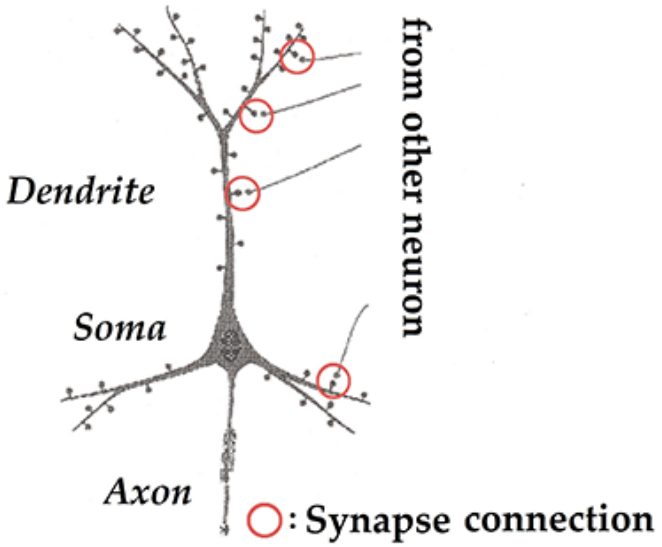
2.1 Introdução

Dada a variedade de estudantes que podem ingressar em um curso de neurociência computacional, muitos deles podem não ter tido algumas disciplinas de base que são necessárias para um bom entendimento do conteúdo apresentado. Por isso, este capítulo contém uma base teórica útil para um nivelamento dos estudantes das mais diversas áreas. As sessões seguintes contemplam conteúdos de neurobiologia e equações diferenciais ordinárias.

2.2 Neurobiologia básica

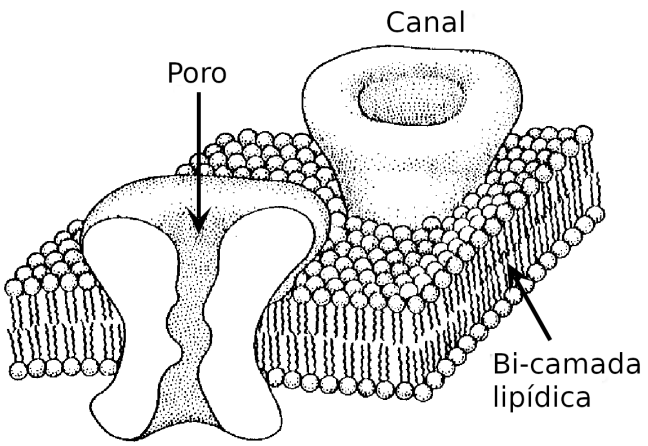
O neurônio, mostrado na Figura 1, é a unidade básica do sistema nervoso. É composto pelo corpo celular (chamado de soma), dendritos, que recebem sinais de outros neurônios, axônio, que transmite os sinais a outros neurônios, e terminais axônicos, que são as extremidades do axônio e se conectam com os dendritos de outros neurônios. Como outras células do corpo humano, o neurônio é composto por íons e moléculas, ambos podendo possuir cargas positivas ou negativas. A célula neuronal é revestida por uma bi-camada lipídica, e geralmente o interior dela possui uma maior concentração de cargas negativas, fazendo com que o potencial de membrana (V_m), que é a diferença de potencial entre a parte interna e a externa da célula

Figura 1 – Neurônio



Fonte: Adaptado de (KITAJIMA; FENG; AZHIM, 2016)

Figura 2 – Membrana do neurônio



Fonte: adaptado de (HILLE, 1992)

neuronal ($V_M = V_{dentro} - V_{fora}$), fique a maior parte do tempo com valor negativo. O potencial de membrana se altera quando há o fluxo de íons através dos poros de canais iônicos (Figura 2).

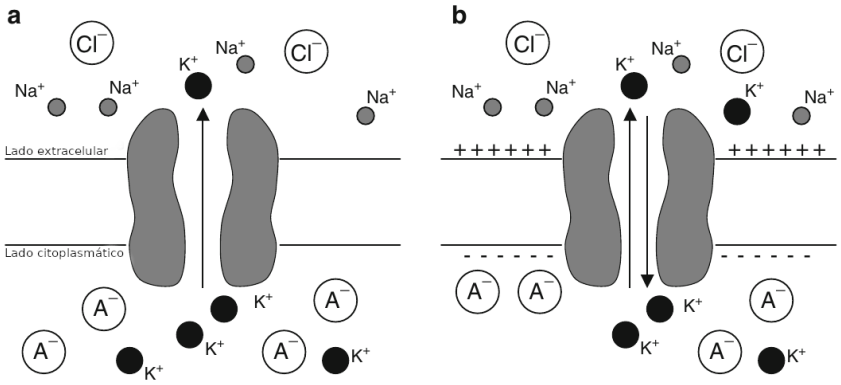
Os canais iônicos são canais proteicos na membrana da célula neuronal e permitem a movimentação de íons através deles, como mostrado na Figura 3. Podem ser de dois tipos: com ou sem portão. Canais sem portão estão sempre abertos, enquanto os com portão podem abrir ou fechar, dependendo do valor do potencial de membrana, e por isso são chamados de canais iônicos dependentes de tensão. Quando o fluxo de corrente elétrica de todos os íons é equilibrado dentro e fora da célula neuronal (ou seja, o potencial de membrana não se altera), ele é chamado de potencial de repouso (ou de equilíbrio). O valor típico é próximo de $-70\ mV$. O valor do potencial de repouso (E_A) é calculado com base na concentração de cada íon dentro e fora da célula neuronal usando a equação de Nernst, dada por:

$$E_A = \frac{k_B T}{z_A q_e} ln\left(\frac{A_{fora}}{A_{dentro}}\right) \tag{2.1}$$

sendo A o íon, z_A a carga do íon, A_{fora} e A_{dentro} a concentração desse íon fora e dentro da célula neuronal, respectivamente, T a temperatura absoluta (em Kelvin), k_B a a constante de *Boltzmann* ($1,39\text{x}10^{-23}\ JK^{-1}$) e q_e a carga elétrica fundamental ($1,6\text{x}10^{-19}\ C$). Os valores de cargas, concentração e potencial de Nernst para alguns íons são apresentados na Tabela 1.

Fora do repouso, a diferença de potencial entre o interior e o exterior do neurônio produz movimentação iônica. Quando íons positivos (como Na^+ ou Ca^{2+}) entram na célula neuronal, o potencial de membrana fica menos negativo (até próximo de 0 mV), fenômeno conhecido como despolarização. De maneira semelhante, quando íons positivos (como K^+)

Figura 3 – Canais iônicos de potássio. Em a os íons de potássio saem da célula, causando um excesso de cargas positivas fora e negativas dentro. Em b o fluxo para fora e dentro é igual, causando equilíbrio



Fonte: Adaptado de (ERMENTROUT; TERMAN, 2010)

Tabela 1 – Concentração de íons

Íon	Carga	Conc. interna (nM)	Conc. externa (nM)	Potencial de reversão (mV)
Sódio	+1	15	120	61,6
Potássio	+1	150	6	-86,1
Cloreto	-1	52	560	-65
Cálcio	+2	50	2	141,7

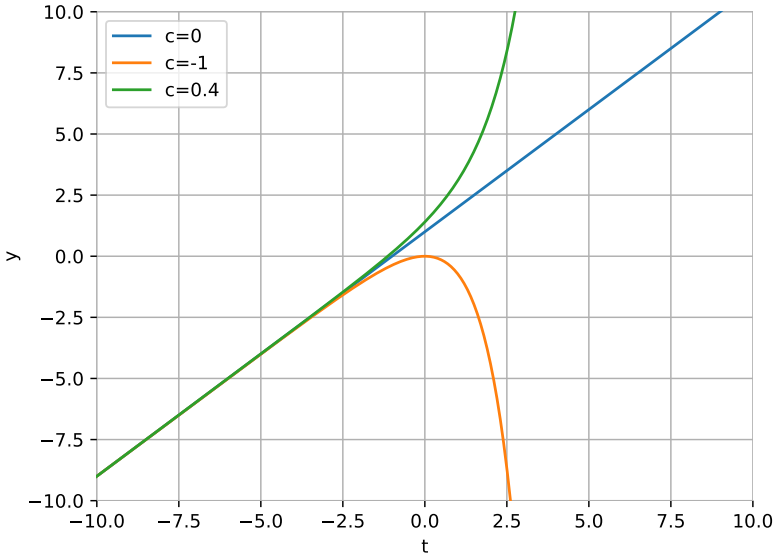
Fonte: O autor (2023)

saem da célula neuronal, ou negativos (como Cl^-) entram, o potencial de membrana fica mais negativo, fenômeno conhecido como hiperpolarização. Os canais de potássio são mais presentes na membrana do neurônio, sendo os principais responsáveis pelo potencial de equilíbrio. Já os canais de sódio, associados à despolarização, são responsáveis pela geração do potencial de ação. Devido ao excesso de cargas negativas no interior da célula neuronal, e a existência de cargas positivas no exterior, a membrana do neurônio acaba funcionando como um capacitor, criando uma capacitância (C_m). O fluxo de íons através de canais sem portão é considerado constante, podendo ser agrupado em um elemento chamado de vazamento (*leak*, em inglês), possuindo um potencial (E_l) e uma condutância (G_l), que é a facilidade desse elemento permitir o fluxo de corrente (o inverso da resistência). Com isso, é possível escrever uma equação relacionando o potencial de membrana da célula neuronal e os elementos de vazamento, como segue:

$$\frac{\mathrm{d}V_m}{\mathrm{d}t} = G_l(E_l - V_m)/C_m \tag{2.2}$$

que é dada na forma de uma equação diferencial, detalhada na seção seguinte.

Figura 4 – Soluções $y(t) = t + 1 + ce^t$ da equação $y' = y - t$ para vários valores de c



Fonte: O autor (2023)

2.3 Equações diferenciais ordinárias

As equações diferenciais são usadas para descrever sistemas dinâmicos, ou seja, que se alteram ao longo do tempo. A forma mais básica de uma equação diferencial é:

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = F(t, y(t)) \tag{2.3}$$

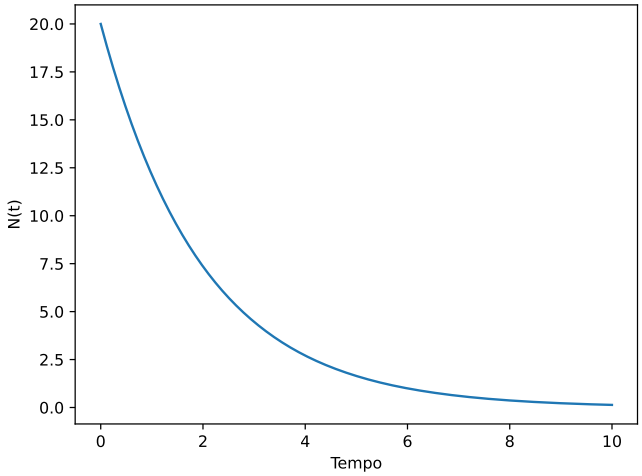
com o lado esquerdo da equação sendo a derivada de $y(t)$, a função presente no lado direito associada à variável t, que geralmente representa o tempo, e F representando uma função qualquer com t e y. Diferentemente das equações algébricas, que possuem como solução um número, as equações diferenciais tem uma família de funções como solução, variando de acordo com a escolha de constantes, como mostrado na Figura 4. Nela, é exibida a família de soluções $y(t) = t + 1 + ce^t$ para a equação diferencial $y' = y - t$. y' é uma representação equivalente para $\frac{\mathrm{d}y(t)}{\mathrm{d}t}$. Qualquer valor de c real que for escolhido representa uma solução possível para a equação, sendo exibidas as curvas para os valores de c iguais a 0, -1 e $0,4$.

2.3.1 Exemplos

2.3.1.1 Decaimento radioativo

Segundo a lei do decaimento radioativo, a taxa na qual os átomos radioativos se desintegram é proporcional ao número total de átomos radioativos presente. Sendo $N(t)$ o número de

Figura 5 – Decaimento radioativo ($\lambda = 0,5$)



Fonte: O autor (2023)

átomos radioativos no tempo t, então $N'(t)$ é a taxa de mudança. A lei do decaimento radioativo é a que segue:

$$N'(t) = -\lambda N(t) \tag{2.4}$$

onde λ é a constante de decaimento. O gráfico de uma possível solução para essa equação diferencial é exibido na Figura 5.

2.3.1.2 Equações de Lotka-Volterra

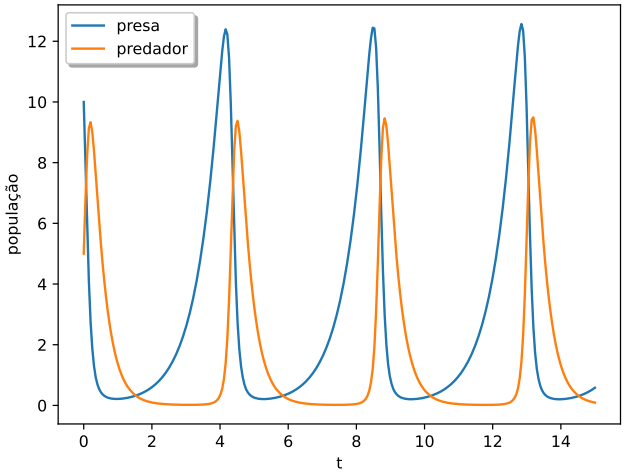
Também conhecidas como equações predador-presa, são um par de equações diferenciais de primeira ordem, frequentemente usadas para descrever a dinâmica de sistemas biológicos de interação entre duas espécies, uma como predadora e a outra como presa. As populações de cada uma das espécies são dadas pelo par de equações:

$$x' = ax - bxy$$

$$y' = dxy - cy \tag{2.5}$$

sendo x a população da presa, y a do predador, x', y' as taxas de variação de cada população, e a, b, c, d os parâmetros que descrevem a interação entre as espécies. A Figura 6 exibe a solução do par de equações para um valor de cada parâmetro de interação.

Figura 6 – Sistema de Lotka-Volterra (a = 1,5; b = 1; c = 3; d = 1)



Fonte: O autor (2023)

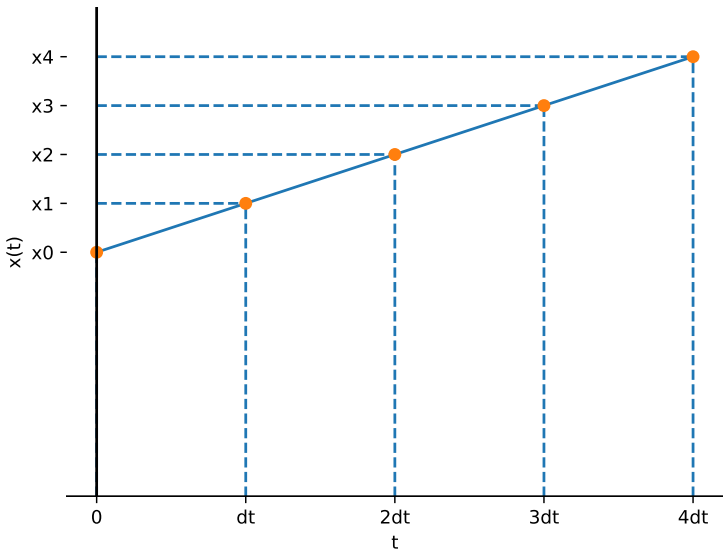
2.3.2 Método de Euler

Equações diferenciais ordinárias podem ser resolvidas analiticamente, não abordado neste texto, ou numericamente. Dentre os vários métodos existentes para a solução numérica, a adotada aqui é o método de Euler. Considerando a equação $\frac{dx}{dt} = f(x,t)$, com $f(x,t)$ uma função qualquer de x em relação a t, dado um valor inicial $x0$ (usualmente com $t = 0$), é possível simular a equação usando pontos discretos com intervalos Δt fixos. Cada valor x_n é dado por $x_n = x(t_n = n\Delta t)$. A partir disso, é possível usar o método de Euler avançado para calcular um valor seguinte a partir do valor anterior, ou seja:

$$x_{n+1} = x_n + f(x_n, t_n)\Delta t$$

como é demonstrado na Figura 7. Outros métodos não abordados no curso incluem o método de Euler reverso e o Runge-Kutta de segunda e quarta ordens, que são mais precisos na solução. O algoritmo (sequência de instruções para executar uma determinada tarefa) do método de Euler será implementado usando a linguagem Python, que tem a sua base detalhada no Apêndice A.

Figura 7 – Método de Euler



Fonte: O autor (2023)

3 MODELOS DE NEURÔNIO DE DISPARO

3.1 Introdução

Os modelos são formas de representar, matemática e/ou computacionalmente, o comportamento do neurônio. Podem variar desde os mais simples, que não representam fielmente o comportamento fisiológico do neurônio mas são úteis para simular grandes quantidades de neurônios, até os mais complexos, que trazem diversas representações de condutância e morfológicas. Alguns modelos de neurônio são apresentados na Tabela 2, incluindo o número de variáveis presentes no modelo, o que reflete o número de equações diferenciais presentes e, consequentemente, a complexidade do mesmo, bem como se o modelo é considerado biologicamente plausível. Os 4 (quatro) primeiros modelos são detalhados neste texto, por ora diferenciados entre os que simulam o disparo do potencial de ação, chamados aqui de modelos de neurônio de disparo, e os que simulam a taxa de disparo, apresentados no Capítulo 4. Há, ainda, distinção, por exemplo, entre modelos de compartimento único, que modelam o potencial de membrana apenas pela variável V, e os multi-compartimento, que consideram variações espaciais no potencial de membrana, porém estes últimos não são abordados neste texto.

