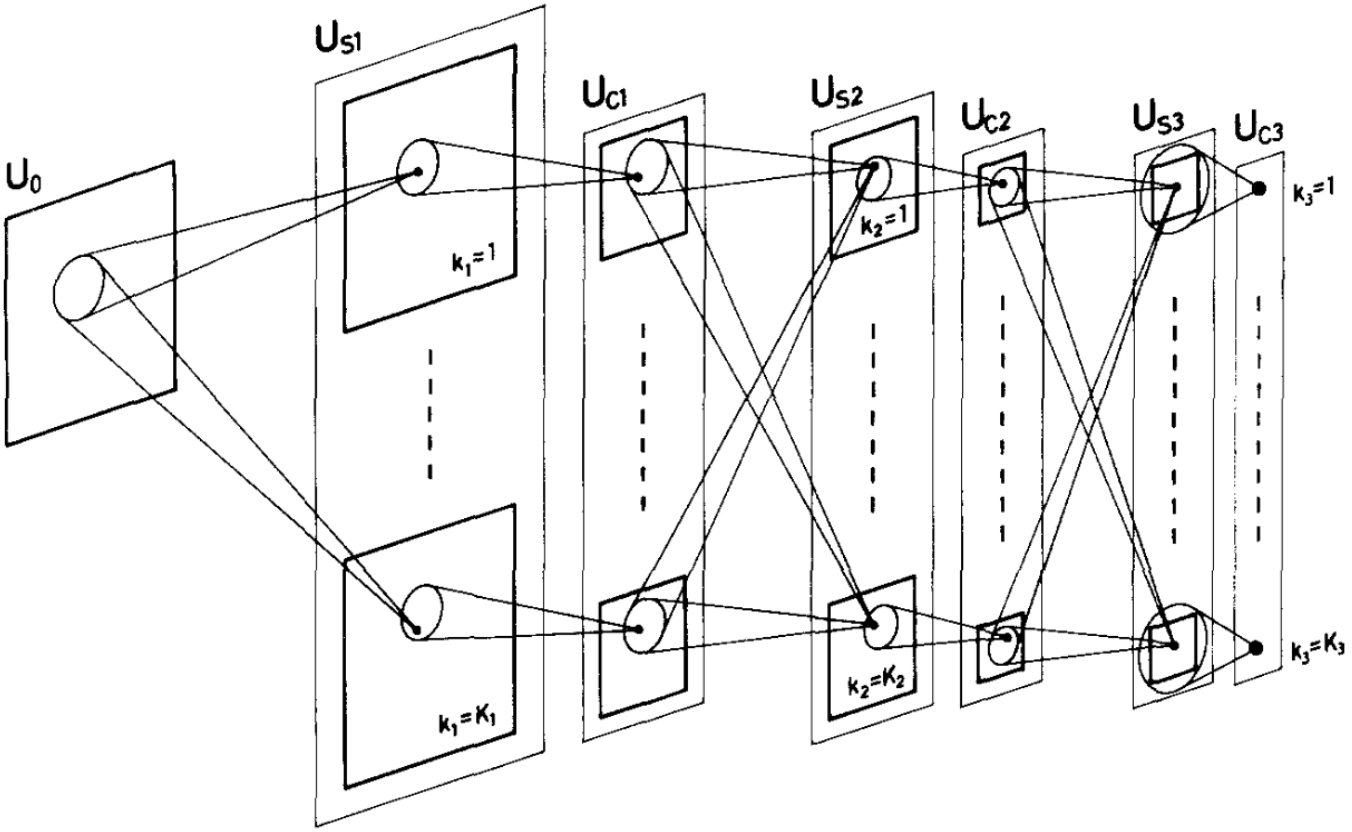
UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

Modelagem Neurocomputacional do Córtex Sensorial

Estudo sobre as Similaridades entre Redes Neurais Artificiais Profundas e os Córtices Visual e Auditivo.



Marcelo Henrique Lopes Ferreira

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

O Neocognitron (imagem ilustrada a pela capa) é a primeira rede neural convolucional (CNN) a ser proposta. Elaborado por Kunihiko Fukushima em 1979, foi um modelo desenvolvido com base em descobertas fundamentais da neurociência da época, especialmente os experimentos de David Hubel e Torsten Wiesel, que investigaram a organização funcional do córtex visual em mamíferos.

A proposta do Neocognitron reflete princípios encontrados na arquitetura do sistema visual humano, como a existência de camadas hierárquicas que processam informações sensoriais de forma progressiva, desde características simples, como bordas, até composições mais complexas, como formas e padrões.

A ilustração do modelo na capa busca capturar a essência dessa conexão entre neurociência e inteligência artificial, na qual destaca-se a interseção entre o funcionamento do cérebro e os avanços tecnológicos na modelagem computacional. Essa abordagem reflete o tema central deste trabalho: a aplicação de modelos de redes neurais como ferramenta para um entendimento mais aprofundado do funcionamento do cérebro humano.

UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA (INF)

MARCELO HENRIQUE LOPES FERREIRA

Modelagem Neurocomputacional do Córtex Sensorial

Estudo sobre as Similaridades entre Redes Neurais Artificiais Profundas e os Córtices Visual e Auditivo

Goiânia
2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): MARCELO HENRIQUE LOPES FERREIRA

Título do trabalho: Modelagem Neurocomputacional do Córtex Sensorial

Estudo sobre as Similaridades entre Redes Neurais Artificiais Profundas e os Córtices Visual e Auditivo

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Marcelo Henrique Lopes Ferreira**, **Discente**, em 12/01/2025, às 15:08, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

Documento assinado eletronicamente por **Fernando Marques Federson**, **Professor do Magistério Superior**, em 15/01/2025, às 16:24, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5089797** e o código CRC **5B9A3A8B**.

Referência: Processo nº 23070.001595/2025-59 SEI nº 5089797

MARCELO HENRIQUE LOPES FERREIRA

Modelagem Neurocomputacional do Córtex Sensorial

Estudo sobre as Similaridades entre Redes Neurais Artificiais Profundas e os Córtices Visual e Auditivo

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.
Orientador: Prof. Dr. Fernando Marques Federson

Goiânia

2025

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

FERREIRA, MARCELO HENRIQUE LOPES
Modelagem Neurocomputacional do Córtex Sensorial [manuscrito] : Estudo sobre as Similaridades entre Redes Neurais Artificiais Profundas e os Córtices Visual e Auditivo / MARCELO HENRIQUE LOPES FERREIRA. - 2025.
217 f.

Orientador: Prof. Dr. Fernando Marques Federson.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Instituto de Informática (INF), Inteligência Artificial, Goiânia, 2025.

1. inteligência artificial. 2. neurociência. 3. modelagem. I. Federson, Fernando Marques , orient. II. Título.

CDU 004

MARCELO HENRIQUE LOPES FERREIRA

Modelagem Neurocomputacional do Córtex Sensorial

Estudo sobre as Similaridades entre Redes Neurais Artificiais Profundas e os Córtices Visual e Auditivo

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Data da Aprovação: 17 de dezembro de 2024.

Prof. Dr. Fernando Marques Federson
Orientador (INF-UFG)

Prof. Dr. Aldo André Díaz Salazar
Coordenador de TCC do BIA (INF-UFG)

Prof. Dr. Anderson da Silva Soares
Coordenador do BIA (INF-UFG)

Prof. Dr. Iwens Gervasio Sene Junior
(INF-UFG)

MARCELO HENRIQUE LOPES FERREIRA

Modelagem Neurocomputacional do Córtex Sensorial

Estudo sobre as Similaridades entre Redes Neurais Artificiais Profundas e os Córtices Visual e Auditivo

RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Modelagem Neurocomputacional (Córtex Auditivo e Visual)**. Uma ilustração e sua narrativa descrevem os períodos de trabalho. Os Apêndices contêm os Termos de Aceite de Entrega e os resultados obtidos durante cada período de trabalho.

Palavras-chave: inteligência artificial, modelos grandes de linguagem, geração automática de datasets.

ABSTRACT

This Course Completion Report aims to bring together the results of my journey to become an expert in **Neurocomputational Modeling (Auditory and Visual Cortex)**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.

Keywords: artificial intelligence, large language models, automatic dataset generation.

Goiânia

2025

Minha Jornada

Marcelo Henrique Lopes Ferreira

Especialista em: Modelagem Neurocomputacional (Córtex Auditivo e Visual)

MINHA JORNADA

Nome: MARCELO HENRIQUE LOPES FERREIRA

Especialidade: Modelagem Neurocomputacional

Objetivo deste documento

Durante o processo da disciplina Residência em IA[1], foram gerados diversos resultados na construção da minha especialização. A cada semana, um conjunto de resultados foi formalizado por um Termo de Aceite de Entrega e avaliado por uma banca, considerando o planejado e o realizado para o período. Este documento tem como objetivo descrever esses resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados.

Minha Jornada

Minha jornada começou na **Semana 1**, na qual tive um contato inicial com a área da neurociência, explorando sua história e os frameworks disponíveis. Para compreendê-la melhor, produzi dois notebooks Colab para investigar os primeiros modelos de neurônios artificiais, o de McCulloch-Pitts e o de Rosenblatt, e elaborei um slide sobre a evolução histórica da neurociência em geral, a fim de entender como surge essa área de estudo do cérebro. Ao investigar os frameworks, descobri a plataforma BrainScore, que permite a comparação entre diferentes modelos de redes neurais artificiais profundas (DNN) e regiões específicas do cérebro ativadas. Na **Semana 2**, aprofundei meus estudos sobre o BrainScore e os fundamentos em neurociência, com foco em neurofisiologia, para entender exatamente como as DNNs se diferenciam dos neurônios biológicos. Desenvolvi um notebook dedicado a testar o framework BrainScore, investigando como importar modelos, conjuntos de dados e métricas de comparação entre as DNNs e dados neurais. Os materiais relacionados a estas duas Semanas podem ser encontrados no **Apêndice 1.**

[1] Dez semanas, entre setembro de 2024 e dezembro de 2024.