3.2 Modelo integra-e-dispara com vazamento

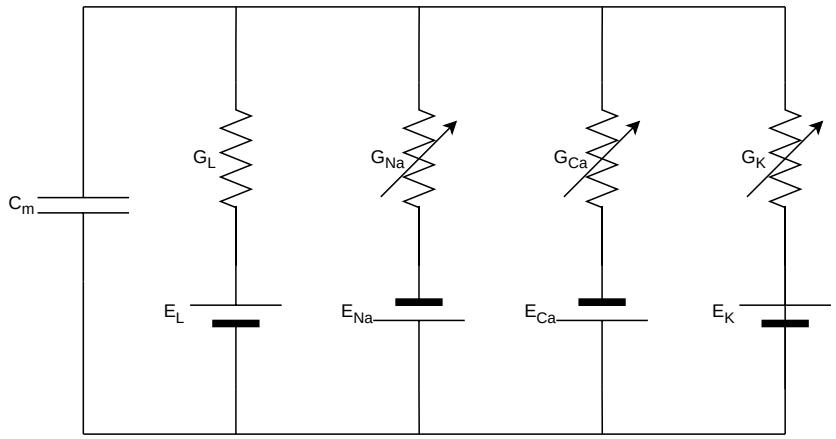
Para modelos de neurônio de compartimento único, o circuito elétrico equivalente da membrana pode ser representado como na Figura 8. O capacitor na esquerda está associado à capacitância da membrana (C_m), e cada resistor em série com uma bateria representa um canal iônico. A série com o L subscrito refere-se ao elemento de vazamento, citado anteriormente, e as séries são referentes aos canais iônicos dependentes de tensão, com o íon subscrito. O circuito

Tabela 2 – Modelos de neurônio

Modelo	N. variáveis	Complexidade	Biol. plausível
Leaky integrate-and-fire	1	Muito baixa	Não
Izhikevich	2	Muito baixa	Não
Hodgkin-Huxley	4	Muito alta	Sim
Wilson-Cowan	2	Média	Não
Spike response model	1	Baixa	Não
FitzHugh-Nagumo	2	Média	Não
Moris-Lecar	3	Alta	Sim

Fonte: O autor (2023)

Figura 8 – Circuito equivalente da membrana



Fonte: O autor (2023)

completo tem associada a seguinte equação:

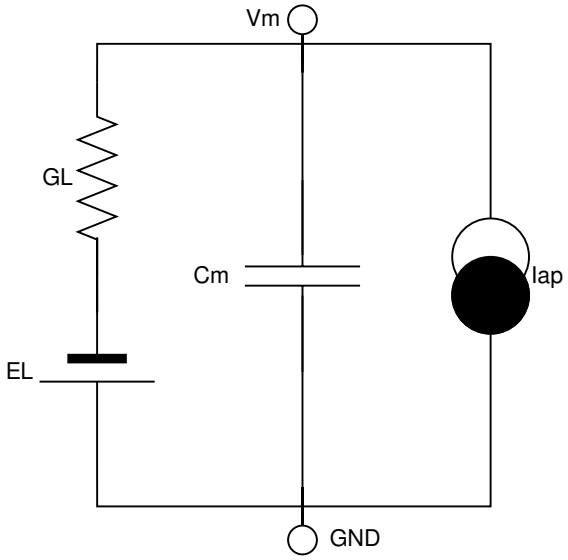
$$C_m \frac{\mathrm{d}V_m}{\mathrm{d}t} = G_{Na}(E_{Na} - V_m) + G_{Ca}(E_{Ca} - V_m) + G_K(E_K - V_m) + G_L(E_L - V_m) \tag{3.1}$$

sendo G_A a condutância do íon A, E_A o potencial do íon A. Ignorando por ora os canais iônicos dependentes de tensão, a equação fica como a 2.2. Adicionando-se a corrente aplicada, incluímos também uma condição que força o disparo do potencial de ação quando o potencial de membrana atinge um determinado limitar (V_{th}), forçando o potencial de membrana para um valor de *reset* (V_{reset}) (MILLER, 2018). Com isso, o circuito fica equivalente à Figura 9 e a equação total é como abaixo.

$$C_m \frac{dV_m}{dt} = G_L(E_L - V_m) + I_{ap}, \qquad \text{se } V_m > V_{th} \text{ então } V_m \to V_{reset} \tag{3.2}$$

Essa é a equação do modelo *leaky integrate-and-fire* (LIF, integra e dispara com vazamento, em tradução livre) (LAPICQUE, 1907). A corrente aplicada no modelo é acumulada (integrada), elevando o potencial de membrana até o valor de limiar, representando o momento onde o neurônio dispara. Devido à simplicidade do modelo, a equação, por si só, não é capaz de representar a hiperpolarização que ocorre fisicamente na célula neuronal após o disparo do potencial de ação, e, por isso, é acrescentada a condição de *reset*. Enquanto a corrente continuar sendo aplicada, o potencial de membrana permanece sendo atualizado pela dinâmica da equação diferencial, como é exibido na Figura 10. Três pulsos de corrente, com valores 0,18 nA, 0,21 nA e 0,24 nA, são aplicados no modelo. Para o primeiro valor, o neurônio despolariza (fica menos negativo), porém não atinge o valor de limiar (como visto na primeira curva da segunda linha de gráficos). Com a injeção dos demais valores de corrente, a célula neuronal atinge o valor de

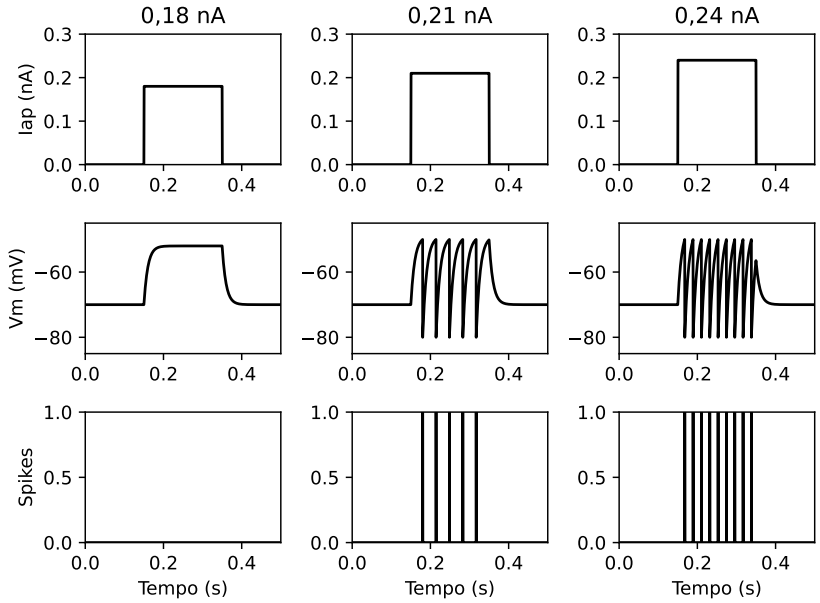
Figura 9 – Circuito equivalente do modelo LIF



Fonte: O autor (2023)

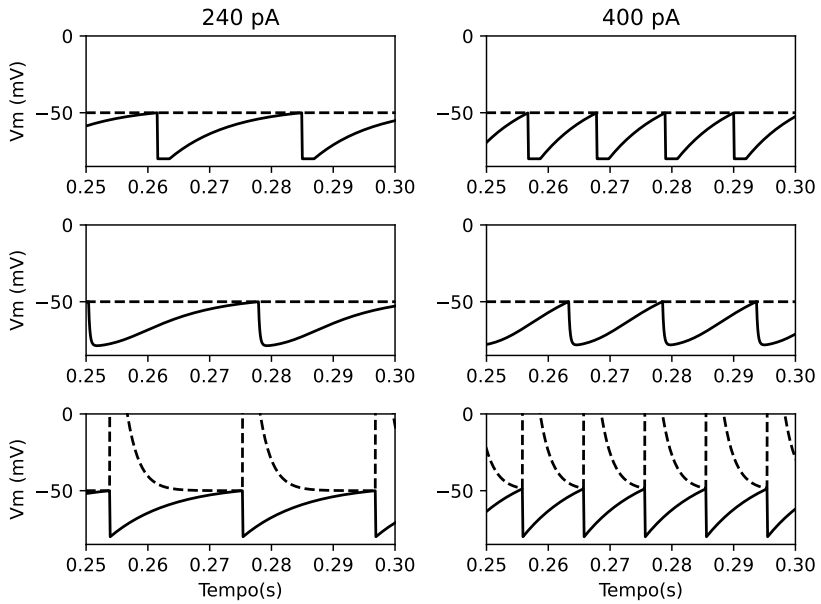
limiar (50 mV nestes exemplos), ocorrendo o disparo do potencial de ação (registrado como uma linha vertical na última linha de gráficos), e com a posterior alteração do potencial de membrana para o valor de *reset* (80 mV nestes exemplos).

Figura 10 – Exemplo da simulação com o modelo LIF



Fonte: O autor (2023)

Figura 11 – Simulação do modelo LIF com período refratário



Fonte: O autor (2023)

3.2.1 Extensões do modelo LIF

Apesar de útil e ser utilizado em inúmeras simulações neuronais, o modelo LIF é carente de diversos comportamentos presentes fisiologicamente no neurônio. Alguns deles são apresentadas aqui como extensões ao modelo, a fim de acrescentar algumas características que podem ser úteis nas simulações. As extensões apresentadas simulam a refratariedade e a adaptação da taxa de disparo, além de versões do modelo LIF com componentes exponenciais e adaptativas, como mostradas na sequência.

3.2.1.1 Período refratário

Imediatamente após a ocorrência do potencial de ação, o neurônio não é capaz de produzir um novo potencial de ação durante um curto período de tempo, e esse tempo é chamado de período refratário. Existem diferentes métodos para simular esse comportamento nos modelos neuronais, e aqui mostramos três, acrescentadas ao modelo LIF, com os seus comportamentos exibidos na Figura 11. Em todos os métodos são simuladas a injeção de corrente de 240 pA (primeira coluna de gráficos) e 400 pA (segunda coluna), e a linha tracejada representa o limiar para disparo do potencial de ação. O primeiro método é chamado de **grampeamento de tensão**, que consiste em fixar o potencial de membrana no valor de *reset* durante um tempo específico logo após o potencial de ação. É, provavelmente, a maneira mais simples de simular

a refratariedade, pois neste método o período refratário é constante. O segundo método é o da **condutância refratária**, consistindo no acréscimo de uma condutância elevada, geralmente uma corrente hiperpolarizante de potássio, que cresce imediatamente após a ocorrência de cada potencial de ação, e decai em seguida de acordo com a seguinte equação diferencial:

$$\frac{dG_{ref}(t)}{dt} = -\frac{G_{ref}(t)}{\tau_{ref}}, \qquad \text{depois do potencial de ação } G_{ref} \rightarrow G_{ref} + \Delta G \tag{3.3}$$

sendo τ_{ref} a constante de tempo do período refratário e ΔG o incremento de condutância.

Como visto na segunda linha de gráficos, o potencial de membrana cresce em um ritmo mais lento que o normal, devido à corrente de potássio que hiperpolariza a célula. Essa corrente é representada acrescentando-se à parte direita da equação do modelo LIF o termo $G_{ref}(t)[E_k - V_m(t)]$, com E_k sendo o potencial de reversão para os íons de potássio (como na Tabela 1). O último método é chamado de **incremento do limiar**, onde é elevado, após cada potencial de ação, o limiar necessário para ocorrer um novo disparo. Esse limiar decai conforme a equação:

$$\frac{dV_{th}(t)}{dt} = \frac{V_{th}^0 - V_{th}(t)}{\tau_{ref}}, \qquad \text{depois do potencial de ação } V_{th} \rightarrow V_{th} + \Delta V \tag{3.4}$$

sendo V_{th}^0 o potencial de limiar de referência (o valor do limiar em repouso).

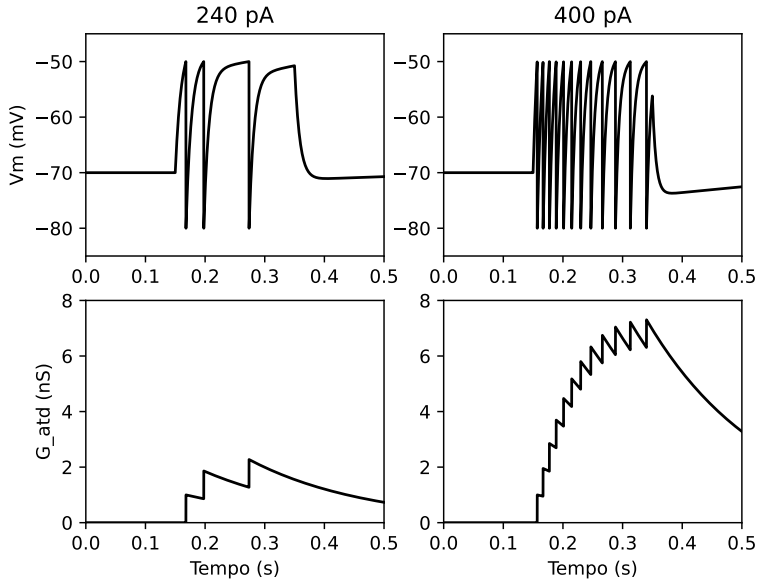
Como visto na última linha de gráficos, o limiar (linha tracejada) cresce absurdamente após cada disparo de potencial de ação, decaindo logo em seguida, ao contrário do potencial de membrana, que decai e cresce em seguida.

3.2.1.2 Adaptação da taxa de disparos

Uma característica presente nos neurônios é a sua capacidade de adaptar a frequência em que os disparos de potencial de ação ocorrem, com uma diminuição da taxa logo após o primeiro disparo. Uma fácil percepção da consequência da adaptação da taxa de disparo é a diferença de percepção de cheiros intensos ao longo do tempo, como quando uma pessoa entra com um perfume forte em um elevador. A implementação da adaptação da taxa de disparos é feita de maneira semelhante ao método da condutância refratária apresentada anteriormente, porém com duas diferenças:

- O incremento da condutância é menor em comparação ao do período refratário. Esse incremento menor não impede o disparo de novos potenciais (o que acontece no período refratário), porém diminui a taxa deles;

Figura 12 – Adaptação da taxa de disparo no neurônio LIF



Fonte: O autor (2023)

- A escala de tempo (a constante) da condutância adaptativa é bem maior, o que permite o acúmulo das condutâncias ao longo da sequência de disparos.

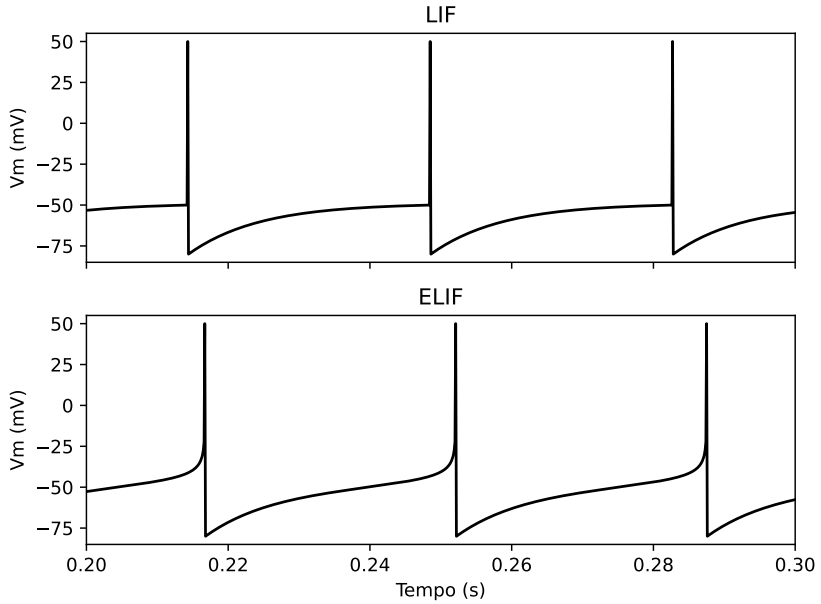
A dinâmica da adaptação da taxa de disparos é exibida na Figura 12. Como no exemplo do período refratário, correntes de 240 pA e 400 pA são injetadas, sendo exibidas as curvas de potencial de membrana (em cima), e da condutância adaptativa (embaixo), onde se pode observar incremento e acúmulo de condutância.

3.2.1.3 Modelo LIF exponencial

O modelo LIF exponencial (*exponential leaky integrate-and-fire*, ELIF, em inglês) incorpora um termo adicional para a geração do potencial de ação, que é uma falta existente no modelo tradicional. Esse termo acrescenta uma corrente despolarizante, elevando quase que instantaneamente o valor do potencial de membrana quando este se aproxima do limiar. Esse crescimento tende ao infinito, porém nas simulações é definido um limite (V_{max}), devido o computador ter problemas para lidar com valores infinitos, alterando a condição do modelo para a equação abaixo:

$$\text{se } V_m > V_{max} \text{ então } V_m \rightarrow V_{reset} \tag{3.5}$$

Figura 13 – Comparação dos modelos LIF e ELIF



Fonte: O autor (2023)

Além disso, não há exatamente um valor fixo para o disparo do potencial, e sim um intervalo (Δ_{th}), e a equação completa deste modelo fica:

$$c_m \frac{dV_m}{dt} = G_L(E_L - V_m) + G_L \Delta_{th} \exp\left(\frac{V_m - V_{th}}{\Delta_{th}}\right) + I_{ap} \tag{3.6}$$

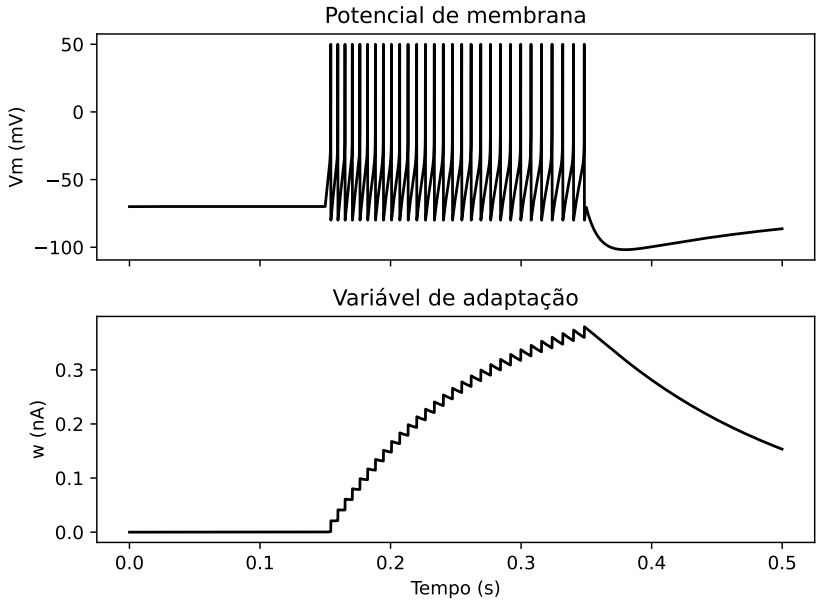
Os gráficos comparando o potencial de membrana do modelo LIF (em cima) e ELIF (embaixo) são mostrados na Figura 13. Uma característica interessante é a inflexão presente no modelo ELIF quando o potencial de membrana se aproxima do instante em que ocorre o disparo do potencial de ação, algo inexistente no modelo LIF.

3.2.1.4 Modelo LIF exponencial adaptativo

O modelo LIF exponencial adaptativo (*adaptative exponential leaky integrate-and-fire*, AELIF, em inglês) adiciona uma corrente adaptativa hiperpolarizante ao modelo anterior, similar à utilizada no método da condutância refratária, porém com o decaimento dependendo do potencial de membrana. É um modelo com duas equações, sendo uma para a variável de adaptação (w) e a outra para o potencial de membrana, que é semelhante à do modelo anterior, e ambas têm um *reset* quando ocorre o disparo do potencial de ação. As equação para este modelo são:

$$c_m \frac{dV_m}{dt} = G_L(E_L - V_m) + G_L \Delta_{th} \exp\left(\frac{V_m - V_{th}}{\Delta_{th}}\right) - w + I_{ap} \tag{3.7}$$

Figura 14 – Resposta do modelo AELIF para um pulso de corrente de 1 *nA*



Fonte: O autor (2023)

$$\tau_w \frac{dw}{dt} = a(V_m - E_L) - w \tag{3.8}$$

com as condições:

$$\text{se } V_m > V_{max} \text{ então } V_m \rightarrow V_{reset} \text{ e } w \rightarrow w + b \tag{3.9}$$

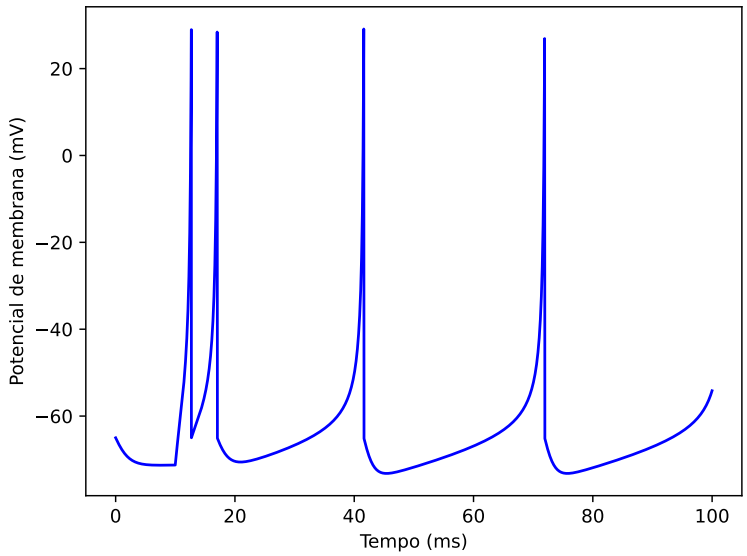
sendo a o parâmetro de agrupamento, indicando o nível de relação entre o potencial de membrana e a variável de adaptação, e b o parâmetro de adaptação, que é o valor do incremento de corrente após o disparo do potencial de ação.

A dinâmica do modelo AELIF é mostrada na Figura 14, com o potencial de membrana exibido em cima e a variável de adaptação embaixo. É possível notar a semelhança da curva de adaptação com a da condutância adaptativa da adaptação da taxa de disparo, também semelhante ao método da condutância refratária.

3.3 Modelo de Izhikevich

Em 2003, Eugene Izhikevich publicou um modelo capaz de simular vários comportamentos de neurônios corticais, combinando a plausabilidade biológica do modelo de Hodgkin-Huxley (que será visto na sequência) com a eficiência computacional do modelo LIF (IZHIKEVICH,

Figura 15 – Potencial de membrana no modelo de Izhikevich



Fonte: O autor (2023)

2003). Ele é composto por duas equações diferenciais, que são as seguintes:

$$v' = 0,04v^2 + 5v + 140 - u + I \tag{3.10}$$

$$u' = a(bv - u) \tag{3.11}$$

com as condições de *reset*

$$\text{se } v \geq 30 \text{ mV, então } \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \tag{3.12}$$

com v sendo o potencial de membrana, u a variável de recuperação da membrana, e I sendo a corrente injetada (a notação adotada aqui é a mesma usada por Izhikevich).

Na Figura 15 é exibido o potencial de membrana para um comportamento do tipo *regular spiking* quando é injetada uma corrente de 10 mA. Os parâmetros a, b, c e d são definidos como segue:

- O parâmetro a representa a escala de tempo da variável de recuperação, onde valores pequenos resultam em uma recuperação mais lenta;
- O parâmetro b representa a sensibilidade da variável de recuperação às sub-flutuações do potencial de membrana, onde valores maiores agrupam mais v e u;

Tabela 3 – Padrões de neurônio do modelo de Izhikevich

Padrão	a	b	c	d
regular spiking	0,02	0,2	-65	8
intrinsically bursting	0,02	0,2	-55	4
chattering	0,02	0,2	-50	2
fast spiking	0,1	0,2	-65	2
thalamo-cortical	0,02	0,25	-65	0,05
resonator	0,1	0,25	-65	2
low-threshold spiking	0,02	0,25	-65	2

Fonte: O autor, baseado em (IZHIKEVICH, 2003)

- O parâmetro *c* representa o valor de *reset* do potencial de membrana após o disparo do potencial de ação;
- O parâmetro *d* representa o valor de *reset* da variável de recuperação após o disparo do potencial de ação.

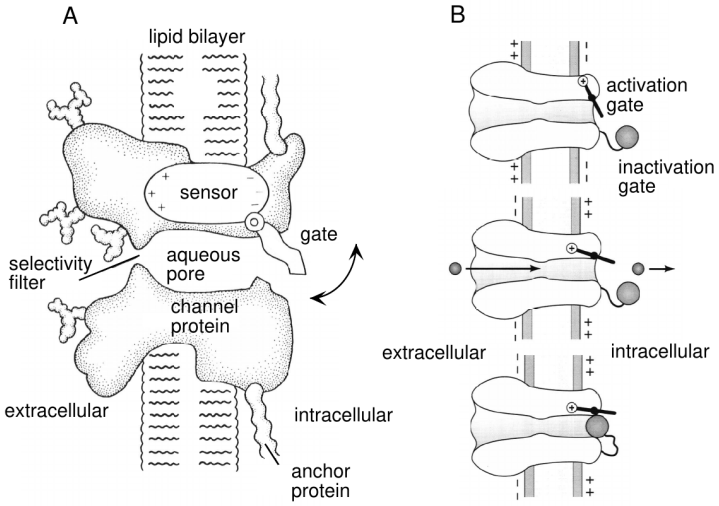
A combinação de determinados valores para os parâmetros acima produz saídas compatíveis com alguns dos comportamentos citados anteriormente, conforme listado na Tabela 3.

3.4 Modelo de Hodgkin-Huxley

O último dos modelos de disparo de neurônio visto neste texto é o de Hodgkin-Huxley (HODGKIN; HUXLEY, 1952). Como visto na Seção 2.2, existem canais iônicos que são dependentes de tensão. A probabilidade de abertura desses canais é modificada em função do potencial de membrana. O mecanismo de funcionamento deles é como mostrado na Figura 16. À esquerda, é possível ver os elementos que compõem esse tipo de canal: um sensor, que identifica o potencial de membrana; um filtro de seletividade, que seleciona apenas íons compatíveis com aquele canal; e os portões, que permitem ou não a passagem dos íons. Há dois tipos de portões, que estão relacionados aos processos de **ativação** e **inativação**. O primeiro refere-se à abertura de um canal iônico, geralmente devido à despolarização, que aumenta a probabilidade de um canal estar aberto. O seu oposto é a **desativação**, que fecha o canal, geralmente por hiperpolarização. A inativação é o processo que impede a abertura dos canais, tendo como oposto, necessário para que o canal se abra, chamado de **desinativação**. (MILLER, 2018)

Associado aos canais iônicos dependentes de tensão, existe a chamada **variável de portão**, que representa a fração desses canais que se encontra em um determinado estado

Figura 16 – Mecanismo de funcionamento dos canais iônicos dependentes de tensão



Fonte: (DAYAN; ABBOTT, 2001)

(ativado/desativado, inativado/desinativado). Como se trata da probabilidade do canal estar em um desses estados, seus valores são entre 0 e 1. No modelo, são consideradas as variáveis m para ativação de sódio, h para inativação de sódio, e n para ativação de potássio, com as equações a seguir:

$$\frac{dm}{dt} = \alpha_m(1-m) - \beta_m m \tag{3.13}$$

$$\frac{dh}{dt} = \alpha_h(1-h) - \beta_h h \tag{3.14}$$

$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n n \tag{3.15}$$

sendo α a constante de crescimento de cada variável, e β a de decrescimento, ambas também dependentes de tensão, conforme as equações abaixo:

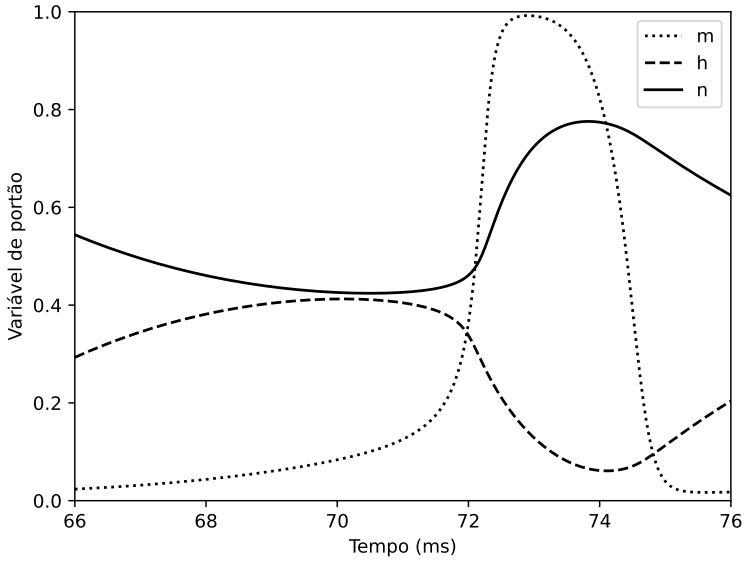
$$\alpha_m = \frac{10^5(-V_m - 0,045)}{\exp[100(-V_m - 0,045)] - 1} \tag{3.16}$$

$$\beta_m = 4\text{x}10^5 \exp\left[\frac{(-V_m - 0,070)}{0,018}\right] \tag{3.17}$$

$$\alpha_h = 70\exp[50(-V_m - 0,070)] \tag{3.18}$$

$$\beta_h = \frac{10^3}{1 + \exp[100(-V_m - 0,040)]} \tag{3.19}$$

Figura 17 – Dinâmica das variáveis de portão



Fonte: O autor (2023)

$$\alpha_n = \frac{10^4(-V_m - 0,060)}{\exp[100(-V_m - 0,060)] - 1} \tag{3.20}$$

$$\beta_n = 125\exp\left[\frac{(-V_m - 0,070)}{0,08}\right] \tag{3.21}$$

A dinâmica das variáveis de portão no modelo de Hodgkin-Huxley pode ser vista na Figura 17, para uma faixa de 66 até 76 ms. As variáveis de portão também podem se definidas pelas suas equações em **regime permanente**, que é o comportamento em repouso depois de um longo período de tempo (ERMENTROUT; TERMAN, 2010), como descritas nas equações abaixo (o símbolo de ∞ representa o valor em regime permanente):

$$m_\infty = \frac{\alpha_m}{\alpha_m + \beta_m} \tag{3.22}$$

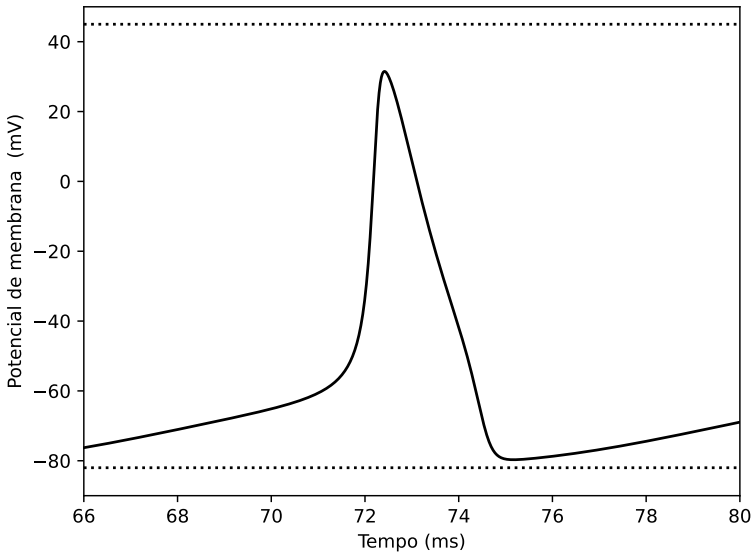
$$h_\infty = \frac{\alpha_h}{\alpha_h + \beta_h} \tag{3.23}$$

$$n_\infty = \frac{\alpha_n}{\alpha_n + \beta_n} \tag{3.24}$$

com as respectivas constantes de tempo:

$$\tau_m = \frac{1}{\alpha_m + \beta_m} \tag{3.25}$$

Figura 18 – Potencial de membrana gerado pelo modelo de Hodgkin-Huxley



Fonte: O autor (2023)

$$\tau_h = \frac{1}{\alpha_h + \beta_h} \tag{3.26}$$

$$\tau_n = \frac{1}{\alpha_n + \beta_n} \tag{3.27}$$

Finalmente, o potencial de membrana é dado pela equação abaixo:

$$C_m \frac{dV_m}{dt} = G_L(E_L - V_m) + G_{Na}^{(max)} m^3 h(E_{Na} - V_m) + G_K^{(max)} n^4 (E_K - V_m) + I_{ap} \tag{3.28}$$

que é semelhante à equação do modelo LIF, porém com dois elementos a mais, um para a condutância dependente de tensão de sódio e outra para a de potássio, que incluem as variáveis de portão. O modelo também assume algumas constantes além das usadas em outros modelos, que são a condutância máxima de sódio ($G_{Na}^{(max)}$), a condutância máxima de potássio ($G_K^{(max)}$), o potencial de reversão do sódio (E_{Na}) e o potencial de reversão do potássio (E_K). A dinâmica do potencial de membrana desse modelo para o mesmo intervalo de tempo é mostrada na Figura 18.