Com um framework já definido, foi necessário compreender melhor a área para realizar aplicações. Dessa forma, a **Semana 3** foi marcada por um aprofundamento na literatura, com a seleção e leitura de artigos científicos. Assim, estabeleci uma direção de pesquisa: é possível estudar o cérebro utilizando DNNs por meio de comparações que permitam desenvolver hipóteses falsificáveis. O resultado foi um compêndio de diversos artigos selecionados, organizados em uma tabela e avaliados quanto ao potencial de leitura mais detalhada no futuro, além de um breve resumo do artigo “How does the brain solve visual object recognition?” de J.J. DiCarlo, importante referência anterior ao BrainScore, que explica conceitos básicos sobre a percepção visual pelo cérebro e disponibiliza um conjunto de dados comumente utilizado pela plataforma. Com esse material selecionado, a próxima semana concentrou-se em seu estudo e em outros trabalhos correlatos. Os materiais relacionados a esta Semana podem ser encontrados no **Apêndice 2.**

Nas **Semanas 4 e 5**, mergulhei na leitura de materiais marcantes. Deparei-me com pesquisas recentes e fascinantes, como a da FlyWire, que realizou um conectoma ao mapear todo o cérebro de uma Drosophila utilizando inteligência artificial (IA), além de conhecer uma competição de detecção de EEG do Instituto Santos Dumont, que despertou meu interesse pelo tema, levando-me a considerar participar desse desafio. Contudo, o material mais investigado foi o artigo de Daniel Yamins, “Using goal-driven deep learning models to understand sensory cortex”, e trabalhos relacionados. Nele, o autor revisa avanços recentes em redes neurais convolucionais hierárquicas orientadas a objetivos e explora como essa abordagem pode aprofundar a compreensão do desenvolvimento e organização do processamento sensorial cortical, especialmente o visual.

Percebi prontamente que, apesar de mencionado no artigo técnico da plataforma BrainScore (“Brain-Score: Which Artificial Neural Network for Object Recognition is most Brain-Like?”), o modelo proposto por Yamins não está pronto para uso direto. Inicialmente, tentei reproduzi-lo com base na arquitetura de redes convolucionais hierárquicas, a partir do conjunto de dados comumente usado na plataforma BrainScore, descrito por Rajalingham em “Large-Scale, High-Resolution Comparison of the Core Visual Object Recognition Behavior of Humans, Monkeys, and State-of-the-Art Deep Artificial Neural Networks”. Posteriormente, realizei leituras adicionais de trabalhos relacionados do próprio autor,

desenvolvi um notebook para estudar uma abordagem clássica frequentemente mencionada (baseada em filtros de Gabor) e implementei uma arquitetura inspirada no artigo de Yamins. Após muito esforço e ainda com dúvidas sobre como obter os mesmos resultados do artigo estudado, entrei em contato com Yamins, que sugeriu concentrar esforços em abordagens mais recentes, dadas as dificuldades de versionamento e manutenção das bibliotecas utilizadas.

A investigação ao longo dessas duas semanas foi de extrema importância, não apenas por permitir o estudo de conceitos e termos até então pouco familiares, mas também por revelar os principais pesquisadores dessa área que eu acabara de conhecer. Os materiais relacionados a estas duas Semanas podem ser encontrados no **Apêndice 3**.

Nas **Semanas 6 e 7**, busquei investigar implementações mais recentes, não necessariamente apenas na área de visão. Primeiro, finalizei o estudo dos fundamentos básicos de neurofisiologia iniciado semanas antes. Na **Semana 6**, investiguei abordagens atuais, assistindo a painéis, lendo novos artigos e deparando-me com aplicações em Ciências Cognitivas. Assim, encontrei artigos relevantes sobre áudio e uma biblioteca promissora para visualização de dados neurais. O mais importante, porém, foi o contato com os artigos “A Task-Optimized Neural Network Replicates Human Auditory Behavior, Predicts Brain Responses, and Reveals a Cortical Processing Hierarchy”, de Kell, e “Many but not all deep neural network audio models capture brain responses and exhibit correspondence between model stages and brain regions”, de Tuckute, que, em vez de focarem no córtex visual, concentram-se no córtex auditivo. Já a **Semana 7**, foi dedicada a entender as limitações das pesquisas mais recentes e esclarecer dúvidas pessoais sobre a parte visual. Notei que arquiteturas como a YOLO não são populares nessa área. Testei, por meio do BrainScore, arquiteturas como a CVT e percebi que aquelas que combinam CNN e transformers apresentam resultados promissores. Finalizei minhas anotações sobre o córtex visual e constatei que poucas abordagens exploram de forma aprofundada tanto o córtex auditivo quanto o visual. Considerando que o processamento hierárquico da informação no córtex auditivo é semelhante ao do visual, e que muitas técnicas para comparar DNNs com dados neurais são similares, decidi prosseguir a pesquisa com o córtex auditivo. Os materiais relacionados a estas duas Semanas podem ser encontrados no **Apêndice 4**.

Em seguida, na **Semana 8**, decidi comparar modelos de transcrição automática de áudio (ASR) da NVIDIA com dados de ressonância magnética funcional (fMRI) de pacientes, inspirando-me em trabalhos semelhantes, como o de Y. Li em “Dissecting neural computations in the human auditory pathway using deep neural networks for speech”. Para isso, iniciei o desenvolvimento do código, disponibilizando-o em notebooks: um para compilar modelos de ASR disponíveis, outro para extrair ativações das redes e um terceiro para manipular o conjunto de dados. Na **Semana 9**, documentei os modelos da NVIDIA e, usando boa parte do código disponibilizado por Greta Tuckute em seu trabalho, escrevi um script para extrair as ativações de cada modelo, ajustando também o código de comparação. Expliquei todo o processo em documentos separados, buscando maior clareza. Os materiais relacionados a estas duas Semanas podem ser encontrados no **Apêndice 5**.

Finalmente, na **Semana 10**, gerei os gráficos finais e organizei estudos pendentes sobre o córtex auditivo. Baseando-me em um modelo criado por Alexander Kell et al., especificamente para tarefas de reconhecimento de fala, foi possível realizar comparações com o modelo Canary, estabelecendo uma baseline sólida para futuras investigações. Observou-se que, apesar de o Canary ser um modelo “estado da arte” em ASR, não apresenta tanta similaridade com o cérebro quanto a baseline puramente convolucional. Os materiais relacionados a esta última Semana podem ser encontrados no **Apêndice 6**.

Este percurso tem sido uma experiência profundamente enriquecedora, combinando aprendizado contínuo, experimentações práticas e reflexões teóricas voltadas para a investigação da relação entre IA e Neurociência. Ao longo desta jornada, adquiri conhecimentos fundamentais, explorei trabalhos relevantes e adentrei em uma área de extrema complexidade, da qual apenas comecei a desvendar a superfície. Pretendo, no futuro, aprofundar ainda mais meus estudos nesta temática, cuja exploração tem sido uma experiência marcante.

APÊNDICE 1

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 18 de set. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

A primeira semana teve o intuito de conhecer a área e sua história, estudar fundamentos básicos e conhecer frameworks propostos.

A fim de atender ao primeiro requisito, foi feita uma pesquisa ampla sobre a história da neurociência em si, e uma exploração sobre possíveis aplicações que envolvem Inteligência Artificial. O material foi reunido em forma de slides disponíveis aqui. Todos os materiais explorados podem ser vistos aqui.

Como exercício, apliquei alguns dos conhecimentos básicos obtidos e implementei alguns dos primeiros modelos inspirados pelo cérebro: O primeiro neurônio proposto e uma MLP:
https://colab.research.google.com/drive/1KErwT8DyRhRN9Z28tlqnbiIH4q_EFUG8?usp=sharing
https://colab.research.google.com/drive/1mvd_m58ng3FkhPCx3JQhJb5gGhq33BpE?usp=sharing

Os conhecimentos das ferramentas nesse caso já eram relativamente conhecidos (python, Torch, TensorFlow, Keras etc), então parti na investigação de um possível framework mais específico na área. Encontrei um artigo bastante citado:
https://www.nature.com/articles/s41593-019-0520-2

Investiguei melhor, e assisti uma aula que considero importante para minha investigação, realizada por um dos co-autores do artigo e que reforça sua ideia:
https://www.youtube.com/watch?v=6NOFtwKU3sA

Na aula, foi apresentada uma plataforma interessante que avalia o quão bem modelos preveem medições neurais e comportamentais do cérebro, além de outros artigos que foram comentados durante a aula e que pretendo investigar aprofundadamente na próxima entrega:
https://www.brain-score.org/
https://github.com/brain-score/vision
https://scholar.google.com/citations?user=1UvYRp0AAAAJ&hl=en

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Enfrentei uma dificuldade significativa em localizar um material que compilasse uma coletânea de estudos sobre a interseção entre neurociência e inteligência artificial. Esta dificuldade resultou na incerteza quanto ao foco exato da pesquisa em neurociência aplicada à IA e quanto ao quão avançado está esse nicho.

Para superar essa limitação, estabelecerei objetivos fundamentais: consolidar fundamentos teóricos em relação a neurociência que foram pouco explorados (ex: Qual a função do hipocampo que é tão comentado) , e demonstrar o conhecimento adquirido por meio de uma anotação básica dos materiais encontrados e investigar mais artigos que abordem especificamente a específica área de estudo mencionada.

Pretendo investigar melhor o brain-score, visto que é a principal plataforma que encontrei relacionada ao tema.

O objetivo principal é definir precisamente qual será o foco de estudo a partir de diante, e explorar melhor plataformas como o brain score para realizar testes de modelos já conhecidos.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[McCulloch-Pitts Neuron.ipynb]

```
# Baseado em https://github.com/kaisdukes/mcculloch-pitts-neuron.git

from dataclasses import dataclass
from enum import Enum, auto

# Aqui, criamos um tipo de dado
class InputType(Enum):
    EXCITATORY = auto()
    INHIBITORY = auto()

@dataclass
class Input:
    type: InputType
    value: bool
```

$$N_i(t) = \begin{cases} 1 & \text{se} \sum_{j=1}^{n} w_{ji} N_j(t-1) \geq \theta_i \\ 0 & \text{se} \sum_{j=1}^{n} w_{ji} N_j(t-1) < \theta_i \end{cases}$$

- $ N_i(t) $ é o estado do neurônio $ i $ no tempo $ t $.
- $ w_{ji} $ é o peso da conexão do neurônio $ j $ para o neurônio $ i $. No modelo de McCulloch e Pitts, esses pesos são fixos e geralmente assumem valores de 1 ou 0.
- $ N_j(t-1) $ é o estado do neurônio $ j $ no tempo anterior $ t-1 $.
- θ_i é o limiar de ativação do neurônio $ i $. Se a soma ponderada das entradas for maior ou igual a θ_i, o neurônio dispara $1 $; caso contrário, ele permanece inativo $ 0$.

```
from dataclasses import dataclass
from typing import List

@dataclass
class McpNeuron:
    inputs: List[Input]
    threshold: int

    def fire(self):
        sum_excitatory = sum(input.value for input in self.inputs if
input.type == InputType.EXCITATORY)
        has_inhibitory = any(input.value for input in self.inputs if
input.type == InputType.INHIBITORY)
```

```
        # O neurônio dispara se a soma das excitatórias for maior ou igual
ao limiar, se não houver nenhuma entrada inibitória ativa
        return sum_excitatory >= self.threshold and not has_inhibitory

# Uma entrada excitatória ativa e outra inibitória inativa
neuron = McpNeuron(
          inputs=[
              Input(type=InputType.EXCITATORY, value=True),
              Input(type=InputType.INHIBITORY, value=False)],
          threshold=1)
neuron.fire()
```

```
True
```

```
# Uma entrada excitatória ativa e outra inibitória ativa
neuron = McpNeuron(
    inputs=[
        Input(type=InputType.EXCITATORY, value=True),
        Input(type=InputType.INHIBITORY, value=True)],
    threshold=1)
neuron.fire()
```

```
False
```

```
# Duas entradas excitatórias ativas e limiar maio
neuron = McpNeuron(
    inputs=[
        Input(type=InputType.EXCITATORY, value=True),
        Input(type=InputType.EXCITATORY, value=True)],
    threshold=3)
neuron.fire()
```

```
False
```

```
# Mil entradas excitatória ativa e uma inibitória inativa
inputs_exc = [Input(type=InputType.EXCITATORY, value=True) for i in
range(1000)]
neuron = McpNeuron(
    inputs=inputs_exc + [Input(type=InputType.INHIBITORY, value=True)],
        threshold=1)
neuron.fire()
```

```
False
```

```
# Duas entradas inibitórias ativas
neuron = McpNeuron(
    inputs=[
        Input(type=InputType.INHIBITORY, value=True),
```

```
        Input(type=InputType.INHIBITORY, value=True)],
    threshold=1)
neuron.fire()
```

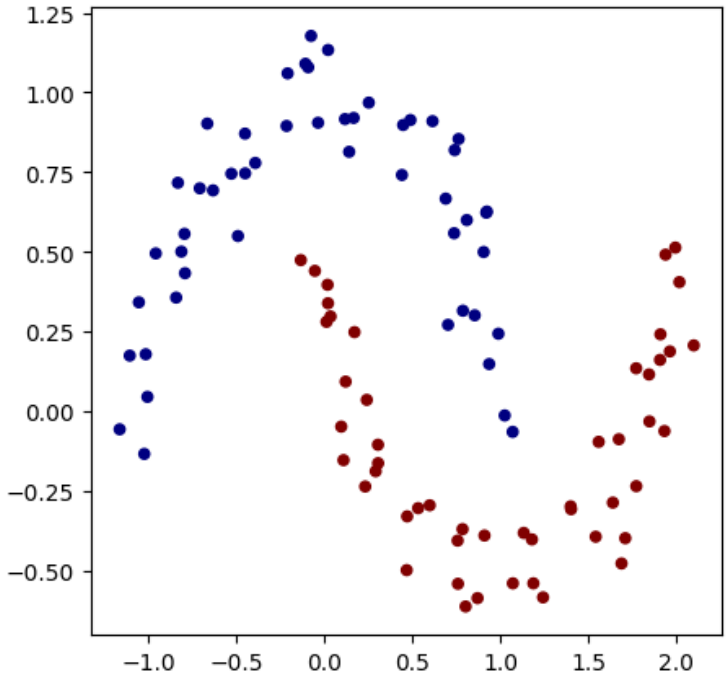
```
False
```

[Rosenblatt Perceptron + MLP.ipynb]

```
# Baseado em https://github.com/4ndr3lu15/VascoGrad/tree/main
```

Dados

```
from dataclasses import dataclass

@dataclass
class Data:
    X: any
    y: any

# Make up Data
import  matplotlib.pyplot as plt
from sklearn.datasets import make_moons, make_blobs
X, y = make_moons(n_samples=100, noise=0.1)

y = y*2 - 1

plt.figure(figsize=(5,5))
plt.scatter(X[:,0], X[:,1], c=y, s=20, cmap='jet')
plt.show()
```




```
data = Data(X, y)
data
```

```
Data(X=array([[-1.10863284,  0.17377957], …,
       [ 1.71027531, -0.39908355]]), y=array([-1, -1,  …,  1,  1,  1]))
```

Arquitetura

Utilis

```python
import math

class Value:
    def __init__(self, data, _children=(), _op='', label=''):
        self.data = data
        self.grad = 0
        self._backward = lambda: None
        self._prev = set(_children)
        self._op = _op
        self.label = label

    def __repr__(self):
        return f"Value(data={self.data})"

    def __add__(self, other):
        other = other if isinstance(other, Value) else Value(other)
        out = Value(self.data + other.data, (self, other), '+')

        def _backward():
            self.grad += 1.0 * out.grad
            other.grad += 1.0 * out.grad
        out._backward = _backward

        return out

    def __radd__(self, other):
        return self + other


    def __mul__(self, other):
        other = other if isinstance(other, Value) else Value(other)
        out = Value(self.data * other.data, (self, other), '*')

        def _backward():
            self.grad += other.data * out.grad
            other.grad += self.data * out.grad
        out._backward = _backward

        return out
```

```
    def __rmul__(self, other):
        return self * other

    def __pow__(self, other):
        assert isinstance(other, (int, float)), "only supporting int/float
powers for now"
        out = Value(self.data**other, (self,), f'**{other}')

        def _backward():
            self.grad += other * self.data ** (other - 1)
        out._backward = _backward

        return out

    def __truediv__(self, other):
        return self * other**-1

    def __rtruediv__(self, other):
        return self * other**-1

    def __neg__(self):
        return self * -1

    def __sub__(self, other):
        return self + (-other)

    def __rsub__(self, other):
        return other + (-self)

    def tanh(self):
        x = self.data
        t = (math.exp(2*x) - 1)/(math.exp(2*x) + 1)
        out = Value(t, (self, ), 'tanh')

        def _backward():
            self.grad += (1 - t**2) * out.grad
        out._backward = _backward

        return out

    def step(self):
        x = self.data
        t = 1 if x >= 0 else 0
        out = Value(t, (self, ), 'step')
```

```
        def _backward():
            # A derivada da função degrau é 0 em todos os pontos, exceto em
x = 0 (onde é indefinida)
            # Mas como em implementações práticas é usada a aproximação,
não há gradiente útil.
            self.grad += 0

        out._backward = _backward

        return out
    def exp(self):
        x = self.data
        out = Value(math.exp(x), (self, ), 'exp')

        def _backward():
            self.grad += out.data * out.grad
        out._backward = _backward

        return out

    def backward(self):
        topo = []
        visited = set()
        def build_topo(v):
            if v not in visited:
                visited.add(v)
                for child in v._prev:
                    build_topo(child)
                topo.append(v)
        build_topo(self)

        self.grad = 1.0
        for node in reversed(topo):
            node._backward()
```

Perceptron

$$\hat{y} = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

- x_i são os inputs (valores de entrada),
- w_i são os pesos associados a cada input,
- b é o **viés**,

- $f(\square)$ é a **função de ativação** (no caso do perceptron clássico, uma função **degrau**),
- $\hat{y}$ é a saída do perceptron.

```
import random

class Module:
    def parameters(self):
        return []

class Neuron(Module):

    def __init__(self, nin):
        # Inicializa com pesos aleatórios
        self.w = [Value(random.uniform(-1, 1)) for _ in range(nin)]
        self.b = Value(random.uniform(-1, 1))

    def __call__(self, x):
            # Passando valores por uma função de ativação
            act = sum((wi*xi for wi, xi in zip(self.w, x)), self.b)
            # utilizaremos essa função de ativação para resolver um
problema linear
            out = act.tanh()
            # out = act.step() # gradiente zero, nunca atualiza
            return out

    def parameters(self):
        return self.w + [self.b]


class Layer(Module):
    def __init__(self, nin, nout):
        self.neurons = [Neuron(nin) for _ in range(nout)]

    def __call__(self, x):
        outs = [n(x) for n in self.neurons]
        return outs[0] if len(outs) == 1 else outs

    def parameters(self):
        return [p for neuron in self.neurons for p in neuron.parameters()]

class MLP(Module):
    def __init__(self, nin, nouts):
        sz = [nin] + nouts
        self.layers = [Layer(sz[i], sz[i+1]) for i in range(len(nouts))]
```

```
    def __call__(self, x):
        for layer in self.layers:
            x = layer(x)
        return x

    def parameters(self):
        return [p for layer in self.layers for p in layer.parameters()]

model = MLP(2, [5, 5, 1])
```

Treinamento

Equações de Treinamento do Perceptron com Gradiente

1. Equação de Ativação

A saída $ y $ do perceptron é calculada como:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Onde:

- $ x_i $ são os inputs,
- $ w_i $ são os pesos,
- $ b $ é o bias,
- $ f(\cdot) $ é a função de ativação (no caso do perceptron clássico, a função **degrau**).

2. Função de Custo

O erro é dado pela diferença entre a saída esperada $ y_{\text{esperado}} $ e a saída prevista $ y $:

$$e = y_{\text{esperado}} - y$$

A função de custo $ L $ pode ser expressa como:

$$L = \frac{1}{2}e^2 = \frac{1}{2}\left(y_{\text{esperado}} - y\right)^2$$

Isso facilita o cálculo do gradiente durante o processo de otimização.

3. Gradiente para Atualização dos Pesos

A atualização dos pesos $ w_i $ segue a **descida de gradiente**. O gradiente da função de custo em relação aos pesos $ w_i $ é dado por:

$$\frac{\partial L}{\partial w_i} = -e \cdot x_i$$

Os pesos são então atualizados pela regra:

$$w_i \leftarrow w_i + \eta \cdot e \cdot x_i$$

Onde:

- $ \eta $ é a **taxa de aprendizado**,
- $ e $ é o erro calculado,
- $ x_i $ é o valor de entrada correspondente ao peso $ w_i $.