4 CONEXÕES ENTRE NEURÔNIOS

4.1 Introdução

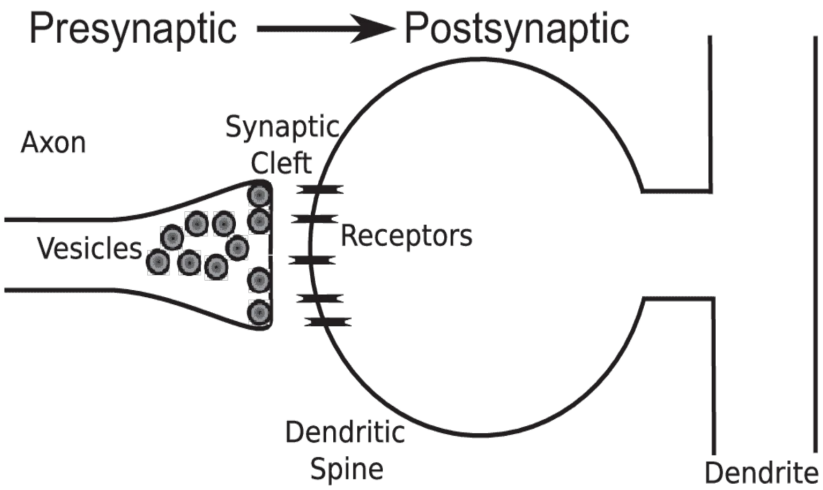
Uma vez visto o comportamento de um neurônio individualmente, este capítulo traz informações sobre como se dá a conexão entre neurônios. Começando pela conexão direta entre dois neurônios, segue-se com alterações possíveis nessas conexões, além de situações onde é simulada a conexão de diversos neurônios, com comportamentos específicos, implementados com base nos neurônios mostrados anteriormente. Segue, ainda, mostrando um modelo que simula diretamente o comportamento do grupo de neurônios, e finaliza com considerações sobre como o cérebro aprende.

4.2 Sinapses

A sinapse é uma conexão entre dois neurônios. Elas podem ser elétricas, que se dão a partir de uma junção entre as duas células, por onde íons podem passar, produzindo uma corrente sináptica proporcional à diferença entre os potenciais de membrana das duas células, ou químicas, que são conectadas através de um espaço entre as células, e a interação entre elas se dá pela liberação de neurotransmissores, que é um elemento químico que se propaga nesse espaço, fazendo a comunicação entre eles. As sinapses químicas são majoritárias no cérebro humano pós-natal, enquanto que as elétricas estão presentes em quantidade relevante apenas durante o período de desenvolvimento embrionário humano. As sinapses químicas são categorizadas em excitatórias ou inibitórias, de acordo com o efeito no potencial de membrana da célula pós-sináptica (a que responde à liberação do neurotransmissor após o potencial de ação do neurônio pré-sináptico). Enquanto as sinapses excitatórias despolarizam a membrana, as inibitórias causam hiperpolarização, facilitando ou dificultando, respectivamente, o potencial de ação da célula pós-sináptica. Os neurotransmissores mais comuns associados aos neurônios corticais são o **glutamato**, que excita a célula pós-sináptica, e o **ácido γ-aminobutírico (GABA)**, que, normalmente, a inibe[1]. O esquema de uma sinapse química é mostrado na Figura 19. Os neurotransmissores são transportados por compartimentos ao redor da membrana neuronal chamados de vesículas.

[1] No cérebro pré-natal o GABA é excitatório (BEN-ARI, 2002)

Figura 19 – Esquema simplificado de uma sinapse química



Fonte: (MILLER, 2018)

A transmissão sináptica é modelada usando a equação abaixo (DAYAN; ABBOTT, 2001):

$$\frac{dG_{sin}(t)}{dt} = \frac{-G_{sin}(t)}{\tau_{sin}} \tag{4.1}$$

sendo $G_{sin}(t)$ a condutância sináptica e τ_{sin} a constante de tempo específica para o tipo de sinapse. Além disso, se houver disparo da célula pré-sináptica, ocorre o seguinte mapeamento:

$$G_{sin}(t) \rightarrow G_{sin}(t) + \Delta G, \tag{4.2}$$

onde ΔG é o incremento de condutância.

Um modelo mais elaborado considera o crescimento e decaimento de condutância usando a equação abaixo (MILLER, 2018):

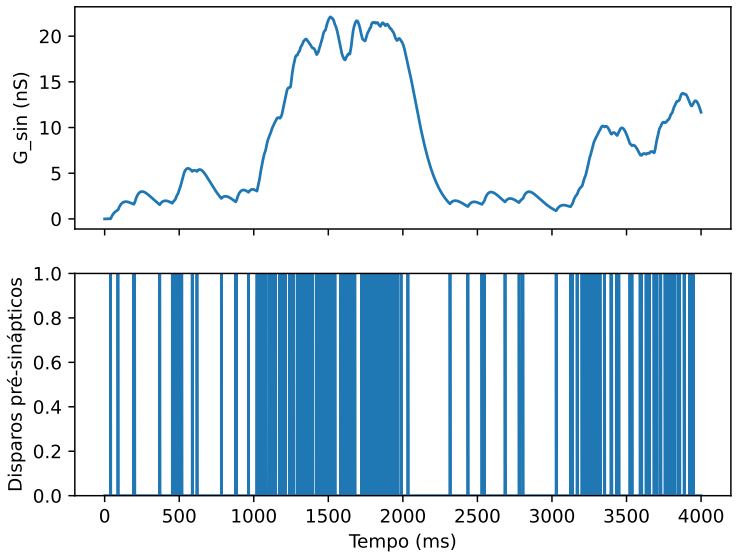
$$\Delta G_{sin}(t) = \frac{\Delta G}{K} e^{-(t-t_{disparo})/\tau_{decai}} \left[1 - e^{-(t-t_{disparo})/\tau_{cresce}}\right], \tag{4.3}$$

com τ_{cresce} sendo a constante de crescimento, τ_{decai} a de decaimento ($\tau_{decai} > \tau_{cresce}$), $t_{disparo}$ o instante do disparo da célula pré-sináptica, e K um fator para garantir que a condutância tenha um valor máximo igual a ΔG, calculado por:

$$K = \left(\frac{\tau_{decai}}{\tau_{cresce} + \tau_{decai}}\right)\left(\frac{\tau_{cresce}}{\tau_{cresce} + \tau_{decai}}\right)^{\tau_{cresce}/\tau_{decai}}. \tag{4.4}$$

A dinâmica da condutância sináptica é mostrada na Figura 20, onde é possível observar o incremento da condutância sináptica (curva de cima) após a ocorrência de um potencial de ação (linhas verticais do gráfico embaixo).

Figura 20 – Condutância sináptica em resposta aos potenciais de ação da célula pré-sináptica



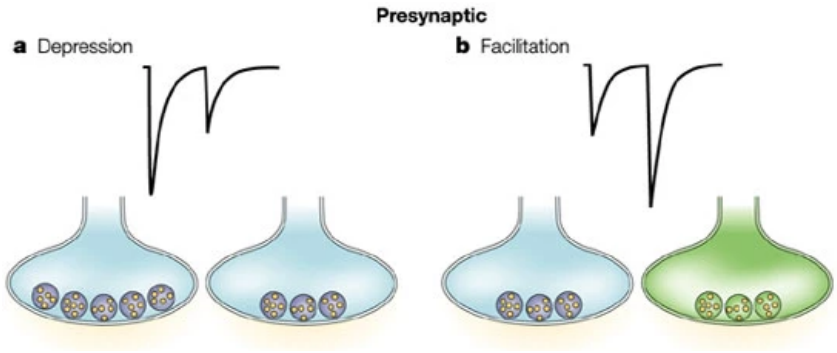
Fonte: O autor (2023)

4.2.1 Sinapses dinâmicas

Acima, os modelos de sinapses possuem pesos fixos. Entretanto, fisiologicamente, as condutâncias sinápticas não são fixas, mas mudam de acordo com a disponibilidade de recursos sinápticos no terminal pré-sináptico, determinados por algumas condições de entrada. Esse comportamento é chamado de sinapse dinâmica, cuja força da conexão varia em um curto período de tempo, ocasionando o a chamada plasticidade de curto prazo (PCP), ou de curta duração. Experimentalmente, são observados dois tipos, com efeitos opostos na eficácia sináptica, como mostrados na Figura 21. À esquerda, é exibido o fenômeno da depressão de curta duração, que causa uma redução temporária na força sináptica, e à direita o fenômeno da facilitação de curta duração, que causa um incremento temporário. O modelo matemático da PCP é fundamentado no conceito de uma quantidade limitada de recursos sinápticos disponíveis para transmissão, tal como a quantidade total de vesículas sinápticas. Esse número de recursos pré-sinápticos muda dinamicamente de acordo com o histórico recente de picos de atividade. Após um pico pré-sináptico, a probabilidade de liberação do conjunto disponível para utilização aumenta devido ao influxo de cálcio induzido pelo pico no terminal pré-sináptico. A simulação de depressão e facilitação sinápticas pode ser feita alterando-se o valor de ΔG acima para:

$$\Delta G = G_{max} p_0 F D \tag{4.5}$$

Figura 21 – Depressão e facilitação sináptica



Fonte: Adaptado de (BLITZ; FOSTER; REGEHR, 2004)

com G_{max} sendo o valor máximo de condutância sináptica, p_0 a probabilidade de liberação do neurotransmissor, e F e D as variáveis de facilitação e depressão sináptica, atualizadas por:

$$\frac{dD}{dt} = \frac{1-D}{\tau_D} \tag{4.6}$$

$$\frac{dF}{dt} = \frac{1-F}{\tau_F} \tag{4.7}$$

e, seguindo cada disparo da célula pré-sináptica, ocorrem os seguintes mapeamentos:

$$D \to D - p_0 F D \tag{4.8}$$

$$F \to F + F_{fat}(F_{max} - F) \tag{4.9}$$

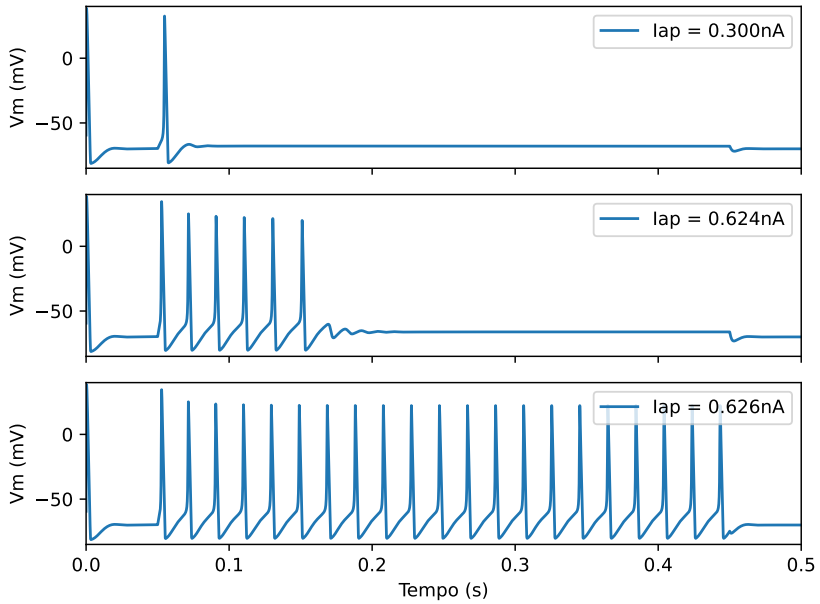
com F_{fat} denotando o grau de facilitação ($0 \leq f_{fat} \leq 1$) e F_{max} o seu valor máximo.

4.3 Multi-estabilidade

4.3.1 Oscilações e multi-estabilidade

Além das alterações de comportamento causadas pelas entradas aplicadas (injeção de corrente), os neurônios podem mudar sua resposta de acordo com as conexões feitas uns com os outros. A capacidade dos neurônios de manter múltiplos estados de atividades mesmo recebendo valores idênticos de entrada é chamada de **multi-estabilidade**. O próprio neurônio de Hodgkin-Huxley, visto anteriormente, pode alternar entre um estado de quiescência, exibindo oscilações sub-limiares (variações no potencial de membrana insuficientes para a produção de um potencial de ação), e um estado de ressonância, que é uma resposta recorrente, conforme mostrado na

Figura 22 – Comportamento variado do modelo de Hodgkin-Huxley para diferentes correntes



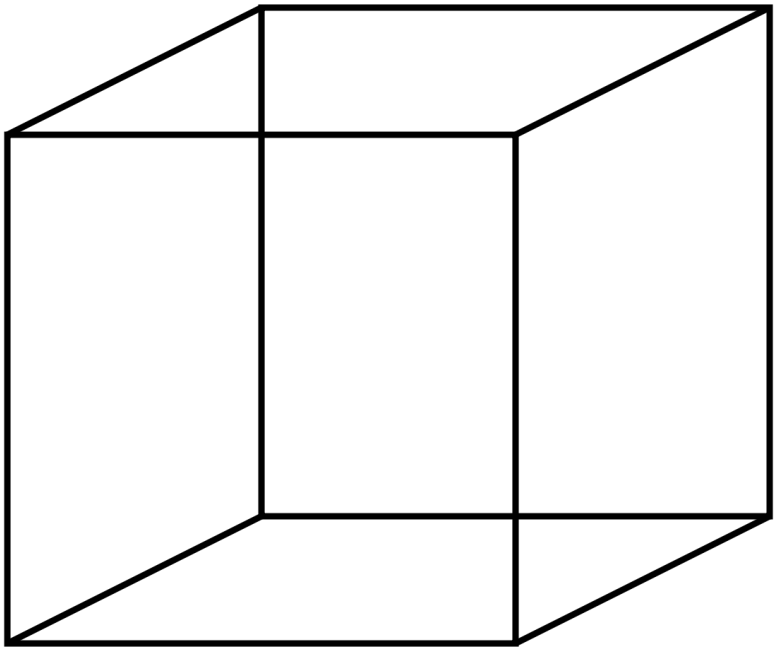
Fonte: O autor (2023)

Figura 22. Nos três cenários é aplicado um pulso de corrente no intervalo de 0,05 até 0,45 s, porém variando a amplitude. É possível observar que, para 0,300 nA (no topo) ocorre apenas um disparo de potencial de ação, com uma oscilação em seguida. Para 0,624 nA (no meio), alguns disparos acontecem, porém não o suficiente para ativar o comportamento ressonante, que é conseguido com um leve incremento de corrente, para 0,626 nA (embaixo).

A forma mais simples de multi-estabilidade é a bi-estabilidade, onde, com um mesmo estímulo, dois estados de atividade neuronal podem ocorrer, gerando a chamada rivalidade perceptual, onde estímulos únicos podem ser percebidos de mais de uma maneira, alterando entre cada um dos perceptos (percepções bi-estáveis), como no exemplo da Figura 23, uma ilusão de ótica onde o cubo, conhecido como Cubo de Necker, pode ser visto com a face frontal em lugares diferentes.

Quando são simuladas as conexões entre um grupo de neurônios, é conveniente considerar a atividade média deles, ou seja, a taxa de disparo dentre eles, e esse conjunto de neurônios é chamado de **unidade**. No caso da bi-estabilidade, por exemplo, duas unidades rivalizam entre si, conforme mostrado na Figura 24. No exemplo, s_1 e s_2 são os estímulos, diferenciados apenas pela eventual presença de ruído. As saídas r_1 e r_2 são as taxas médias de disparo de cada unidade, e cada uma delas também serve como entrada para a mesma, comportamento chamado de ***feedback* recorrente**. Além disso, há uma interconexão entre as unidades, com a saída de uma

Figura 23 – Cubo de Necker



Fonte: Domínio público

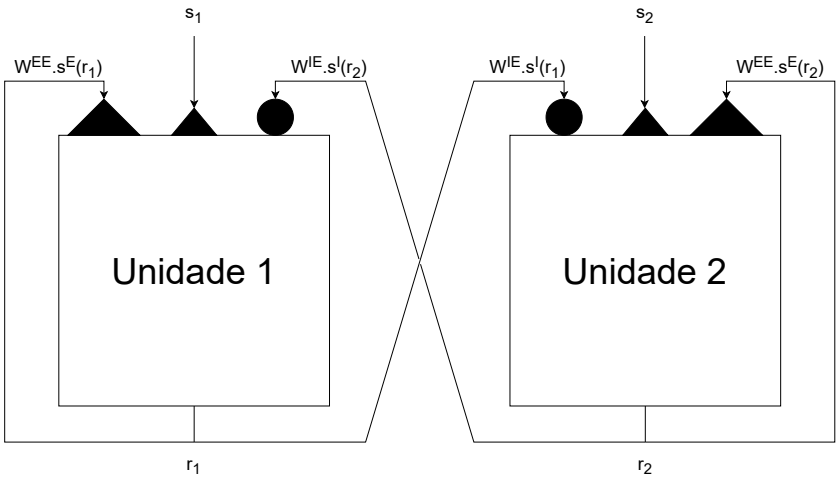
se tornando entrada de outra. Tanto as entradas quando as conexões recorrentes são excitatórias (representadas pelos triângulos no topo das unidades), reforçando a taxa de disparo, enquanto as interconexões são inibitórias (representadas pelos círculos pretos). W^{EE} e W^{IE} são as forças das conexões, com o EE indicando a conexão excitatória e IE a inibitória, e s^E e s^I são a fração de canais sinápticos excitatórios e inibitórios, respectivamente, que estão abertos, e são função da taxa de disparos. Além do ruído na entrada, pode ocorrer ruído dentro das unidades, e a presença de ambos pode induzir às transições entre estados bi-estáveis.

4.3.2 Circuitos de multi-estabilidade

Um circuito conhecido de multi-estabilidade é o **gerador de padrão central** (*central pattern generator*, CPG, em inglês), que são circuitos neurais que podem reproduzir padrões rítmicos de atividade neuronal sem receber entradas rítmicas (IJSPEERT, 2008). Eles são fundamentais, dentre outras coisas, para circuitos neurais associados à locomoção.