4. Gradiente para Atualização do Bias

O gradiente da função de custo em relação ao bias $ b $ é dado por:

$$\frac{\partial L}{\partial b} = -e$$

Assim, o bias é atualizado pela regra:

$$b \leftarrow b + \eta \cdot e$$

5. Resumo do Processo de Treinamento

O processo de atualização dos pesos e do bias ocorre iterativamente para cada amostra de treinamento até que o erro seja minimizado ou o número máximo de iterações seja atingido:

1. Calcule a saída do perceptron $ y $.
1. Calcule o erro $ e = y_{\text{esperado}} - y $.
2. Atualize os pesos $ w_i $ e o bias $ b $ de acordo com as regras de descida de gradiente:

$$w_i \leftarrow w_i + \eta \cdot e \cdot x_i$$

$$b \leftarrow b + \eta \cdot e$$

```
class Optimizer:
    def __init__(self, model: MLP, learning_rate: float = 1e-2) -> None:
```

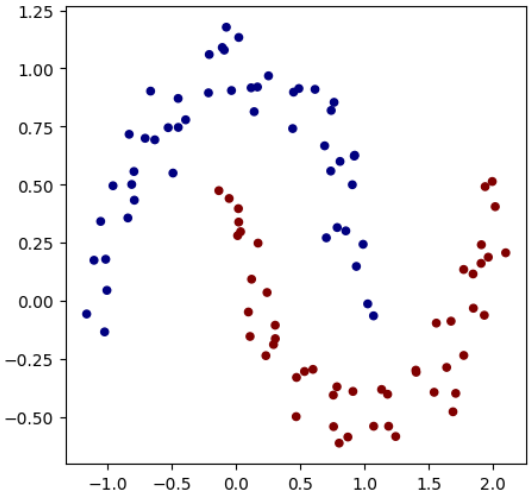
```
        self.model = model
        self.learning_rate = learning_rate

    def step(self):
        # Atualiza os parâmetros do modelo com base nos gradientes
        for p in self.model.parameters():
            p.data -= self.learning_rate * p.grad

    def zero_grad(self):
        for p in self.model.parameters():
            p.grad = 0.0

class Learner:
    def __init__(self, model: MLP, optimizer: Optimizer):
        self.model = model
        self.optimizer = optimizer

    def predict(self, x):
        return self.model(x)

    def update(self, loss: Value) -> None:
        # Backpropagation
        self.optimizer.zero_grad()  # Zera os gradientes antes do backprop
        loss.backward()  # Calcula os gradientes via retropropagação
        self.optimizer.step()  # Atualiza os parâmetros do modelo


class Evaluator:
    """Classe para calcular a função de perda"""
    def __init__(self, loss_function):
        self.loss_fn = loss_function

    def get_loss(self, y, y_hat):
        # Calcula a perda  entre predições e valores reais
        return self.loss_fn(y_hat, y)

class Trainer:
    def __init__(self, data: Data, learner: Learner, evaluator: Evaluator):
        self.data = data
        self.learner = learner
        self.evaluator = evaluator

    def one_epoch(self):
        y_hat = [self.learner.predict(x) for x in self.data.X]
        loss = self.evaluator.get_loss(self.data.y, y_hat)
        self.learner.update(loss)
```

```
        return loss

    def run(self, n_epochs : int):
        for epoch in range(n_epochs):
            loss = self.one_epoch()
            if epoch % 10 == 0:
                print(f' epoch: {epoch} | loss : {loss.data}')
        print("Done!")

def loss_function(y, y_hat):
    return sum([(yout - ygt)**2 for ygt, yout in zip(y, y_hat)])

optimizer = Optimizer(model)
learner = Learner(model, optimizer)
evaluator = Evaluator(loss_function)

trainer = Trainer(data, learner, evaluator)

trainer.run(300)
```

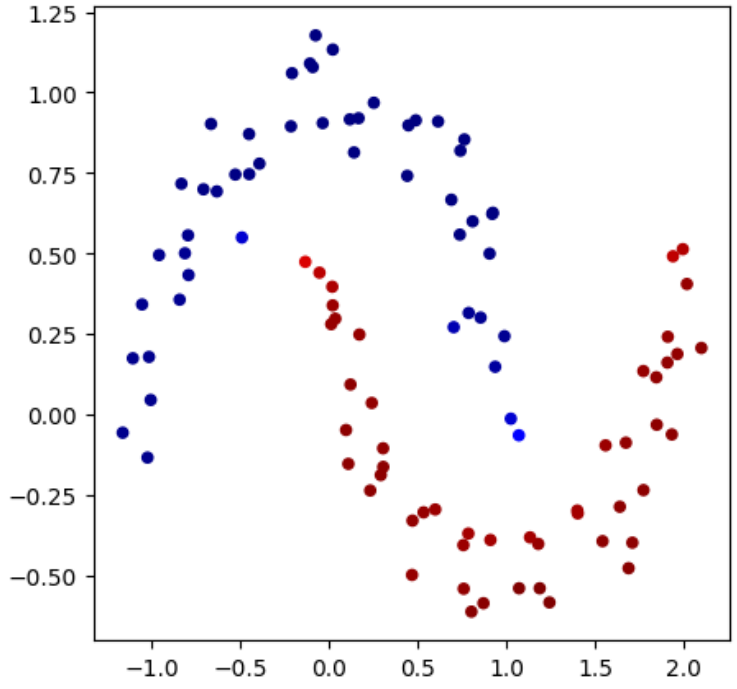
```
 epoch: 0 | loss : 119.61652545276968

  ...
 epoch: 290 | loss : 0.3188832372125559
Done!
```

Resultados

```
# REAL
import  matplotlib.pyplot as plt
from sklearn.datasets import make_moons, make_blobs
X, y = data.X, data.y

plt.figure(figsize=(5,5))
plt.scatter(X[:,0], X[:,1], c=y, s=20, cmap='jet')
plt.show()
```




```
# PREDITO
import numpy as np
y_hat = [pred.data for pred in [learner.predict(x) for x in data.X]]

plt.figure(figsize=(5,5))
plt.scatter(X[:,0], X[:,1], c=y_hat, s=20, cmap='jet')
plt.show()
```

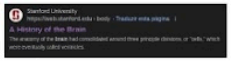


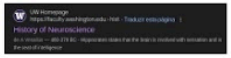
[Slides História da Neurociência]

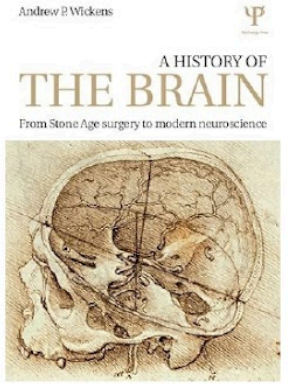
Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Breve História da Neurociência

Fontes

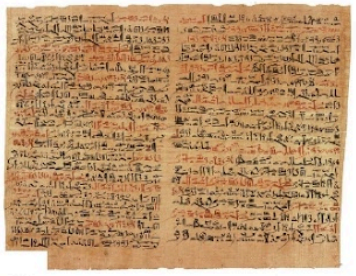






1600 a.C

- Manual de cirurgia militar do Egito Antigo
  - Lesões, deslocamentos, ferimentos, fraturas e tumores
- **Primeira menção conhecida do cérebro em texto**
- Casos aparentes de traumatismo craniano, ataque epiléptico e até mesmo afasia

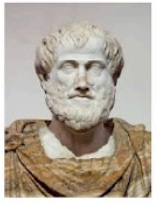


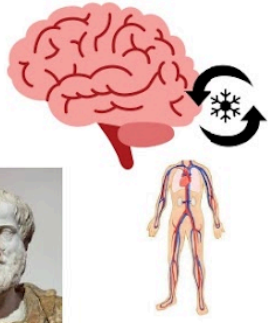
Papiro de Edwin Smith

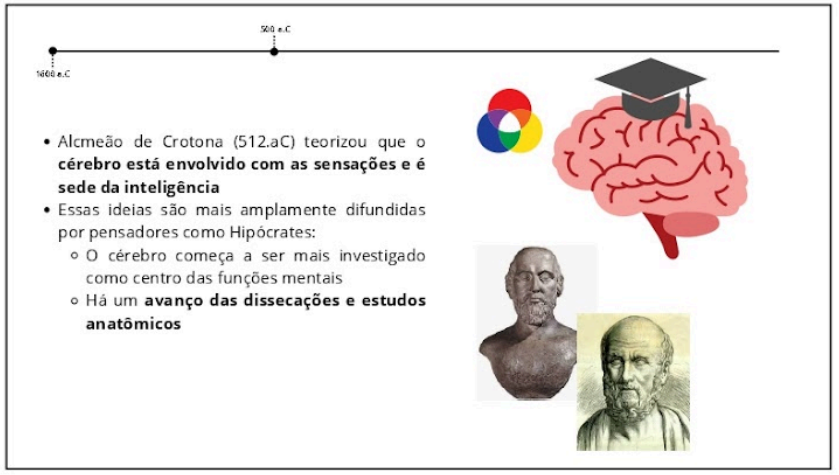
500 a.C

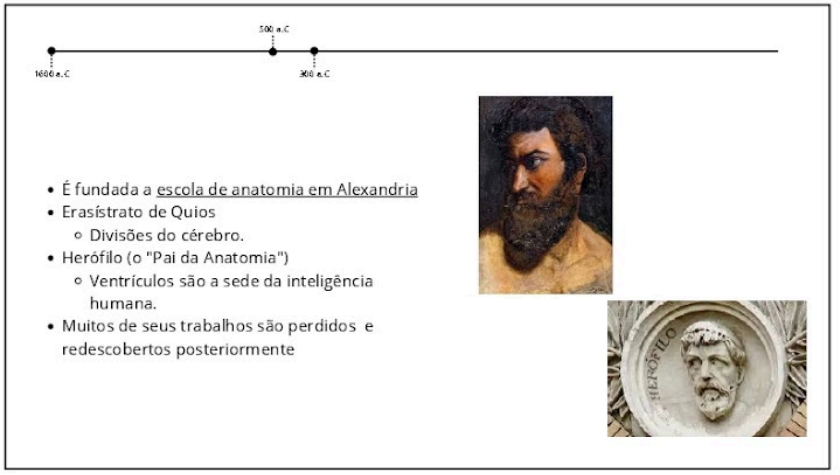
1600 a.C

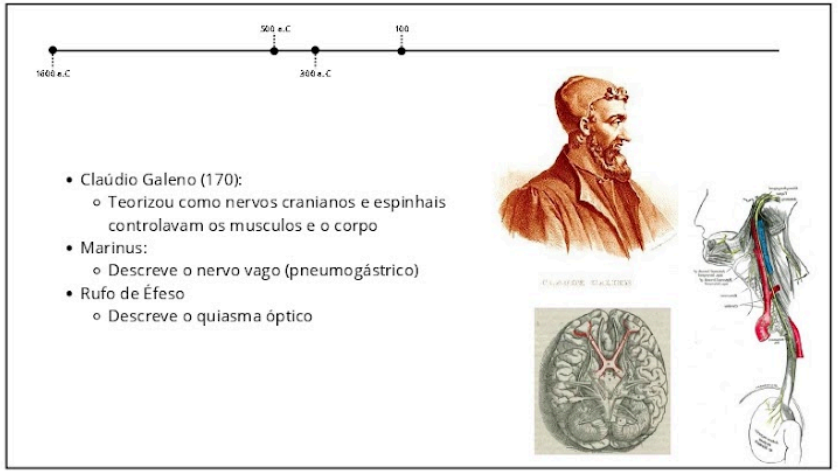
- Assim como no Egito Antigo, **a ideia de que o coração era responsável pela conciência ainda era muito comum na Grécia**
- Para Aristóteles, o cérebro era responsável por resfriar o coração e permitir a circulação do espírito.
- sensus communis: o lugar onde os espíritos se reuniam