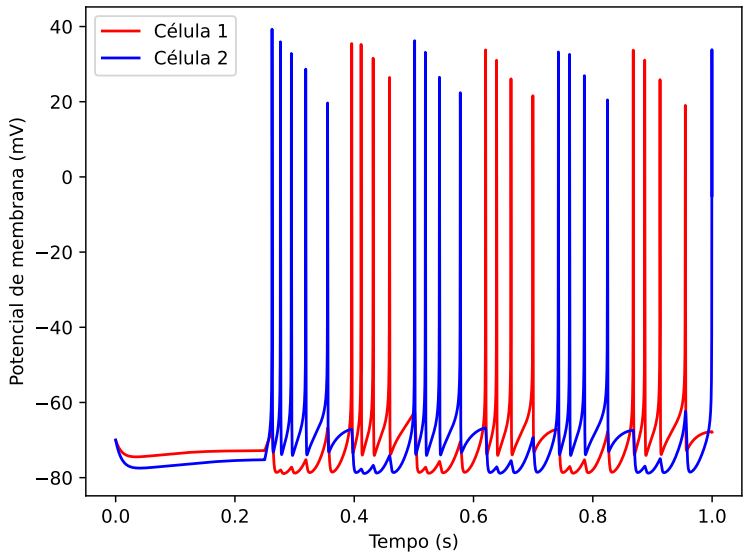
Um exemplo do seu comportamento é exibido na Figura 25. Nela, são exibidas as curvas do potencial de membrana de membrana de duas unidades, conectadas como mostrado anteriormente, e com um pulso de corrente aplicado por volta de 0,25 s. É possível observar que,

Figura 24 – Unidades de decisão com feedback recorrente e interconexão



Fonte: O autor (2023)

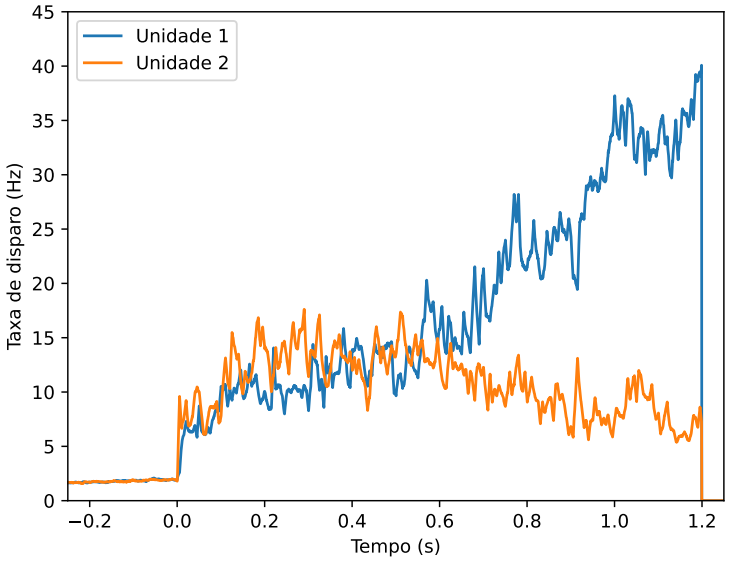
Figura 25 – Gerador de padrão central de duas unidades



Fonte: O autor (2023)

quando a célula 1 é ativada (representado pelo disparo dos potenciais de ação), a célula 2 é inibida. O contrário também ocorre, com a célula 1 sendo inibida com a ativação da célula 2, alternando entre qual das unidades dispara durante um intervalo de tempo. Não há o disparo simultâneo das duas células. Um correlato dessa atividade com a locomoção é o movimento das duas pernas humanas. Se associarmos cada perna com uma unidade, ora uma delas está em movimento (uma célula ativada), ora é a outra, alternando ao longo do tempo. Circuitos geradores de padrão central são bastante empregados no projeto de locomoção de robôs articulados (IJSPEERT, 2008).

Figura 26 – Circuito de tomada de decisão



Fonte: O autor (2023)

Outros circuitos conhecidos de multi-estabilidade são os de tomada de decisão. Esses circuitos acumulam informações sensoriais ao longo do tempo até que uma decisão possa ser tomada. A computação da informação pode ser feita simplesmente integrando a atividade neural ao longo do tempo (CAIN; SHEA-BROWN, 2012), como pode ser visto na Figura 26. Neste exemplo, um estímulo é aplicado simultaneamente (instante 0 s) em duas unidades, também conectadas como acima, que acumulam as evidências desse estímulo ao longo do tempo, alterando a taxa de disparo média das unidades. Neste caso, a decisão tomada, ou seja, a unidade escolhida, é a que atingir primeiro um determinado limiar, definido como a taxa de disparo de 40 Hz. A unidade 1 (em azul) foi a vencedora.

4.4 Modelos de taxa de disparo

Nos exemplos acima, os cálculos de taxa de disparo são feitos simulando vários neurônios individualmente. Mesmo usando modelos mais simplificados, como os de integra-e-dispara, esse estratégia pode não ser muito adequada, principalmente do ponto de vista da eficiência computacional, onde a simulação de um grande número de neurônios (milhares ou dezenas de milhares) pode se tornar impraticável mesmo utilizando computadores mais robustos. Nesses casos, convém utilizar um modelo que fornece a taxa de disparo. Sem dúvidas, o mais conhecido

deles é o de Wilson-Cowan (WILSON; COWAN, 1972). Esse modelo leva em consideração as subpopulações (unidades) de neurônios excitatórios e inibitórios separadamente, levando em conta o chamado **Princípio de Dale**, onde cada neurônio é classificado como puramente excitatório ou inibitório, não podendo ser os dois ao mesmo tempo (DALE, 1935).

As equações do modelo de Wilson-Cowan possuem o formato abaixo, com a Equação 4.10 referente à variação da taxa de disparos para a população de neurônios excitatórios, e a Equação 4.11 para a população de neurônios inibitórios:

$$\tau_e \frac{\mathrm{d}r_e}{\mathrm{d}t} = -r_e + F_e(w_{ee}r_e - w_{ie}r_i + T_e(t)) \tag{4.10}$$

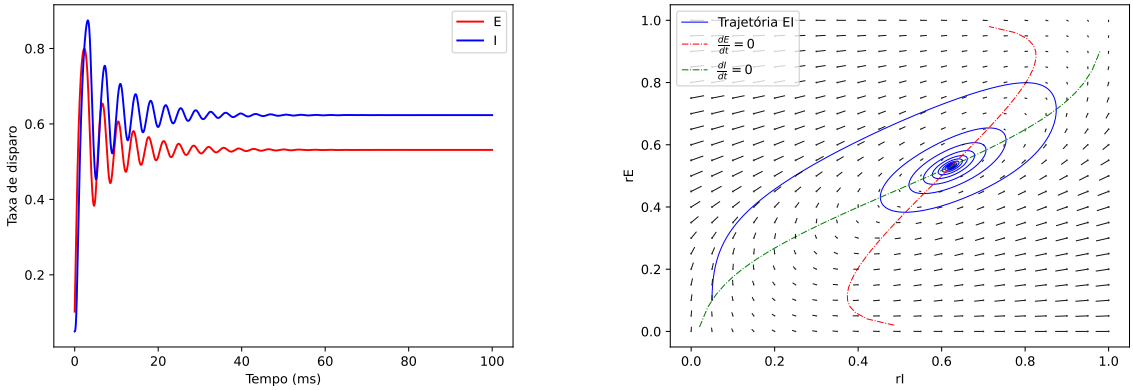
$$\tau_i \frac{\mathrm{d}r_i}{\mathrm{d}t} = -r_i + F_i(w_{ei}r_e - w_{ii}r_i + T_i(t)) \tag{4.11}$$

sendo $T_e(t)$ e $T_i(t)$ as entradas do modelo, r_e e r_i as taxas de disparo, τ_e e τ_i as constantes de tempo, w_{ee} e w_{ii} os pesos das conexões excitatórias (recorrentes) e w_{ie} e w_{ei} os pesos das conexões inibitórias. F é uma função de ativação que nesse caso, define a fração de neurônios que disparam para cada subpopulação.

O comportamento observado ao simular o modelo de Wilson-Cowan é exibido na Figura 27. A atividade, em termos de taxa de disparo de cada subpopulação, pode convergir para um valor estável em regime permanente, como mostrado na Figura 27a. Após aproximadamente 50 ms, as taxas de disparo não se alteram, para ambas as subpopulações. Esse modelo também possui uma significância importante em comparação outros trabalhos por ser uma introdução formal de ferramentas para análise de sistemas dinâmicos em neurociência (RAMEZANIAN-PANAHI et al., 2022). De maneira simplificada, sistemas dinâmicos são aqueles onde as variáveis mudam ao longo do tempo, como no caso das taxas de disparo se alterando. Quando as taxas de disparo de cada população não se alteram, a curva que representa a trajetória delas é chamada de ***nullcline*** (isóclina de inclinação nula, em tradução livre), ocorrendo quando as derivadas das equações 4.10 e 4.11 são iguais a zero, como mostradas na Figura 27b, sendo a curva em vermelho para a subpopulação excitatória, e em verde para a inibitória. A curva em azul representa a trajetória do sistema como um todo, que converge para o ponto de interseção entre as curvas das duas subpopulações, onde ambas as taxas não se alteram, sendo este o ponto de equilíbrio do sistema. As setas na cor cinza representam as trajetórias das alterações do sistema em direção ao ponto de equilíbrio.

Figura 27 – Comportamento das simulações do modelo de Wilson-Cowan

(a) Atividade **(b) Trajetória**



Fonte: o autor (2023)

4.5 Aprendizado e plasticidade de longa duração

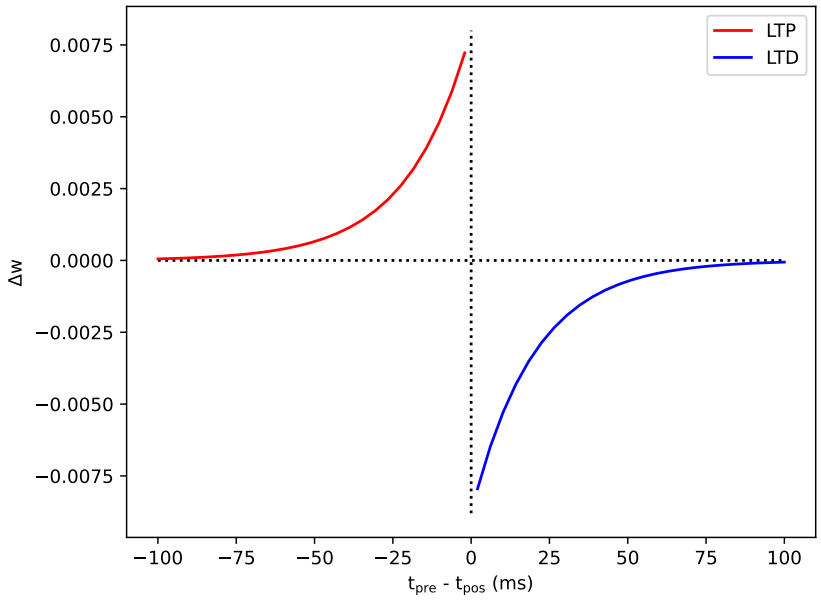
A plasticidade é a capacidade do cérebro de se adaptar, tanto estrutural quanto funcionalmente, em resposta a estímulos externos (MATEOS-APARICIO; RODRíGUEZ-MORENO, 2019). O termo se tornou bastante conhecido principalmente devido à chamada **Regra de Hebb**, que diz que se um neurônio A excita um neurônio B repetidas vezes, contribuindo para o disparo deste, então a sinapse entre os dois deve ser fortalecida (HEBB, 2005). Diferentemente das alterações de curta duração mostradas na Seção 4.2.1, as mudanças das forças de conexões neuronais da plasticidade sináptica persistem por longos períodos de tempo. Um exemplo é o conceito de **potencialização de longa duração** (*long-term potentiation, LTP, em inglês*), que se trata de um crescimento da força de conexão sináptica que duram por algumas dezenas de minutos (BLISS; LøMO, 1973). Do mesmo modo, há a **depressão de longa duração** (*long-term depression, LTD, em inglês*), que é a diminuição da força de conexão por um longo período de tempo (BEAR; ABRAHAM, 1996). A regra de plasticidade mais simples é dada pela equação abaixo:

$$\tau_w \frac{\mathrm{d}\mathbf{w}}{\mathrm{d}t} = v\mathbf{u} \tag{4.12}$$

onde τ_w é uma constante de tempo que controla a taxa em que os pesos se atualizam, $\mathbf{w}$ é o vetor de pesos sinápticos, v é a atividade do neurônio pós-sináptico e $\mathbf{u}$ o vetor de atividades dos neurônios pré-sinápticos.

Na versão simplificada de plasticidade exibida acima, tanto a atividade do neurônio pré-sináptico quanto a do pós aumentam a força da conexão. Na prática, porém, há uma espécie

Figura 28 – Plasticidade dependente do tempo de disparo (STDP)



Fonte: o autor (2023)

de competição entre diferentes sinapses, que pode ser observada na chamada **plasticidade dependente do tempo de disparo** (*spike-timing dependent plasticity*, STDP, em inglês), que depende do tempo relativo do disparo das células pré e pós-sinápticas, como exibido na Figura 28 (SONG; MILLER; ABBOTT, 2000). Na abscissa é apresentada a diferença de tempo entre os disparos dos neurônios pré e pós-sinápticos. 0 (zero) significa que ambos disparam ao mesmo tempo, valores negativos estão associados ao neurônio pré-sináptico disparando primeiro, e valores positivos ao disparo do neurônio pós-sináptico primeiro. Na primeira situação, com o pré seguido do pós (curva em vermelho), a atividade resulta em uma potencialização de longa duração, com a diferença de peso sendo maior proporcionalmente à diminuição do tempo entre os disparos, até próximo de 0 (zero), enquanto na situação onde o pós-sináptico dispara primeiro (curva em azul), a atividade resulta em uma depressão de longa duração, também com a diferença de peso aumentando proporcionalmente com a diminuição da diferença de tempo entre os disparos.

A modelagem da STDP é dada pela equação abaixo (SONG; MILLER; ABBOTT, 2000):

$$\Delta_w = \begin{cases} A_+ \exp(\Delta t/\tau_+) & \text{se } \Delta t < 0 \\ -A_- \exp(-\Delta t/\tau_-) & \text{se } \Delta t \geq 0 \end{cases} \tag{4.13}$$

sendo A_+ e A_- os valores máximos de modificação sináptica, ambos com valor positivo e o máximo próximo de $\Delta_t = 0$. τ_+ e τ_- são as faixas de intervalo entre disparo das células pré para

pós-sinápticas. A STDP é uma das bases de aprendizado de diversos algoritmos de aprendizado de máquina, que serão apresentados no Capítulo seguinte.

5 INTELIGÊNCIA ARTIFICIAL

5.1 Introdução

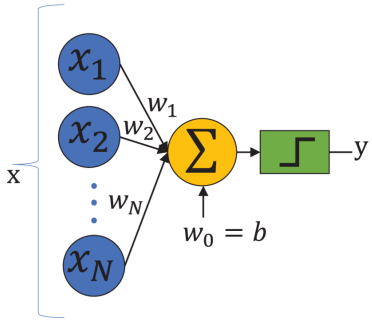
A inteligência artificial é um amplo ramo que trata de sistemas capazes de aprender e interpretar informações, e usar esse aprendizado para objetivos específicos (HAENLEIN; KAPLAN, 2019). Um dos campos da inteligência artificial é o aprendizado de máquina, onde a unidade de computação aprende a executar uma tarefa a partir de um conjuntos de exemplos de treinamento (LOURIDAS; EBERT, 2016), funcionando como um modelo computacional do cérebro. Esses modelos são construídos a partir da interconexão entre unidades de processamento, por vezes chamados de neurônios artificiais, e por isso são conhecidas como **redes neurais artificiais**.

O tipo mais comum de aprendizado de máquina é o chamado **aprendizado supervisionado**, onde o modelo é apresentado a um conjunto de dados de exemplo que são rotulados, ou seja, cada entrada tem a sua saída definida. Durante o aprendizado, o modelo fornece uma saída e, a partir da saída verdadeira, ele se ajusta internamente para se aproximar mais do real (LECUN; BENGIO; HINTON, 2015). Quando não há uma saída real rotulada disponível, pode-se utilizar o **aprendizado não-supervisionado**. Nesse caso, o modelo tenta aprender alguns padrões nos dados, como grupos de exemplos similares. O aprendizado não-supervisionado tem a capacidade de identificar novos tipos de informações, já que não estão restritos aos rótulos (ASNICAR et al., 2023).

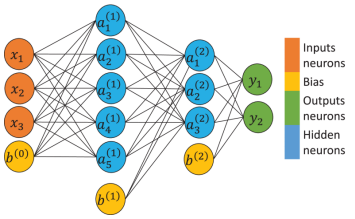
Os algoritmos de aprendizado de máquina também podem ser divididos quanto à atividade a ser realizada. Atividades de **classificação** identificam a classe da informação (por exemplo, se um animal é um cão ou gato) a partir de dados rotulados, o **agrupamento** (*clustering*, em inglês) determina classes agrupando informações similares em grupos (*clusters*) rotulados (como o agrupamento de filmes em gêneros), a **regressão** é usada para prever algum valor ou quantidade (como o preço de ações ao longo do tempo), e a **redução de dimensionalidade** representa dados de várias dimensões em outras menores (como as várias informações de um paciente sendo reduzidas às mais importantes para o diagnóstico de uma doença).

Figura 29 – Arquiteturas do Perceptron

(a) Perceptron



(b) Perceptron de multi-camadas



Fonte: (PARHI; UNNIKRISHNAN, 2020)

5.2 Redes neurais

O primeiro modelo de neurônio artificial foi o de McCulloch-Pitts, baseado em lógica binária e que possuía duas fase, a primeira onde o neurônio soma as contribuições de todos os neurônios anteriores, e a segunda onde o neurônio dispara apenas se um limiar for ultrapassado e não houver disparo de neurônio inibitório (MCCULLOCH; PITTS, 1943). Apesar de revolucionário para a época, um dos principais problemas do modelo de McCulloch-Pitts era a ausência do ajuste de pesos, ou seja, as conexões entre os neurônios sempre tinham a mesma força. Muitas pesquisas se desenvolveram na área para suprir essa necessidade, até que Frank Rosenblatt, inspirado pela teoria de Hebb vista anteriormente, criou o *Perceptron* (ROSENBLATT, 1958), desenvolvido para ser um neurônio mais generalizado, sendo a base do aprendizado de máquina moderno. Matematicamente, o *Perceptron* pode ser modelado pela seguinte equação:

$$y = \sigma\left(\sum_{k=1}^{N} w_k x_k + b\right) \tag{5.1}$$

onde x_k é a resposta da sinapse k, w_k é o peso da sinapse, b é o *bias* (viés, em português), um peso de referência que representa um conhecimento a priori, σ é uma função de ativação não-linear, e y é a saída do neurônio, como representado na Figura 29a. Resolvido o problema da atualização dos pesos, o *Perceptron* sozinho é incapaz de realizar tarefas de separação não-linear (MINSKY; PAPERT, 2017), levando à criação, muitos anos depois, do *Perceptron* de multi-camadas (MLP, *Multilayer perceptron*, em inglês), que supre as limitações do *Perceptron* simples ao agrupar camadas de neurônios, como na Figura 29b. As camadas presentes entre a entrada e a saída da rede são chamadas de camadas ocultas (ou intermediárias).

Os vários neurônios presentes nessa arquitetura frequentemente têm seus pesos atualizados através do algoritmo de gradiente descendente, que busca minimizar o erro entre a saída

obtida pela rede e a real (aprendizado supervisionado), e essa atualização é feita através da metodologia da retro-propagação (*backpropagation*, em inglês), ou seja, as atualizações das camadas finais da rede são propagadas em direção às iniciais (WERBOS, 1974). Quando há uma grande quantidade de camadas na rede ela é chamada de rede neural profunda (*deep neural network*, DNN, em inglês), permitindo um mapeamento de dados mais complexos, o que não seria muito eficiente em redes não profundas. Uma lista não exaustiva de arquiteturas, profundas ou não, de redes neurais artificiais (*artificial neural networks* (ANN), em inglês) é apresentada abaixo:

a) ***Convolutional Neural Networks* (CNN, Redes neurais convolucionais)**: possui camadas que implementam a operação matemática de convolução, aplicando um filtro no sinal, e camadas de sub-amostragem (*downsampling*), chamadas de *pooling*. Essas redes lidam bem com sinais em duas dimensões, e por isso são frequentemente aplicadas em tarefas de visão computacional (relacionadas a imagens e vídeos) (LECUN et al., 1998);

b) ***Recurrent Neural Networks* (RNN, Redes neurais recorrentes)**: muito usadas em tarefas de entradas sequenciais, como processamento de fala e linguagem, possuem a característica de processar os dados elemento a elemento, considerando também informações dos elementos passados, armazenados em pesos de conexões recorrentes, que funcionam como uma memória dinâmica para a rede (ELMAN, 1990);

c) ***Hopfield Networks* (Redes Hopfield)**: proposta por J. J. Hopfield, são redes que são treinadas para aprender padrões (memórias) dos dados de maneira associativa, baseado no princípio de Hebb de que neurônios que disparam juntos se conectam;

d) ***Autoencoder* (auto codificador)**: redes que aprendem a representar os dados reduzindo sua dimensionalidade, diminuindo as camadas ocultas, e, posteriormente, recriando os dados originais (HINTON; SALAKHUTDINOV, 2006);

e) ***Long Short-Term Memory* (LSTM, memória longa de curto prazo)**: uma variação da RNN composta de 4 (quatro) blocos de memória, sendo um principal e os outros três, chamados de portões de entrada, esquecimento e saída, responsáveis por alterar o estado da célula como um todo (HOCHREITER; SCHMIDHUBER, 1997). Devido à sua capacidade de reter informações de longo tempo em séries temporais, são muito usadas em sistemas de reconhecimento de voz.

Além das arquiteturas citadas acima, existem outras que são baseadas no princípio de funcionamento do neurônio, envolvendo, inclusive, alterações físicas dos equipamentos onde são implementadas. Essas redes são chamadas de **neuromórficas**, detalhadas na seção seguinte.

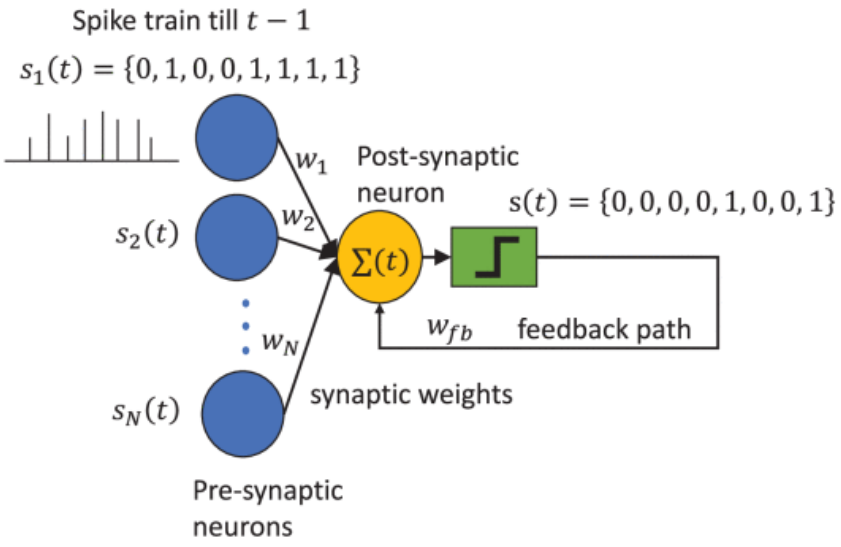
5.3 Redes neuromórficas

A maioria das redes neurais convencionais, derivadas da MLP, são implementadas com base na arquitetura de computadores de Von Neumann, onde as unidades de memória e processamento ficam separadas (NEUMANN, 1993). Algumas características dessa arquitetura incluem a codificação da informação para sinais binários (0 e 1) e os programas escritos em instruções explícitas (algoritmos), que são computadas pelo processador e os dados são armazenados na memória, trafegando por um barramento que conecta os dois elementos.

As arquiteturas neuromórficas surgem como uma opção onde estrutura e funcionamento são mais semelhantes ao do cérebro, em especial os neurônios e sinapses (IVANOV et al., 2022). As redes neurais desenvolvidas com base nos princípios dessa arquitetura são chamadas de **redes neurais de disparo** (*spiking neural networks*, SNN, em inglês), cuja arquitetura é mostrada na Figura 30. Diferentemente da arquitetura de Von Neumann, a neuromórfica distribui o processamento e memória em conjunto nos elementos de neurônio e sinapses, respectivamente, evitando o tempo da transmissão de informação através do barramento entre memória e processador (INDIVERI; LIU, 2015). Outra diferença da arquitetura neuromórfica é que a informação é codificada, tanto espacial quanto temporalmente, em sequências de potenciais de ação (FRENKEL; BOL; INDIVERI, 2023). Além disso, os programas de computadores neuromórficos são definidos pela estrutura da rede neural e seus parâmetros (SCHUMAN et al., 2022).

Alguns projetos de dispositivos de *hardware* foram desenvolvidos com base na arquitetura neuromórfica. O projeto ***SpiNNaker*** propôs um computador paralelo massivo de milhões de núcleos, desenvolvido para a modelagem de redes neurais de disparo de grande escala (FURBER et al., 2014). O ***BrainScaleS*** é um robusto sistema integrado de *hardware* neuromórfico capaz de ser usado para modelar a complexidade neural, bem como topologias de rede realistas, permitindo a execução direta de modelos que anteriormente só eram possíveis de serem simulados numericamente (SCHEMMEL et al., 2010). O projeto responsável pela sua criação, chamado de FACETS, participou do desenvolvimento do modelo AELIF, visto anteriormente, e esse modelo foi implementado fisicamente como um *chip* presente no *BrainScaleS* (AAMIR et al., 2017). O

Figura 30 – Arquitetura de redes neurais de disparo (SNN)



Fonte: (PARHI; UNNIKRISHNAN, 2020)

projeto ***TrueNorth*** criou um chip com mais de 5 (cinco) bilhões de transistores e 4096 (quatro mil e noventa e seis) núcleos sinápticos, interconectados por uma rede de chips internos que integra 1 (um) milhão de neurônios de disparo programáveis e 256 (duzentas e cinquenta e seis) milhões de sinapses configuráveis (MEROLLA et al., 2014). O ***Loihi*** é um chip de 60 mm^2 fabricado com processadores da Intel de 14 nm que integra uma variedade de recursos para o campo de redes neurais de disparo, como conectividade, compartimentos dendríticos, atrasos sinápticos e, mais importante, regras de aprendizado sinápticos programáveis (DAVIES et al., 2018).

Diversos outros chips neuromórficos existem ou estão sendo pesquisados, com diferentes tecnologias de desenvolvimento e modelagem aplicadas (MEHONIC; KENYON, 2022), alguns deles utilizando uma tecnologia chamada de **memristor**, uma espécie de transistor capaz de alterar a sua condutividade dependendo da corrente ou tensão aplicada aos seus terminais, sendo úteis para a implementação eficiente de computação em memória para as redes neurais de disparo (MEHONIC et al., 2020). Vale destacar, também, que muitos desses dispositivos podem ser usados com redes neurais convencionais, como RNNs e CNNs, fazendo com que os estudos desse tipo de arquitetura estejam em alta.

As redes neurais de disparo usam, principalmente, neurônios do tipo integra-e-dispara para trocar informações via disparos de potencial de ação, e as unidades de computação são conectadas entre si e interagem através das sinapses (ROY; JAISWAL; PANDA, 2019). É possível

converter as redes neurais convencionais em redes de disparo fazendo alguns ajustes de pesos e normalização, a fim de tirar proveito das vantagens proporcionados pelo uso do *hardware* neuromórfico (DIEHL et al., 2016). Para fazer a conversão entre as redes neurais artificiais e as redes neurais de disparo, os dados de entrada precisam ser codificados em disparos de potencial de ação, e essa codificação pode ser baseada na taxa de disparo, na ordem de disparos de uma população, ou temporal.

Na codificação de taxa, a média temporal (quantidade de disparos por intervalo de tempo) ou a média de disparos de diferentes *trials* é usada, tendo um correlato com o que ocorre em populações de neurônios do córtex auditivo primário (DECHARMS; MERZENICH, 1996). Na codificação de ordem de disparos de uma população vários neurônios diferentes disparam um potencial de ação, porém o importante não é o instante exato em que eles disparam, e sim a ordem deles, ou seja, qual neurônio disparou primeiro, qual foi o segundo, e assim por diante (MALLOT, 2013). A codificação temporal considera o exato momento em que ocorre o disparo do potencial de ação de cada neurônio, um comportamento neuronal de relevância (BOHTE, 2004). Subclassificações dessa codificação podem considerar apenas o instante do primeiro disparo ou a latência (a diferença de tempo entre os disparos). Os critérios para seleção do método de codificação variam por diferentes aspectos, como a minimização da perda de informação após a decodificação ou o aumento da acurácia de previsão/classificação.

Do ponto de vista do aprendizado das redes neurais de disparo, o não-supervisionado frequentemente faz uso da STDP como parte do seu mecanismo, com os pesos de potencialização e depressão de longa duração sendo atualizados durante o aprendizado (KHERADPISHEH et al., 2018). No caso do aprendizado supervisionado, existe uma variação do método *backpropagation*, relacionando as ativações das unidades de redes neurais com as taxas de disparo (DIEHL et al., 2015). No entanto, há discussões acerca do treinamento direto com *backpropagation* na simulação do cérebro, principalmente na conversão de ANNs para SNNs. Algumas alternativas são encontradas na literatura, como o *SpikeProp*, semelhante ao *backpropagation* (BOHTE; KOK; POUTRé, 2002), o *ReSuMe*, que pode ser usado em tarefas de tomada de decisão (PONULAK; KASIńSKI, 2010), e o *SPAN*, que transforma as sequências de potenciais de ação na fase de treinamento em sinais analógicos, para que operações matemáticas comuns possam ser aplicadas (MOHEMMED et al., 2012).

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou uma série de conteúdos para a execução de um curso introdutório de neurociência computacional, voltado para diversos públicos, servindo como um guia de execução. Baseado em algumas execuções do curso, uma possível sequência didática para o mesmo é mostrada na Tabela 4. Também baseado em execuções do curso, foi elaborada a proposta de atividades avaliativas, referentes a cada capítulo apresentado, como segue:

a) Introdução: questões simples de neurobiologia, equações diferenciais e implementações de códigos em Python;

b) Modelos: alterações de parâmetros nos modelos LIF, ELIF e/ou AELIF;

c) Conexões: implementação de circuitos de multi-estabilidade com o modelo de Wilson-Cowan;

d) Tarefa final: implementação de codificação para a rede de disparo.

A proposta de execução do curso é utilizando o Google Colaboratory (Colab), que fornece um ambiente de programação em Python gratuito, incluindo recursos de *hardware* robustos para aplicações de aprendizado de máquina (BISONG, 2019). Como exibido na Figura 31, o Colab é executado a partir de um *Notebook*, que facilita a execução e reprodução de códigos (bloco mais escuro) e conteúdos em conjunto (SHEN, 2014). Todos os códigos-fonte utilizados durante o curso, incluindo os que geram algumas das imagens deste texto, estão disponíveis em repositório online[1], de maneira gratuita e com uma licença permissiva livre para uso. Como os códigos estão no formato dos *notbooks*, eles estão prontos para o uso diretamente no Colab.

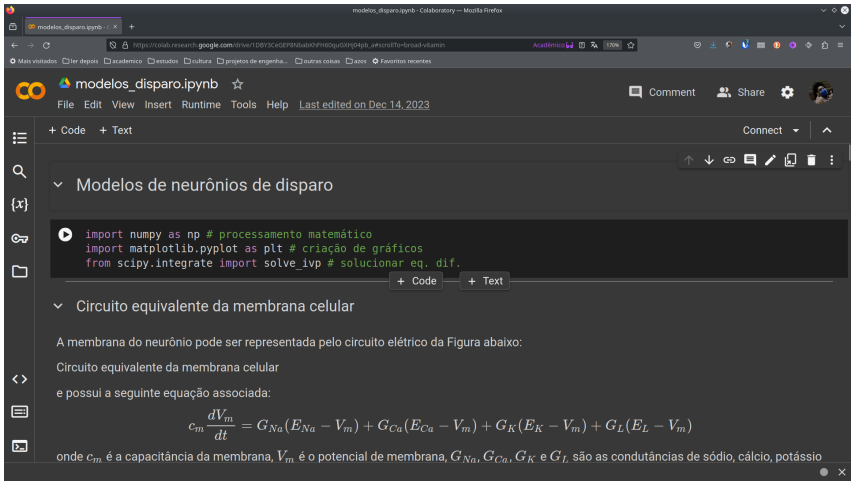
[1] <https://gitlab.com/weversonvn/intro_neurocomp>

Tabela 4 – Proposta de sequência didática

Número da aula	Tema	Conteúdo
1	Apresentação da disciplina	Apresentação do estado da arte
2-4	Introdução	Neurobiologia; equações diferenciais; introdução ao Python
5-9	Modelos de neurônio de disparo	Modelos LIF, ELIF, AELIF, Izhikevich, Hodgkin-Huxley
10-17	Conexões entre neurônios	Sinapses; Sinapses dinâmicas; Multi-estabilidade; Modelos de Wilson-Cowan; Aprendizado
18-21	Inteligência artificial	Redes neurais; Redes neuromórficas
22-24	Conclusão	Apresentações/entregas de trabalhos; Avaliação da disciplina

Fonte: o autor (2023)

Figura 31 – Ambiente do Google Colaboratory com o conteúdo de neurônios de disparo



Fonte: o autor (2023)

Possíveis melhorias para o curso começam pelo refinamento dos conteúdos já existentes, baseado nas considerações que venham a ser obtidas por eventuais alunos que o façam, como a alteração da ordem dos conteúdos, maior detalhamento de algum tópico específico, diminuição de outros. Além disso, um material voltado para exercícios pode ser bastante útil, tanto dos conteúdos teóricos quanto dos práticos, com destaque para tarefas em Python voltadas para cursos onde o público alvo não tem grande familiaridade com a linguagem ou com programação.