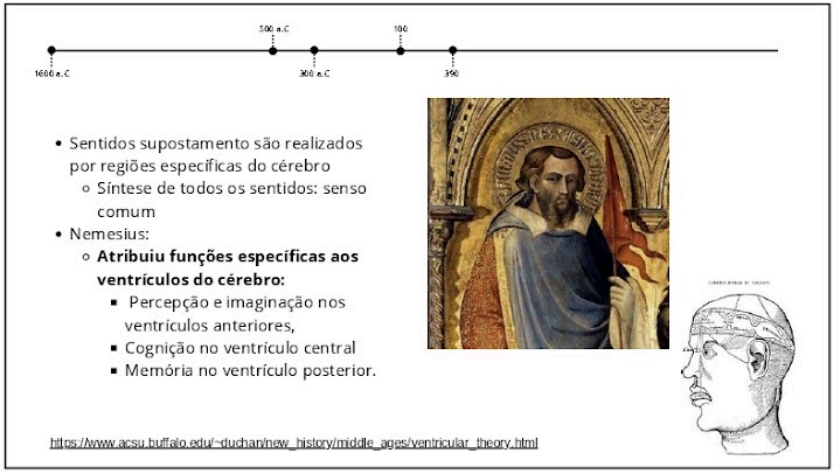


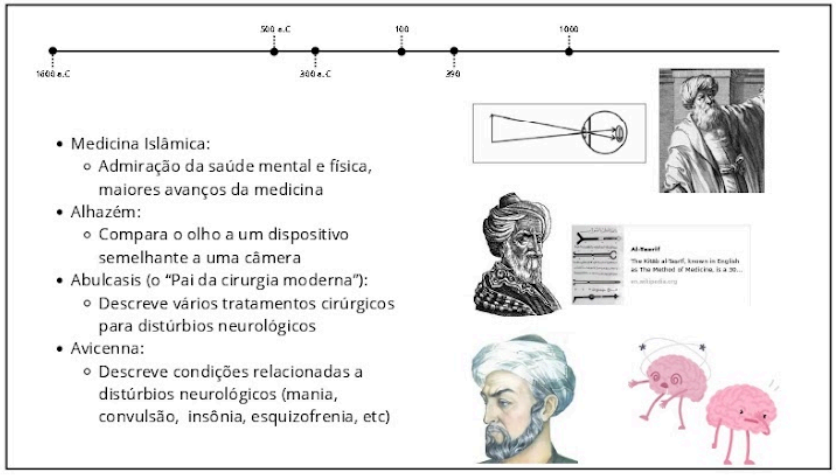



500 a.C
1600 a.C
• Alcmeão de Crotona (512.aC) teorizou que o **cérebro está envolvido com as sensações e é sede da inteligência**
• Essas ideias são mais amplamente difundidas por pensadores como Hipócrates:
◦ O cérebro começa a ser mais investigado como centro das funções mentais
◦ Há um **avanço das dissecações e estudos anatômicos**

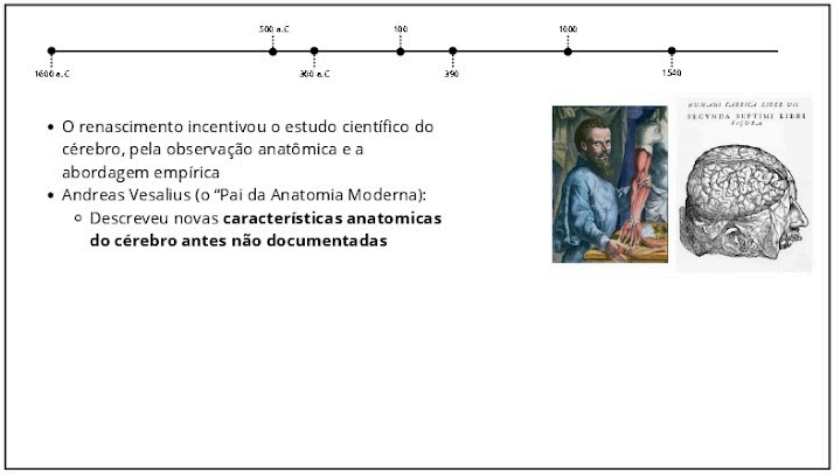

300 a.C
1600 a.C
• É fundada a escola de anatomia em Alexandria
• Erasístrato de Quios
◦ Divisões do cérebro.
• Herófilo (o "Pai da Anatomia")
◦ Ventrículos são a sede da inteligência humana.
• Muitos de seus trabalhos são perdidos e redescobertos posteriormente
HERÓFILO


100
300 a.C
1600 a.C
• Claúdio Galeno (170):
◦ Teorizou como nervos cranianos e espinhais controlavam os musculos e o corpo
• Marinus:
◦ Descreve o nervo vago (pneumogástrico)
• Rufo de Éfeso
◦ Descreve o quiasma óptico

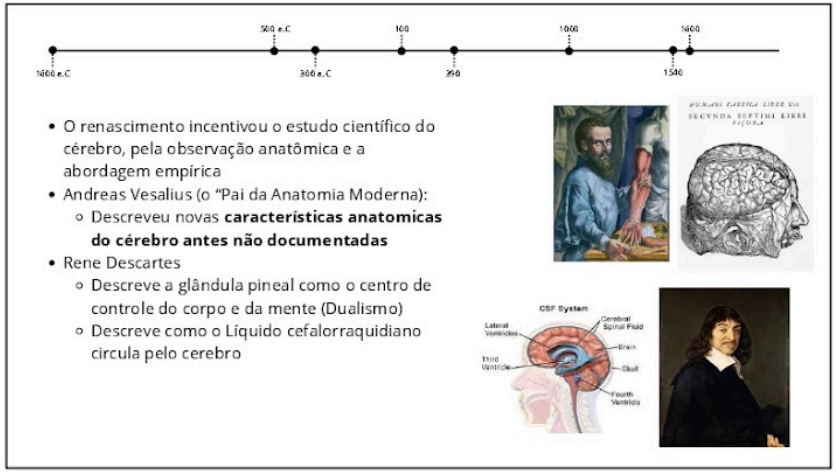

100
300 a.C
350
1600 a.C
• Sentidos supostamento são realizados por regiões específicas do cérebro
◦ Síntese de todos os sentidos: senso comum
• Nemesius:
◦ **Atribuiu funções específicas aos ventrículos do cérebro:**
▪ Percepção e imaginação nos ventrículos anteriores,
▪ Cognição no ventrículo central
▪ Memória no ventrículo posterior.
https://www.acsu.buffalo.edu/~duchan/new_history/middle_ages/ventricular_theory.html



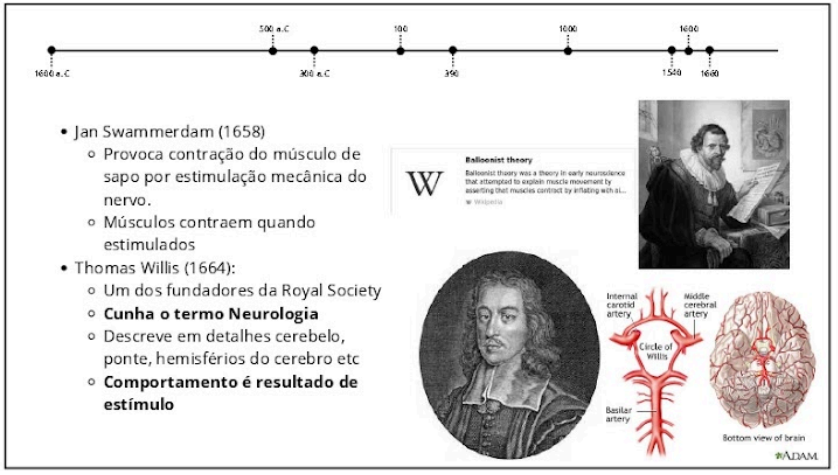
- Medicina Islâmica:
  - Admiração da saúde mental e física, maiores avanços da medicina
- Alhazém:
  - Compara o olho a um dispositivo semelhante a uma câmera
- Abulcasis (o "Pai da cirurgia moderna"):
  - Descreve vários tratamentos cirúrgicos para distúrbios neurológicos
- Avicenna:
  - Descreve condições relacionadas a distúrbios neurológicos (mania, convulsão, insônia, esquizofrenia, etc)



- O renascimento incentivou o estudo científico do cérebro, pela observação anatômica e a abordagem empírica
- Andreas Vesalius (o "Pai da Anatomia Moderna):
  - Descreveu novas **características anatomicas do cérebro antes não documentadas**



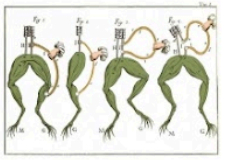
- O renascimento incentivou o estudo científico do cérebro, pela observação anatômica e a abordagem empírica
- Andreas Vesalius (o "Pai da Anatomia Moderna):
  - Descreveu novas **características anatomicas do cérebro antes não documentadas**
- Rene Descartes
  - Descreve a glândula pineal como o centro de controle do corpo e da mente (Dualismo)
  - Descreve como o Líquido cefalorraquidiano circula pelo cerebro

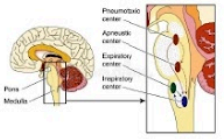


- Jan Swammerdam (1658)
  - Provoca contração do músculo de sapo por estimulação mecânica do nervo.
  - Músculos contraem quando estimulados
- Thomas Willis (1664):
  - Um dos fundadores da Royal Society
  - **Cunha o termo Neurologia**
  - Descreve em detalhes cerebelo, ponte, hemisférios do cerebro etc
  - **Comportamento é resultado de estímulo**

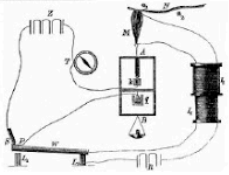
Avanços tecnólogicos e científicos alavancaram descobertas na Neurociência!

- (1791) Luigi Galvani e Lucia Galeazzi Galvani
  - Publicado trabalho sobre a estimulação elétrica dos nervos de sapo.
  - Incentiva Alessandro Volta a pesquisar tais fenômenos
- (1811) Julien Jean Legallois:
  - Descobre centro respiratório na medula
  - Primeira vez mapeado uma área do cérebro a uma função

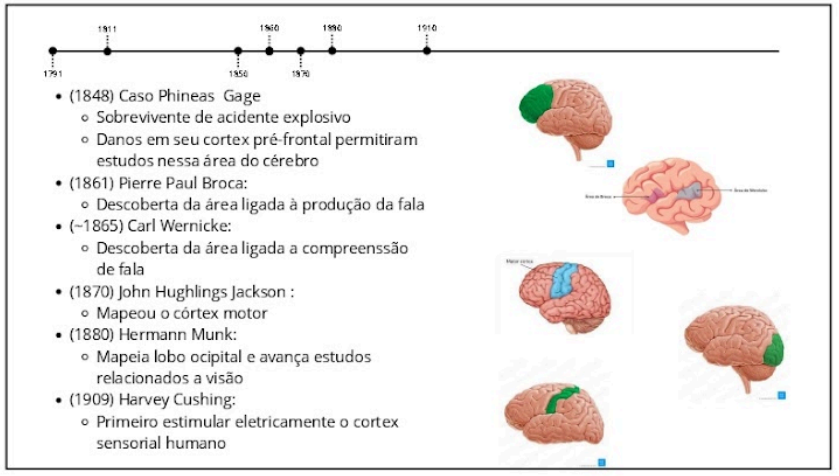




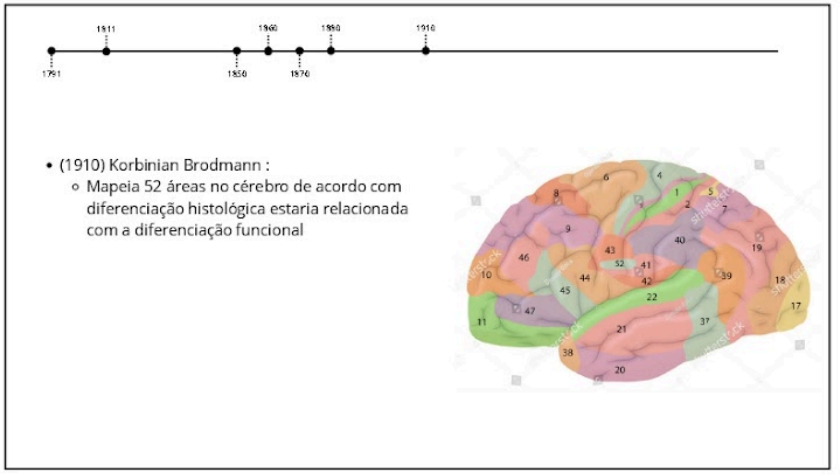
- (1850) Emil Du Bois-Reymond
  - Inventa galvanômetro nervoso
  - Comprovou a natureza elétrica dos sinais nervosos
- (1849) Hermann von Helmholtz:
  - Mede a velocidade de um impulso nervoso
  - 24.6 - 38.4 m/s



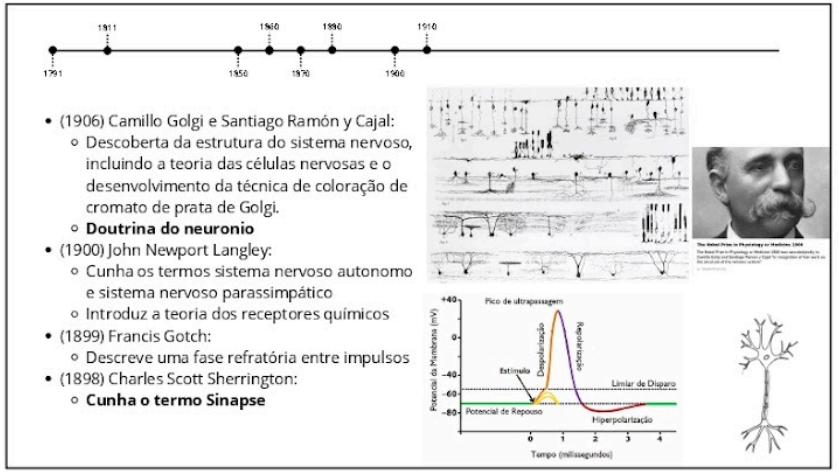
- (1848) Caso Phineas Gage
  - Sobrevivente de acidente explosivo
  - Danos em seu cortex pré-frontal permitiram estudos nessa área do cérebro



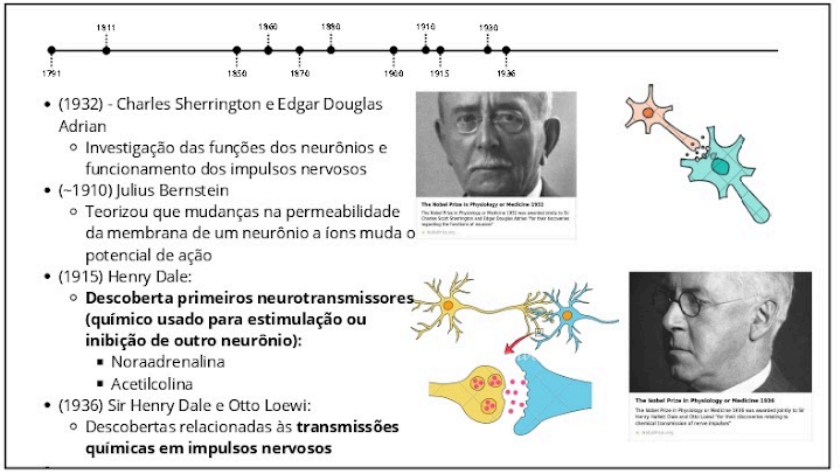
- (1848) Caso Phineas Gage
  - Sobrevivente de acidente explosivo
  - Danos em seu cortex pré-frontal permitiram estudos nessa área do cérebro
- (1861) Pierre Paul Broca:
  - Descoberta da área ligada à produção da fala
- (~1865) Carl Wernicke:
  - Descoberta da área ligada a compreenssão de fala
- (1870) John Hughlings Jackson :
  - Mapeou o córtex motor
- (1880) Hermann Munk:
  - Mapeia lobo ocipital e avança estudos relacionados a visão
- (1909) Harvey Cushing:
  - Primeiro estimular eletricamente o cortex sensorial humano



- (1910) Korbinian Brodmann :
  - Mapeia 52 áreas no cérebro de acordo com diferenciação histológica estaria relacionada com a diferenciação funcional



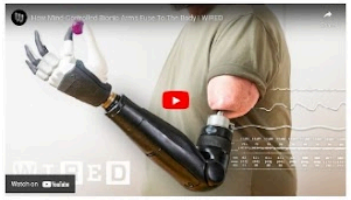
- (1906) Camillo Golgi e Santiago Ramón y Cajal:
  - Descoberta da estrutura do sistema nervoso, incluindo a teoria das células nervosas e o desenvolvimento da técnica de coloração de cromato de prata de Golgi.
  - **Doutrina do neuronio**
- (1900) John Newport Langley:
  - Cunha os termos sistema nervoso autonomo e sistema nervoso parassimpático
  - Introduz a teoria dos receptores químicos
- (1899) Francis Gotch:
  - Descreve uma fase refratória entre impulsos
- (1898) Charles Scott Sherrington:
  - **Cunha o termo Sinapse**

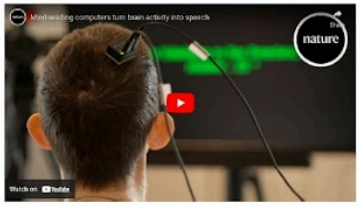


- (1932) - Charles Sherrington e Edgar Douglas Adrian
  - Investigação das funções dos neurônios e funcionamento dos impulsos nervosos
- (~1910) Julius Bernstein
  - Teorizou que mudanças na permeabilidade da membrana de um neurônio a íons muda o potencial de ação
- (1915) Henry Dale:
  - **Descoberta primeiros neurotransmissores (químico usado para estimulação ou inibição de outro neurônio):**
    - Noraadrenalina
    - Acetilcolina
- (1936) Sir Henry Dale e Otto Loewi:
  - Descobertas relacionadas às **transmissões químicas em impulsos nervosos**

Pra quê

Neuropróteses





Pra quê

Computação Neuromórfica

20 W

350 W

Pra quê

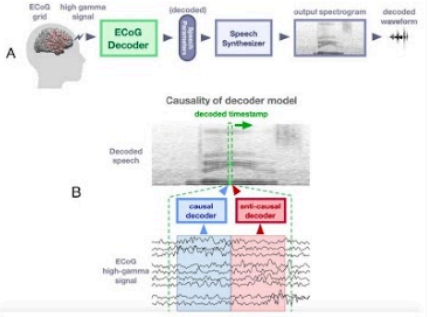
Inteligência Artificial

Melhor entendimento do cérebro e novas propostas de modelos

Pra quê

Neural decoding

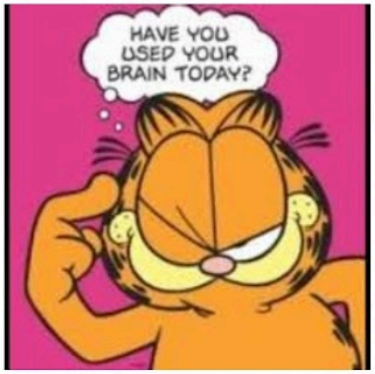
Ler mentes ...



Pra quê

Estudar o cérebro é desafiar a fronteira do conhecimento e enfrentar o maior desafio epistemiológico

Obrigado



Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 25 de set. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

A planejamento da entrega desse GATE foi fundamentado em três tópicos:

- Investigar a plataforma BrainScore
- Explorar trabalhos relacionados
- Consolidar fundamentos teóricos em relação a neurociência;

Para entender melhor a plataforma, assisti a apresentação do autor do artigo descrevendo as motivações da plataforma exisitir:
https://www.youtube.com/watch?v=-Hau9_8r2Ew&t=438s

Para investigar os trabalhos relacionados e discussões a respeito da plataforma, assisti a conferência que, justamente, apresentava os 10 melhores modelos de visão desse ano como também promoveu um debate sobre a legibilidade da plataforma:
https://www.youtube.com/watch?v=fYoW8TxUAco&t=2875s

Consegui testar os básicos do BrainScore, houve uma dificuldade na interpretação dos dados e uso dos modelos:
https://colab.research.google.com/drive/1YH6TT6sjbnUP4UK-0IgKV3JGx1JVEfcB?usp=sharing
https://www.brain-score.org/competition/

Avancei em relação a minha fundamentação teórica assistindo aulas:

- Até a parte de neurotransmissores:
  https://www.youtube.com/@bingsbrain/videos

Os estudos estão registrados em meu caderninho vermelho, que pode ser digitalizado caso seja requisitado.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

A exploração dos trabalhos revelou-se insatisfatória, em parte devido à incerteza da investigação de um tema tão nichado, fundamentado predominantemente em ciência de base ao invés de aplicada, além da escassez de materiais disponíveis.

A fim de consolidar o estudo da temática atual, proponho avançar mais seriamente montando um compêndio de trabalhos correlatos a fim de investigar o “ecossistema” de estudos nessa área, seguido por uma seleção através de palavras chaves do tópico sendo estudado, e por fim realizando uma análise simples e qualitativa de trabalhos que serão considerados base.

Será preciso investigar um pouco melhor a plataforma do BrainScore para resolver pendências de conhecimento mencionadas.