Como conteúdos novos pode-se incluir a simulação de modelos multi-compartimento, com destaque para a estimulação elétrica axonal, para estudar a propagação do potencial de ação ao longo do axônio (RATTAY; LUTTER; FELIX, 2001). Outra possibilidade é a análise de sinais de eletroencefalograma (EEG), que são respostas elétricas obtidas das atividades cerebrais, podendo ser utilizadas para associações com comorbidades, como depressão e ansiedade (CAVANAGH et al., 2019). Por fim, a apresentação de pacotes do Python voltados para análises semelhantes às apresentadas no curso pode ser incluído, uma vez que o uso destes potencialmente se torna mais simplificado com o entendimento obtido a partir do conteúdo deste trabalho.

REFERÊNCIAS

AAMIR, S. A. et al. From LIF to AdEx neuron models: Accelerated analog 65 nm CMOS implementation. In: **2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)**. IEEE, 2017. p. 1–4. ISBN 978-1-5090-5803-7. Disponível em: <http://ieeexplore.ieee.org/document/8325167/>.

ASNICAR, F. et al. Machine learning for microbiologists. p. 1–15, 2023. ISSN 1740-1534. Publisher: Nature Publishing Group. Disponível em: <https://www.nature.com/articles/s41579-023-00984-1>.

BEAR, M. F.; ABRAHAM, W. C. Long-term depression in hippocampus. v. 19, n. 1, p. 437–462, 1996. ISSN 0147-006X, 1545-4126. Disponível em: <https://www.annualreviews.org/doi/10.1146/annurev.ne.19.030196.002253>.

BEN-ARI, Y. Excitatory actions of gaba during development: the nature of the nurture. v. 3, n. 9, p. 728–739, 2002. ISSN 1471-0048. Number: 9 Publisher: Nature Publishing Group. Disponível em: <https://www.nature.com/articles/nrn920>.

BISONG, E. Google colaboratory. In: ____. **Building Machine Learning and Deep Learning Models on Google Cloud Platform**. Apress, 2019. p. 59–64. ISBN 978-1-4842-4469-2 978-1-4842-4470-8. Disponível em: <http://link.springer.com/10.1007/978-1-4842-4470-8_7>.

BLISS, T. V. P.; LøMO, T. Long-lasting potentiation of synaptic transmission in the dentate area of the anaesthetized rabbit following stimulation of the perforant path. v. 232, n. 2, p. 331–356, 1973. ISSN 0022-3751, 1469-7793. Disponível em: <https://physoc.onlinelibrary.wiley.com/doi/10.1113/jphysiol.1973.sp010273>.

BLITZ, D. M.; FOSTER, K. A.; REGEHR, W. G. Short-term synaptic plasticity: a comparison of two synapses. **Nature Reviews Neuroscience**, v. 5, n. 8, p. 630–640, 2004. ISSN 1471-003X, 1471-0048. Disponível em: <https://www.nature.com/articles/nrn1475>.

BOHTE, S. M. The evidence for neural information processing with precise spike-times: A survey. v. 3, n. 2, p. 195–206, 2004. ISSN 1567-7818. Disponível em: <http://link.springer.com/10.1023/B:NACO.0000027755.02868.60>.

BOHTE, S. M.; KOK, J. N.; POUTRé, H. L. Error-backpropagation in temporally encoded networks of spiking neurons. v. 48, n. 1, p. 17–37, 2002. ISSN 09252312. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0925231201006580>.

CAIN, N.; SHEA-BROWN, E. Computational models of decision making: integration, stability, and noise. v. 22, n. 6, p. 1047–1053, 2012. ISSN 0959-4388. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0959438812000785>.

CAVANAGH, J. F. et al. Multiple dissociations between comorbid depression and anxiety on reward and punishment processing: Evidence from computationally informed EEG. v. 3, n. 0, p. 1, 2019. ISSN 2379-6227. Disponível em: <https://cpsyjournal.org/article/10.1162/CPSY_a_00024/>.

DALE, H. Pharmacology and nerve-endings. v. 28, n. 3, p. 319–332, 1935. ISSN 0035-9157. Disponível em: <http://journals.sagepub.com/doi/10.1177/003591573502800330>.

DAVIES, M. et al. Loihi: A neuromorphic manycore processor with on-chip learning. v. 38, n. 1, p. 82–99, 2018. ISSN 0272-1732, 1937-4143. Disponível em: <https://ieeexplore.ieee.org/document/8259423/>.

DAYAN, P.; ABBOTT, L. F. **Theoretical neuroscience: computational and mathematical modeling of neural systems**. [S.l.]: Massachusetts Institute of Technology Press, 2001. (Computational neuroscience). ISBN 978-0-262-04199-7.

DECHARMS, R. C.; MERZENICH, M. M. Primary cortical representation of sounds by the coordination of action-potential timing. v. 381, n. 6583, p. 610–613, 1996. ISSN 0028-0836, 1476-4687. Disponível em: <https://www.nature.com/articles/381610a0>.

DIEHL, P. U. et al. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In: **2015 International Joint Conference on Neural Networks (IJCNN)**. IEEE, 2015. p. 1–8. ISBN 978-1-4799-1960-4. Disponível em: <http://ieeexplore.ieee.org/document/7280696/>.

DIEHL, P. U. et al. Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware. In: **2016 IEEE International Conference on Rebooting Computing (ICRC)**. IEEE, 2016. p. 1–8. ISBN 978-1-5090-1370-8. Disponível em: <http://ieeexplore.ieee.org/document/7738691/>.

ELMAN, J. L. Finding structure in time. v. 14, n. 2, p. 179–211, 1990. ISSN 0364-0213, 1551-6709. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1207/s15516709cog1402_1>.

ERMENTROUT, B.; TERMAN, D. H. **Mathematical foundations of neuroscience**. [S.l.]: Springer, 2010. v. 35. (Interdisciplinary applied mathematics, v. 35). ISBN 978-0-387-87708-2.

FRENKEL, C.; BOL, D.; INDIVERI, G. Bottom-up and top-down approaches for the design of neuromorphic processing systems: Tradeoffs and synergies between natural and artificial intelligence. v. 111, n. 6, p. 623–652, 2023. ISSN 0018-9219, 1558-2256. Disponível em: <https://ieeexplore.ieee.org/document/10144567/>.

FURBER, S. B. et al. The SpiNNaker project. v. 102, n. 5, p. 652–665, 2014. ISSN 0018-9219, 1558-2256. Disponível em: <https://ieeexplore.ieee.org/document/6750072/>.

HAENLEIN, M.; KAPLAN, A. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. v. 61, n. 4, p. 5–14, 2019. ISSN 0008-1256. Publisher: SAGE Publications Inc. Disponível em: <https://doi.org/10.1177/0008125619864925>.

HARRIS, C. R. et al. Array programming with NumPy. v. 585, n. 7825, p. 357–362, 2020. ISSN 0028-0836, 1476-4687. Disponível em: <https://www.nature.com/articles/s41586-020-2649-2>.

HEBB, D. **The Organization of Behavior**. 0. ed. [S.l.]: Psychology Press, 2005. ISBN 978-1-4106-1240-3.

HILLE, B. Ionic channels of excitable membranes. v. 306, n. 2, p. 277–278, 1992. ISSN 1873-3468. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1016/0014-5793%2892%2981020-M. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1016/0014-5793%2892%2981020-M>.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. v. 313, n. 5786, p. 504–507, 2006. ISSN 0036-8075, 1095-9203. Disponível em: <https://www.science.org/doi/10.1126/science.1127647>.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. v. 9, n. 8, p. 1735–1780, 1997. ISSN 0899-7667, 1530-888X. Disponível em: <https://direct.mit.edu/neco/article/9/8/1735-1780/6109>.

HODGKIN, A. L.; HUXLEY, A. F. A quantitative description of membrane current and its application to conduction and excitation in nerve. v. 117, n. 4, p. 500–544, 1952. ISSN 0022-3751, 1469-7793. Disponível em: <https://physoc.onlinelibrary.wiley.com/doi/10.1113/jphysiol.1952.sp004764>.

HUNTER, J. D. Matplotlib: A 2d graphics environment. v. 9, n. 3, p. 90–95, 2007. ISSN 1521-9615. Disponível em: <http://ieeexplore.ieee.org/document/4160265/>.

IJSPEERT, A. J. Central pattern generators for locomotion control in animals and robots: A review. v. 21, n. 4, p. 642–653, 2008. ISSN 0893-6080. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0893608008000804>.

INDIVERI, G.; LIU, S.-C. Memory and information processing in neuromorphic systems. v. 103, n. 8, p. 1379–1397, 2015. ISSN 0018-9219, 1558-2256. Disponível em: <http://ieeexplore.ieee.org/document/7159144/>.

IVANOV, D. et al. Neuromorphic artificial intelligence systems. v. 16, 2022. ISSN 1662-453X. Disponível em: <https://www.frontiersin.org/articles/10.3389/fnins.2022.959626>.

IZHIKEVICH, E. Simple model of spiking neurons. v. 14, n. 6, p. 1569–1572, 2003. ISSN 1941-0093. Conference Name: IEEE Transactions on Neural Networks.

KHERADPISHEH, S. R. et al. STDP-based spiking deep convolutional neural networks for object recognition. v. 99, p. 56–67, 2018. ISSN 08936080. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0893608017302903>.

KITAJIMA, T.; FENG, Z.; AZHIM, A. Simulation of neural behavior. In: LOPEZ-RUIZ, R. (Ed.). **Numerical Simulation - From Brain Imaging to Turbulent Flows**. InTech, 2016. ISBN 978-953-51-2564-8 978-953-51-2565-5. Disponível em: <http://www.intechopen.com/books/numerical-simulation-from-brain-imaging-to-turbulent-flows/simulation-of-neural-behavior>.

LAPICQUE, L. Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarization. v. 9, p. 620–635, 1907.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. v. 521, n. 7553, p. 436–444, 2015. ISSN 1476-4687. Number: 7553 Publisher: Nature Publishing Group. Disponível em: <https://www.nature.com/articles/nature14539>.

LECUN, Y. et al. Gradient-based learning applied to document recognition. v. 86, n. 11, p. 2278–2324, 1998. ISSN 00189219. Disponível em: <http://ieeexplore.ieee.org/document/726791/>.

LOURIDAS, P.; EBERT, C. Machine learning. v. 33, n. 5, p. 110–115, 2016. ISSN 1937-4194. Conference Name: IEEE Software. Disponível em: <https://ieeexplore.ieee.org/document/7548905>.

MALLOT, H. A. Coding and representation. In: ____. **Computational Neuroscience**. Springer International Publishing, 2013. v. 2, p. 113–129. ISBN 978-3-319-00860-8 978-3-319-00861-5. Series Title: Springer Series in Bio-/Neuroinformatics. Disponível em: <http://link.springer.com/10.1007/978-3-319-00861-5_5>.

MATEOS-APARICIO, P.; RODRíGUEZ-MORENO, A. The impact of studying brain plasticity. v. 13, p. 66, 2019. ISSN 1662-5102. Disponível em: <https://www.frontiersin.org/article/10.3389/fncel.2019.00066/full>.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. v. 5, n. 4, p. 115–133, 1943. ISSN 1522-9602. Disponível em: <https://doi.org/10.1007/BF02478259>.

MEHONIC, A.; KENYON, A. J. Brain-inspired computing needs a master plan. v. 604, n. 7905, p. 255–260, 2022. ISSN 0028-0836, 1476-4687. Disponível em: <https://www.nature.com/articles/s41586-021-04362-w>.

MEHONIC, A. et al. Memristors—from in-memory computing, deep learning acceleration, and spiking neural networks to the future of neuromorphic and bio-inspired computing. v. 2, n. 11, p. 2000085, 2020. ISSN 2640-4567, 2640-4567. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1002/aisy.202000085>.

MEROLLA, P. A. et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. v. 345, n. 6197, p. 668–673, 2014. ISSN 0036-8075, 1095-9203. Disponível em: <https://www.science.org/doi/10.1126/science.1254642>.

MILLER, P. **An Introductory Course in Computational Neuroscience**. The MIT Press, 2018. (Computational neuroscience series). ISBN 978-0-262-03825-6. Disponível em: <https://mitpress.ublish.com/book/intro-computational-neuroscience>.

MINSKY, M.; PAPERT, S. A. **Perceptrons: An Introduction to Computational Geometry**. The MIT Press, 2017. ISBN 978-0-262-34393-0. Disponível em: <https://direct.mit.edu/books/book/3132/perceptronsan-introduction-to-computational>.

MOHEMMED, A. et al. Span: spike pattern association neuron for learning spatio-temporal spike patterns. v. 22, n. 4, p. 1250012, 2012. ISSN 0129-0657, 1793-6462. Disponível em: <https://www.worldscientific.com/doi/abs/10.1142/S0129065712500128>.

NEUMANN, J. V. First draft of a report on the EDVAC. v. 15, n. 4, p. 27–75, 1993. ISSN 1058-6180. Disponível em: <http://ieeexplore.ieee.org/document/238389/>.

PARHI, K. K.; UNNIKRISHNAN, N. K. Brain-inspired computing: Models and architectures. v. 1, p. 185–204, 2020. ISSN 2644-1225. Conference Name: IEEE Open Journal of Circuits and Systems. Disponível em: <https://ieeexplore.ieee.org/document/9238476>.

PONULAK, F.; KASIńSKI, A. Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting. v. 22, n. 2, p. 467–510, 2010. ISSN 0899-7667, 1530-888X. Disponível em: <https://direct.mit.edu/neco/article/22/2/467-510/7529>.

RAMEZANIAN-PANAHI, M. et al. Generative models of brain dynamics. v. 5, p. 807406, 2022. ISSN 2624-8212. Disponível em: <https://www.frontiersin.org/articles/10.3389/frai.2022.807406/full>.

RATTAY, F.; LUTTER, P.; FELIX, H. A model of the electrically excited human cochlear neuron. v. 153, n. 1, p. 43–63, 2001. ISSN 03785955. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0378595500002562>.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. v. 65, n. 6, p. 386–408, 1958. ISSN 1939-1471. Place: US Publisher: American Psychological Association.

ROY, K.; JAISWAL, A.; PANDA, P. Towards spike-based machine intelligence with neuromorphic computing. v. 575, n. 7784, p. 607–617, 2019. ISSN 0028-0836, 1476-4687. Disponível em: <https://www.nature.com/articles/s41586-019-1677-2>.

SCHEMMEL, J. et al. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In: **Proceedings of 2010 IEEE International Symposium on Circuits and Systems**. IEEE, 2010. p. 1947–1950. ISBN 978-1-4244-5308-5. Disponível em: <http://ieeexplore.ieee.org/document/5536970/>.

SCHUMAN, C. D. et al. Opportunities for neuromorphic computing algorithms and applications. v. 2, n. 1, p. 10–19, 2022. ISSN 2662-8457. Disponível em: <https://www.nature.com/articles/s43588-021-00184-y>.

SHEN, H. Interactive notebooks: Sharing the code. v. 515, n. 7525, p. 151–152, 2014. ISSN 1476-4687. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 7525 Primary_atype: Special Features Publisher: Nature Publishing Group Subject_term: Communication;Computational biology and bioinformatics;Information technology;Publishing Subject_term_id: communication;computational-biology-and-bioinformatics;information-technology;publishing. Disponível em: <https://www.nature.com/articles/515151a>.

SONG, S.; MILLER, K. D.; ABBOTT, L. F. Competitive hebbian learning through spike-timing-dependent synaptic plasticity. v. 3, n. 9, p. 919–926, 2000. ISSN 1097-6256, 1546-1726. Disponível em: <https://www.nature.com/articles/nn0900_919>.

WERBOS, P. J. **Beyond regression: new tools for prediction and analysis in the behavioral sciences**. phdthesis — Harvard University, 1974.

WILSON, H. R.; COWAN, J. D. Excitatory and inhibitory interactions in localized populations of model neurons. v. 12, n. 1, p. 1–24, 1972. ISSN 00063495. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0006349572860685>.