Será necessário continuar os estudos de fundamentos da Neurociência, com o propósito de interpretar melhor os estudos da área.

Observação: [caso precise fazer alguma observação, de qualquer “natureza”]

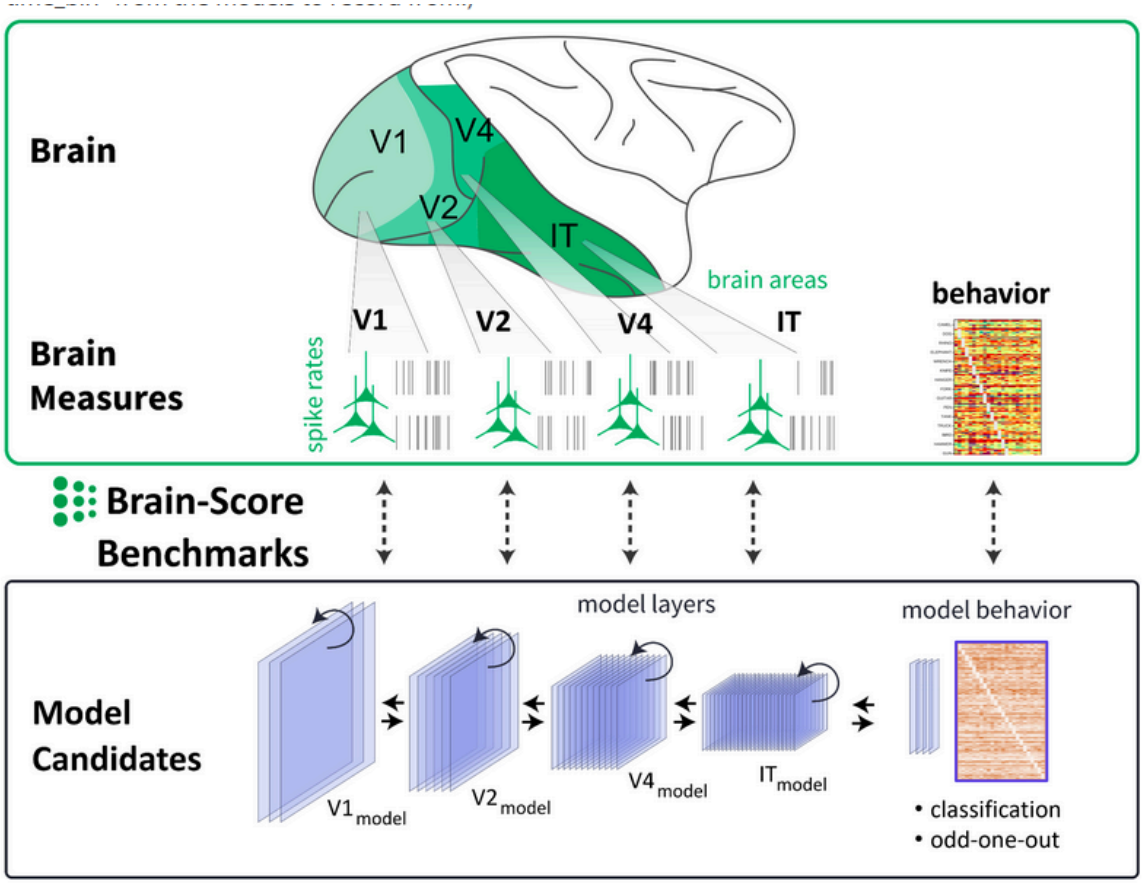
Observando as principais temáticas relacionadas a neurociência e IA em conferências, percebi que as principais vias estão relacionadas a diagnósticos, interação cérebro-máquina, e Neurociência cognitiva baseada em IA.

ACEITE DA ENTREGA:

LEONARDO ALVES: Go!

[test_brainscore.ipynb]

```
%%capture
!git clone https://github.com/brain-score/vision.git
```



Data

```
import brainscore_vision
neural_data = brainscore_vision.load_dataset("MajajHong2015.public")
neural_data = neural_data.transpose('presentation', 'neuroid', 'time_bin')

neural_data
```

```
<xarray.NeuronRecordingAssembly 'dicarlo.MajajHong2015.public' (

presentation: 148480,
                                                        neuroid:
256,
```

```
                                                                  time_bin:
1)>
dask.array<transpose, shape=(148480, 256, 1), dtype=float32,
chunksize=(148480, 256, 1), chunktype=numpy.ndarray>
Coordinates:
  * neuroid          (neuroid) MultiIndex
  - neuroid_id       (neuroid) object 'Chabo_L_M_5_9' ... 'Tito_L_M_8_0'
  - arr              (neuroid) object 'M' 'M' 'M' 'M' 'M' ... 'M' 'M' 'M'
'M'
  - col              (neuroid) int64 9 9 8 9 8 8 7 7 5 6 ... 1 0 1 0 1 0 0
1 1 0
  - hemisphere       (neuroid) object 'L' 'L' 'L' 'L' 'L' ... 'L' 'L' 'L'
'L'
  - subregion        (neuroid) object 'cIT' 'cIT' 'cIT' ... 'pIT' 'pIT'
'pIT'
  - animal           (neuroid) object 'Chabo' 'Chabo' 'Chabo' ... 'Tito'
'Tito'
  - y                (neuroid) float64 0.2 0.6 0.2 1.0 0.6 ... 1.0 1.0 1.8
1.4
  - x                (neuroid) float64 1.8 1.8 1.4 1.8 ... -1.8 -1.4 -1.4
-1.8
  - region           (neuroid) object 'IT' 'IT' 'IT' 'IT' ... 'IT' 'IT'
'IT'
  - row              (neuroid) int64 5 6 5 7 6 7 9 7 9 8 ... 4 4 5 5 6 6 7
7 9 8
  * presentation     (presentation) MultiIndex
  - image_id         (presentation) object
'8a72e2bfdb8c267b57232bf96f069374d...
  - repetition       (presentation) int64 0 18 18 18 18 18 ... 16 16 16 17
17 17
  - stimulus         (presentation) int64 0 426 427 428 429 ... 2569 2566 0
1 2
  - stimulus_id      (presentation) object
'8a72e2bfdb8c267b57232bf96f069374d...
  - id               (presentation) int64 1 418 419 420 421 ... 3197 641
642 643
  - background_id    (presentation) object
'ecd40f3f6d7a4d6d88134d648884e0b9b...
  - s                (presentation) float64 1.0 1.0 1.0 ... 1.246 1.296
0.9114
  - image_file_name  (presentation) object
'astra_rx+00.000_ry+00.000_rz+00.0...
  - filename         (presentation) object
'astra_rx+00.000_ry+00.000_rz+00.0...
  - rxy              (presentation) float64 -0.0 -0.0 -0.0 ... 0.02724
-10.4
```

```
  - tz              (presentation) float64 0.0 0.0 0.0 ... -0.269 -0.599
0.211
  - category_name   (presentation) object 'Cars' 'Faces' ... 'Fruits'
'Fruits'
  - rxz_semantic    (presentation) float64 0.0 0.0 0.0 ... 13.22 -2.621
-14.72
  - ty              (presentation) float64 0.0 0.0 0.0 ... -0.191 -0.213
0.277
  - ryz             (presentation) float64 -0.0 -0.0 -0.0 ... -16.89
-0.2055
  - object_name     (presentation) object 'car_astra' 'face0' ... 'apple'
  - variation       (presentation) int64 0 0 0 0 0 0 0 0 0 ... 3 3 3 3 3 3
3 3
  - size            (presentation) float64 256.0 256.0 256.0 ... 256.0
256.0
  - rxy_semantic    (presentation) float64 90.0 -0.0 -0.0 ... 0.02724
-10.4
  - ryz_semantic    (presentation) float64 -0.0 -0.0 -0.0 ... -16.89
-0.2055
  - rxz             (presentation) float64 0.0 0.0 0.0 ... 13.22 -2.621
-14.72
  * time_bin        (time_bin) MultiIndex
  - time_bin_start  (time_bin) int64 70
  - time_bin_end    (time_bin) int64 170
Attributes:
    stimulus_set_identifier:  hvm-public
    stimulus_set:                     id
backgrou...
    identifier:               dicarlo.MajajHong2015.public
```