APÊNDICE A – TUTORIAL PYTHON

A.1 Introdução

- Linguagem aberta, gratuita e de fácil aprendizado
- Linguagem de programação de alto nível (a programação é com palavras comuns da língua inglês, como *print* para *exibir* na tela)
- O uso no curso pode ser instalando o próprio pacote oficial do Python, instalando o Anaconda (um conjunto de ferramentas para processamento científico) ou usando o Google Colaboratory
- O *Python* é uma linguagem interpretativa, significando que não é necessária a compilação para seu uso (não é necessário converter previamente o código para linguagem de máquina, como em outras linguagens). O interpretador do *Python* pode ser executado interativamente, o que é útil para testar em tempo real a linguagem. Neste texto, códigos começados por >>> são escritas diretamente no interpretador, e não em um arquivo de texto, e as linhas seguintes sem o >>> são a resposta do comando executado. Linhas de continuação são representadas por ... e aparecem em construções de várias linhas, sempre endentadas (com um recuo, geralmente de 4 espaços em branco):

```
>>> print("Olá, mundo.")
Olá, mundo.
>>> if True:
...     print("Multi-linhas sempre precisam de endentação.")
...
Multi-linhas sempre precisam de endentação.
```

A.2 Comandos básicos

- Atribuições são feitas utilizando = e comentários usando #

```
# primeiro comentário
spam = 1  # segundo comentário
```

```
# um terceiro comentário
texto = "# este não é um comentário, pois está entre aspas."
```

- O interpretador funciona como uma simples calculadora, com os símbolos padrão funcionando para a maioria dos operadores:

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5  # divisão sempre retorna um número em ponto flutuante (não inteiro
1.6
```

- Para descartar a parte fracionária da divisão usa-se `//` e `%` para calcular o resto da divisão

```
>>> 17 / 3  # divisão clássica retorna ponto flutuante
5.666666666666667
>>>
>>> 17 // 3  # descarta a parte fracionária
5
>>> 17 % 3  # retorna o resto da divisão
2
>>> 5 * 3 + 2  # quociente * divisor + resto
17
```

- Usa-se `**` para calcular potências

```
>>> 5 ** 2  # 5 ao quadrado
25
>>> 2 ** 7  # 2 elevado a 7
128
```

- Nenhum resultado é exibido ao fazer atribuições a uma variável

```
>>> largura = 20
>>> altura = 5 * 9
```

```
>>> largura * altura
900
```

A.3 Sequências de texto (*strings*)

- Sequências de texto podem ser definidas entre aspas simples ('...') ou duplas ("...") com o mesmo resultado, e o caractere \ é usado para escapar as aspas (considerá-las como parte do texto, e não o marcador de fim da *string*):

```
>>> 'ovo frito'  # aspas simples
'ovo frito'
>>> 'McDonald\'s' # usando \' para escapar a aspa simples
"McDonald's"
>>> "McDonald's"  # ... ou use aspas duplas
"McDonald's"
>>> '"Sim", eles disseram.'
'"Sim", eles disseram.'
>>> "\"Sim\", eles disseram."
'"Sim", eles disseram.'
```

- *Strings* podem ser concatenadas (juntadas) usando o operador +, e repetidas com *

```
>>> # 'a' seguido de 'ra' duas vezes
>>> 'a' + 2 * 'ra'
'arara'
```

- *Strings* podem ser indexadas, com o primeiro caractere tendo índice 0, e pode-se utilizar índices negativos, para contar da direita para a esquerda

```
>>> palavra = 'Python'
>>> palavra[0]  # caractere na posição 0
'P'
>>> palavra[5]  # caractere na posição 5
'n'
>>> palavra[-1]  # último caractere
```

```
'n'
>>> palavra[-2]  # penúltimo caractere
'o'
>>> palavra[-6]
'P'
```

- O fatiamento (*slicing*) também é suportado. Enquanto a indexação é usado para o acesso a um único caractere, o fatiamento permite a obtenção de sub-sequências. O começo sempre é incluído, enquanto o final é sempre excluído, o que garante que `s[:i] + s[i:]` é sempre igual à `s`

```
>>> palavra[0:2]  # caractere da posição 0 (incluído) até 2 (excluído)
'Py'
>>> palavra[2:5]  # caractere da posição 2 (incluído) até 5 (excluído)
'tho'
>>> palavra[:2] + palavra[2:]
'Python'
>>> palavra[:4] + palavra[4:]
'Python'
```

- Os índices de fatiamento possuem alguns padrões úteis: a omissão do primeiro item tem 0 como padrão, enquanto que a omissão do segundo tem como padrão o comprimento da *string*

```
>>> palavra[:2]  # caracteres do começo até a posição 2 (excluído)
'Py'
>>> palavra[4:]  # caracteres da posição 4 (incluído) até o final
'on'
>>> palavra[-2:]  # caracteres da penúltima posição (incluído) até o final
'on'
```

- A função `len()` retorna o comprimento da string

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
```

A.4 Estruturas de controle de fluxo

enquanto (*while*) Repete uma sequência de ações enquanto a condição for verdadeira

```
>>> # Sequência de Fibonacci:
... # a soma de dois elementos define o próximo
... a, b = 0, 1  # atribuição múltipla (a recebe 0, b recebe 1)
>>> while a < 10:  # a condição é o valor de a ser menor que 10
...     print(a)
...     a, b = b, a+b
...
0
1
1
2
3
5
8
```

se (*if*) Executa uma ação se a condição for verdadeira. Pode ter inúmeras outras condições caso a primeira seja falsa (comando `elif`) e uma caso todas sejam falsas (comando `else`)

```
>>> x = int(input("Por favor, digite um inteiro: "))
Por favor, digite um inteiro: 42
>>> if x < 0:
...     x =  0
...     print('Valor negativo alterado para zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Unitário')
... else:
...     print('Mais')
...
Mais
```

para (*for*) Itera ao longo dos itens de uma sequência

```
>>> # Mede algumas strings:
>>> palavras = ['gato', 'janela', 'marmota']
>>> for palavra in palavras:
...     print(palavra, len(palavra))
...
gato 4
janela 6
marmota 7
```

faixa (*range*) Gera progressões aritméticas. É útil para iterar ao longo de uma sequência de números. Os argumentos são, na ordem, **início** (primeiro valor da sequência, inclusivo), **fim** (valor final da sequência, exclusivo) e **passo** (de quanto em quanto os valores são contados). Caso tenha apenas dois argumentos, assume que o passo é 1, e caso só tenha um argumento, assume que o primeiro valor é 0

```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
"""
range(5, 10)
5, 6, 7, 8, 9

range(0, 10, 3)
0, 3, 6, 9

range(-10, -100, -30)
-10, -40, -70
```

```
"""
```

A.5 Funções

- Funções em *Python* são definidas pela palavra-chave `def`, seguida do nome da função e a lista de parâmetros entre parêntesis
- Todas as definições dentro da função precisam estar endentadas
- A primeira linha de definição pode ser uma sequência de texto, chamada de *docstring*. Se presente, é usada para descrever a função e seu uso [1]

```
>>> def fib(n):    # escreve a sequência de Fibonacci até n
...     """Escreve a sequência de Fibonacci até n."""
...     a, b = 0, 1
...     while a < n:
...         # o 'end' aqui indica que os valores serão separados por espaços
...         # ao invés de separados por linha (o padrão)
...         print(a, end=' ')
...         a, b = b, a+b
...     print()
...
>>> # Agora chama a função que acabou de ser definida:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

A.6 Bibliotecas

- Bibliotecas (módulos) no *Python* são importadas usando a sintaxe abaixo:

```
import sys
```

- Podem ser importadas atribuindo-se um apelido:

```
import builtins as bt
```

[1] Mais informações sobre *docstrings* podem ser encontradas na documentação online do *Python* (em inglês)

- Funções específicas podem ser importadas diretamente:

```
import sound.effects.echo
# ou
from sound.effects import echo
```

- E também podem ter apelidos:

```
import sound.effects.echo as see
# ou
from sound.effects import echo as see
```

- Todas as funções de um pacote podem ser importadas (não recomendável):

```
from sound.effects import *
```

- Códigos de outros arquivos podem ser importados como módulos:

```
import meu_arquivo
# ou
from meu_arquivo import minha_funcao
```

A.6.1 Numpy

- Biblioteca para processamento matemático (HARRIS et al., 2020)
- Geralmente é importada usando

```
import numpy as np
```

- O elemento base é o vetor (*array*), que possui diversos atributos e métodos associados a ele

```
>>> a = np.array([2, 3, 4])
>>> a
array([2, 3, 4])
>>> a.shape
(3,)
>>> a.ndim
1
```

- É comum não se saber o conteúdo futuro de um vetor, mas o tamanho sim, e por isso existem funções para criar vetores com conteúdos e tamanhos específicos:

  zeros Cria um vetor preenchido com zeros

  ```
  >>> np.zeros((3,4))
  array([[0., 0., 0., 0.],
  [0., 0., 0., 0.],
  [0., 0., 0., 0.]])
  ```

  ones Cria um vetor preenchido com 1

  ```
  >>> np.ones((2,3))
  array([[1., 1., 1.],
  [1., 1., 1.]])
  ```

  empty Cria um vetor com valores aleatórios extremamente pequenos, obtidos com base no que está presente na memória, e por isso é ligeiramente mais rápido que os anteriores para ser gerado

  ```
  >>> np.empty((2, 4))
  array([[4.67379042e-310, 0.00000000e+000, 6.79038654e-313,
  2.22809558e-312],
  [2.14321575e-312, 2.46151512e-312, 2.41907520e-312,
  5.97819431e-322]])
  ```

  arange Cria um vetor com uma sequência de números com um passo

  ```
  >>> np.arange(10, 30, 5) # argumentos: inicio, fim, passo
  array([10, 15, 20, 25])
  ```

  linspace Cria um vetor com uma sequência de números com tamanho específico

  ```
  >>> np.linspace(0, 2, 9) # 9 números de 0 até 2
  array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  ])
  ```

- O pacote também contém funções matemáticas comuns (*sin*, *cos*, *exp*, *sqrt*, etc)

```
>>> B = np.arange(3)
>>> B
array([0, 1, 2])
>>> np.exp(B)
```

```
array([1.        , 2.71828183, 7.3890561 ])
>>> np.sqrt(B)
array([0.        , 1.        , 1.41421356])
>>> C = np.array([2., -1., 4.])
>>> np.add(B, C)
array([2., 0., 6.])
```

- Indexação e fatiamento (*slices*) também são possíveis em vetores (semelhante à listas):

```
>>> a = np.arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> # equivalente a a[0:6:2] = 1000;
>>> # do início até a posição 6, exclusivo, define cada segundo elemento para
>>> a[:6:2] = 1000
>>> a
array([1000,    1, 1000,   27, 1000,  125,  216,  343,  512,  729])
>>> a[::-1]  # a reverso
array([ 729,  512,  343,  216,  125, 1000,   27, 1000,    1, 1000])
>>> for i in a:
...     print(i**(1 / 3.))
...
9.999999999999998
1.0
9.999999999999998
3.0
9.999999999999998
4.999999999999999
5.999999999999999
6.999999999999999
```

```
7.999999999999999
8.999999999999998
```

- Mais comandos e exemplos do *Numpy* estão na documentação online (em inglês)

A.6.2 Matplotlib

- Biblioteca para gerar visualizações estáticas, animadas ou interativas (gráficos) (HUNTER, 2007)
- Geralmente é importada usando

```
import matplotlib.pyplot as plt
```

frequentemente junto com a *Numpy*

- O elemento base é a figura (*Figure*), cada uma contendo um ou mais eixos (*Axes*), isto é, a área onde os pontos podem ser especificados em termos de coordenadas x-y, theta-r, x-y-z, etc.

```
import matplotlib.pyplot as plt
import numpy as np
fig, ax = plt.subplots() # cria uma figura contendo um único eixo
ax.plot([1, 2, 3, 4], [1, 4, 2, 3] # exibe alguns dados no eixo
```

- Tipos de gráficos comuns:

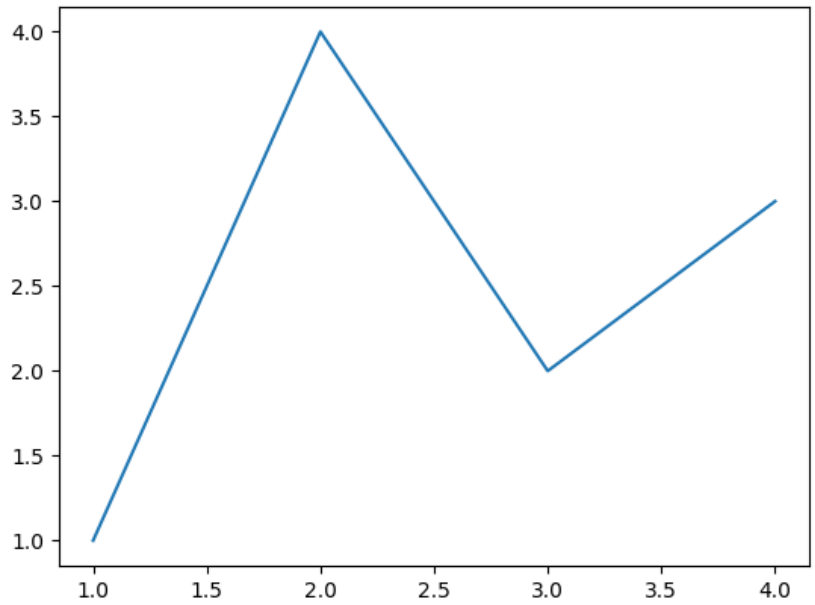
plot Plota *x* vs *y* como linhas e/ou marcadores

```
>>> plot(x, y)        # plota x e y usado tipo padrão de linha e cor
>>> plot(x, y, 'bo')  # plota x e y usando círculos azuis como marcadores
>>> plot(y)           # plota y usando x como um vetor de 0 até N-1
>>> plot(y, 'r+')     # igual acima, mas com + vermelhos de marcador
```

stem Plota linhas perpendiculares em relação à linha de base, começando na linha de base e terminando no topo, e coloca um marca nesse lugar. Normalmente, *x* são as posiçoes e *y* os topos.

```
x = np.linspace(0.1, 2 * np.pi, 41)
y = np.exp(np.sin(x))
```

Figura 32 – Exemplo simples de plot



Fonte: O autor (2023)

```
plt.stem(x, y)
plt.show()
```

scatter Plota y vs x com diferentes tamanhos e cores dos marcadores

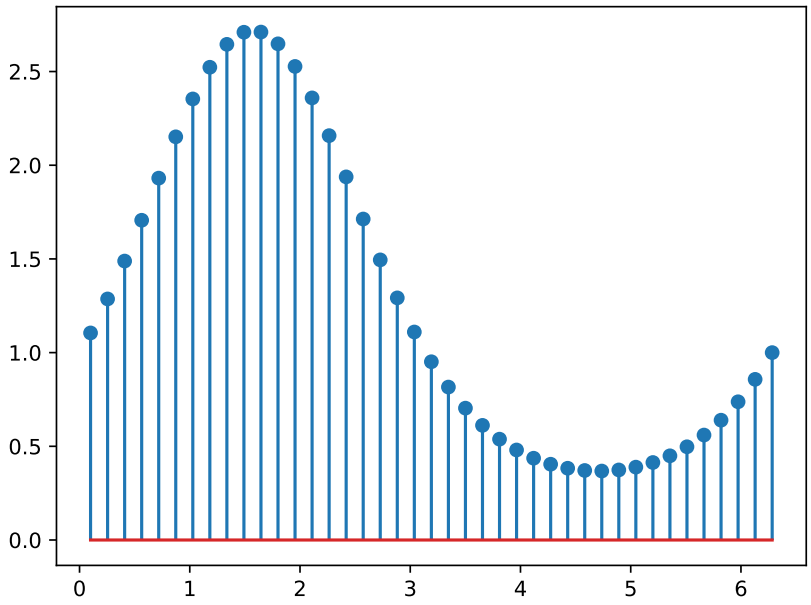
```
# fixa a semente de números aleatórios
np.random.seed(19680801)

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2  # 0 até 15 pontos o tamanho do raio

plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```

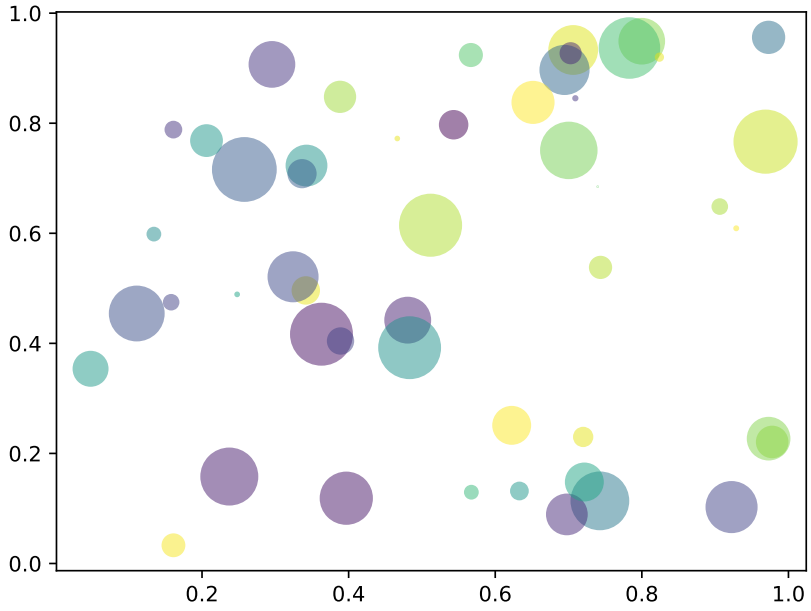
- Mais comandos e exemplos do *Matplotlib* estão na documentação online (em inglês)

Figura 33 – Gráfico do tipo Stem



Fonte: O autor (2023)

Figura 34 – Gráfico de dispersão (scatter)



Fonte: O autor (2023)

A.7 Dicas gerais

- Use 4 espaços para endentação, e não tabulações (alguns editores de texto convertem automaticamente as tabulações para 4 espaços, vale verificar)
- Nomeie as variáveis de acordo com o que elas se referem de maneira explícita (usar `area` ao invés de `a`, por exemplo)
- Nomeie variáveis e funções com letras minúsculas e palavras separadas por espaço (esse método é chamado de `snake_case`)
- Escreva linhas com até 79 caracteres, e até 72 se for linha com comentário (a maioria dos editores de texto exibe o número de caracteres)
- Use linhas em branco para separar funções ou longos blocos de texto dentro de funções
- Quando possível, coloque o comentário na linha a que se refere (respeitando o limite de caracteres citado acima)
- Use *docstrings* para arquivos e funções, descrevendo de maneira adequada o uso
- Use espaços entre operadores e depois de vírgulas (ex.: `a = f(1, 2) + g(3, 4))`
- Mais comandos e recursos do *Python* podem ser encontrados no tutorial online (em inglês), e dicas de estilo (como deixar o código "bonito"para o *Python*) estão no documento chamado PEP 8 (em inglês)