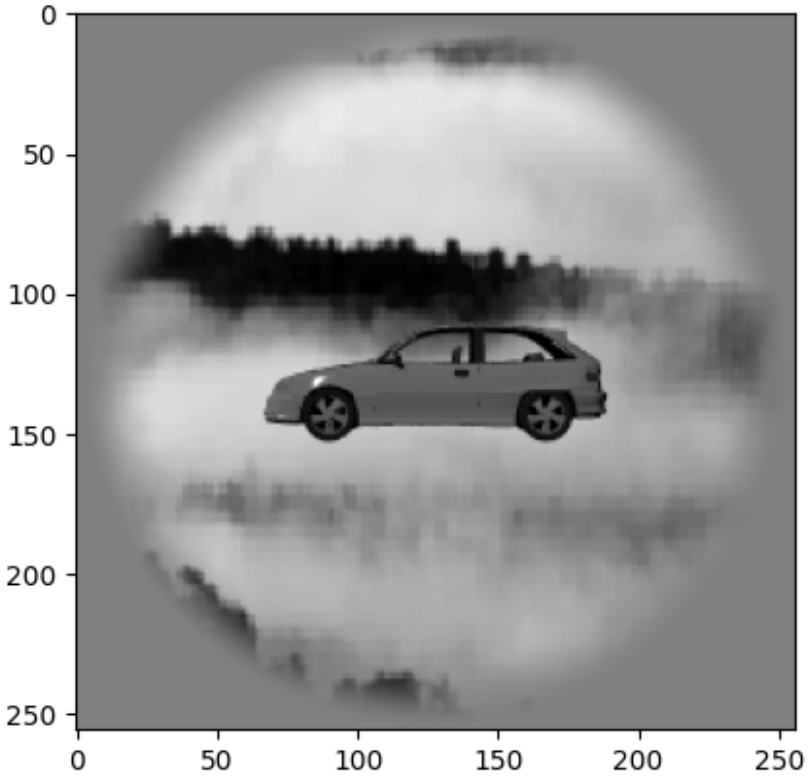
```
stimulus_set = neural_data.attrs['stimulus_set']
stimulus_set
```

```
        id                              background_id         s  \
0        1  ecd40f3f6d7a4d6d88134d648884e0b9b364efc9  1.000000
1        2  006d66c207c6417574f62f0560c6b2b40a9ec5a1  1.000000
2        3  3b3c1d65865028d0fad0b0bf8f305098db717e7f  1.000000
3        4  687ade2f9ee4d52af9705865395471a24ba38d5f  1.000000
4        5  724e5703cc42aa2c3ff135e3508038a90e4ebcb3  1.000000
...    ...                                       ...       ...
3195  3196  70222407751181c4b7723cb09dbda63e7f9c7333  1.286154
3196  3197  f144e3aabccefb4228b8257506a2aa26ba5b4a6d  1.234000
3197  3198  9b87c8be1440ce9626bcfe656700ff3ac8f909fe  0.895714
3198  3199  c3edce2a8fce8c088605bd27fe1f2e7958939f54  1.187143
3199  3200  91e68c65ff92e5a77f0fb13693bfe01bc429da18  0.795714

                                    image_id  \
```

```
0     8a72e2bfdb8c267b57232bf96f069374d5b21832
1     27f69468c9d6019ed0d22b9583c94c5b58198c1c
2     6af1cbb28aacea6c582faa07e92d8325fa7a29d7
3     d0f7a45b377d4920c3466ec7a20dce9437a150d6
4     d4c3b4d4aefd29fd168a2c3c9a9962d99653a715
...                                        ...
3195  caf392d996023d49d9be520da255f865a014f646
3196  4fefbea44756719f5cded1539b1c092e352beab0
3197  b1dbd1fc08edf4dc0814c5578e2c5544ea4d255d
3198  7c5174365cf0ed5a3319250d5913bcd8217f1454
3199  3ade751b115c7813365f97174e75106af2b88310

                                      image_file_name  \
0     astra_rx+00.000_ry+00.000_rz+00.000_tx+00.000_...
1     _12_rx+00.000_ry+00.000_rz+00.000_tx+00.000_ty...
2     face0003_rx+00.000_ry+00.000_rz+00.000_tx+00.0...
3     walnut_obj_rx-90.000_ry+00.000_rz+00.000_tx+00...
4     walnut_obj_rx-90.000_ry+00.000_rz+00.000_tx+00...
...                                                 ...
3195  _19_flyingBoat_rx+27.024_ry+104.520_rz+100.874...
3196  Beetle_rx-06.302_ry-39.833_rz+22.244_tx-00.197...
3197  _001_rx-13.140_ry-26.381_rz+09.423_tx+00.218_t...
3198  _004_rx+11.926_ry-44.019_rz+38.652_tx-00.103_t...
3199  _004_rx-36.603_ry+10.321_rz+16.609_tx+00.024_t...

                                             filename        rxy     tz
\
0     astra_rx+00.000_ry+00.000_rz+00.000_tx+00.000_...  -0.000000  0.000
1     _12_rx+00.000_ry+00.000_rz+00.000_tx+00.000_ty...  -0.000000  0.000
2     face0003_rx+00.000_ry+00.000_rz+00.000_tx+00.0...  -0.000000  0.000
3     walnut_obj_rx-90.000_ry+00.000_rz+00.000_tx+00...  -0.000000  0.000
4     walnut_obj_rx-90.000_ry+00.000_rz+00.000_tx+00...  -0.000000  0.000
...                                                 ...        ...    ...
3195  _19_flyingBoat_rx+27.024_ry+104.520_rz+100.874...  27.783925  0.280
3196  Beetle_rx-06.302_ry-39.833_rz+22.244_tx-00.197... -39.833000 -0.042
3197  _001_rx-13.140_ry-26.381_rz+09.423_tx+00.218_t... -26.381000  0.000
3198  _004_rx+11.926_ry-44.019_rz+38.652_tx-00.103_t... -44.019000  0.521
3199  _004_rx-36.603_ry+10.321_rz+16.609_tx+00.024_t...  10.321000  0.249

     category_name  rxz_semantic     ty        ryz object_name  variation
\
0             Cars      0.000000  0.000  -0.000000   car_astra          0
1           Tables      0.000000  0.000  -0.000000      table3          0
2            Faces      0.000000  0.000  -0.000000       face2          0
3           Fruits      0.000000  0.000  -0.000000      walnut          0
4           Fruits      0.000000  0.000  -0.000000      walnut          0
```

```
...           ...           ...    ...        ...          ...        ...
3195        Planes    -12.905496  0.096  17.585453   airplane2          3
3196          Cars     -6.302000 -0.197  22.244000  car_beetle          3
3197        Chairs    -13.140000  0.218   9.423000      chair0          3
3198        Chairs     11.926000 -0.103  38.652000      chair1          3
3199        Chairs    -36.603000  0.024  16.609000      chair1          3

       size  rxy_semantic  ryz_semantic        rxz  \
0     256.0     90.000000     -0.000000   0.000000
1     256.0     -0.000000     -0.000000   0.000000
2     256.0     -0.000000     -0.000000   0.000000
3     256.0     -0.000000     -0.000000   0.000000
4     256.0     -0.000000     -0.000000   0.000000
...     ...           ...           ...        ...
3195  256.0    117.783925     17.585453 -12.905496
3196  256.0     50.167000     22.244000  -6.302000
3197  256.0    -26.381000      9.423000 -13.140000
3198  256.0    -44.019000     38.652000  11.926000
3199  256.0     10.321000     16.609000 -36.603000

                                  stimulus_id
0     8a72e2bfdb8c267b57232bf96f069374d5b21832
1     27f69468c9d6019ed0d22b9583c94c5b58198c1c
2     6af1cbb28aacea6c582faa07e92d8325fa7a29d7
3     d0f7a45b377d4920c3466ec7a20dce9437a150d6
4     d4c3b4d4aefd29fd168a2c3c9a9962d99653a715
...                                       ...
3195  caf392d996023d49d9be520da255f865a014f646
3196  4fefbea44756719f5cded1539b1c092e352beab0
3197  b1dbd1fc08edf4dc0814c5578e2c5544ea4d255d
3198  7c5174365cf0ed5a3319250d5913bcd8217f1454
3199  3ade751b115c7813365f97174e75106af2b88310

[3200 rows x 19 columns]
```

```
%matplotlib inline
from matplotlib import pyplot, image

i=0
id = neural_data['stimulus_id'].values[i]
local_path = stimulus_set.get_stimulus(id)

img = image.imread(local_path)
pyplot.imshow(img)
pyplot.show()
```




```
neural_data[0]
```

```
<xarray.NeuronRecordingAssembly 'dicarlo.MajajHong2015.public' (neuroid:
256,
                                                               time_bin:
1)>
dask.array<getitem, shape=(256, 1), dtype=float32, chunksize=(256, 1),
chunktype=numpy.ndarray>
Coordinates:
  * neuroid         (neuroid) MultiIndex
  - neuroid_id      (neuroid) object 'Chabo_L_M_5_9' ... 'Tito_L_M_8_0'
  - arr             (neuroid) object 'M' 'M' 'M' 'M' 'M' ... 'M' 'M' 'M'
'M' 'M'
  - col             (neuroid) int64 9 9 8 9 8 8 7 7 5 6 ... 1 0 1 0 1 0 0 1
1 0
  - hemisphere      (neuroid) object 'L' 'L' 'L' 'L' 'L' ... 'L' 'L' 'L'
'L' 'L'
  - subregion       (neuroid) object 'cIT' 'cIT' 'cIT' ... 'pIT' 'pIT'
'pIT'
  - animal          (neuroid) object 'Chabo' 'Chabo' 'Chabo' ... 'Tito'
'Tito'
  - y               (neuroid) float64 0.2 0.6 0.2 1.0 0.6 ... 1.0 1.0 1.8
```

```
1.4
  - x               (neuroid) float64 1.8 1.8 1.4 1.8 ... -1.8 -1.4 -1.4
-1.8
  - region          (neuroid) object 'IT' 'IT' 'IT' 'IT' ... 'IT' 'IT' 'IT'
'IT'
  - row             (neuroid) int64 5 6 5 7 6 7 9 7 9 8 ... 4 4 5 5 6 6 7 7
9 8
    presentation    object ('8a72e2bfdb8c267b57232bf96f069374d5b21832', 0,
0,...
  * time_bin        (time_bin) MultiIndex
  - time_bin_start  (time_bin) int64 70
  - time_bin_end    (time_bin) int64 170
Attributes:
    stimulus_set_identifier:  hvm-public
    stimulus_set:                     id
backgrou...
    identifier:               dicarlo.MajajHong2015.public
```

Benchmark Data

```
from brainscore_vision.benchmark_helpers.neural_common import
average_repetition
benchmark_data = neural_data.sel(region='IT')

benchmark_data = benchmark_data.squeeze('time_bin')  # (3)
print("shape", benchmark_data.shape)

shape (3200, 168)
```

Behavioral Data

```
behavioral_data = brainscore_vision.load_dataset('Rajalingham2018.public')

brainio-brainscore/assy_dicarlo_Rajalingham2018_public.nc: 100%|██████████|
253M/253M [00:20<00:00, 12.1MB/s]
brainio-brainscore/stimulus_objectome_public.csv: 100%|██████████|
270k/270k [00:01<00:00, 203kB/s]
brainio-brainscore/stimulus_objectome_public.zip: 100%|██████████|
79.6M/79.6M [00:10<00:00, 7.36MB/s]
```

Eval

Pre-defined metrics

```
from brainscore_vision import load_metric

metric = load_metric('ridge')
```

```
rdm_metric = load_metric('rdm')

rdm_cv = load_metric('rdm_cv')

import numpy as np
from numpy.random import RandomState

from brainio.assemblies import NeuroidAssembly

rnd = RandomState(0)  # seed for reproducibility
assembly = NeuroidAssembly((np.arange(30 * 25) + rnd.standard_normal(30 *
25)).reshape((30, 25)),
                           coords={'stimulus_id': ('presentation',
np.arange(30)),
                                   'object_name': ('presentation', ['a',
'b', 'c'] * 10),
                                   'neuroid_id': ('neuroid',
np.arange(25)),
                                   'region': ('neuroid', ['V1'] * 25)},
                           dims=['presentation', 'neuroid'])

source, target = assembly, assembly  # we're testing how well the metric
can predict the dataset itself
score = metric(source=source, target=target)
print(score)

cross-validation: 100%|██████████| 10/10 [00:00<00:00, 22.07it/s]

<xarray.Score ()>
array(0.99999893)
Attributes:
    raw:      <xarray.Score (split: 10, neuroid: 25)>\narray([[0.99999998,
0....
    error:    <xarray.Score ()>\narray(7.31630413e-07)
```

Model

```
%%capture
!apt update
!apt install libgl1-mesa-glx -y
!apt install libglib2.0-0 -y

import torch
torch.cuda.empty_cache()
```

```
from brainscore_vision import score

similarity_score = score(model_identifier='alexnet',
benchmark_identifier='MajajHong2015public.IT-pls')
```

```
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager, possibly
rendering your system unusable.It is recommended to use a virtual
environment instead: https://pip.pypa.io/warnings/venv. Use the
--root-user-action option if you know what you are doing and want to
suppress this warning.
```

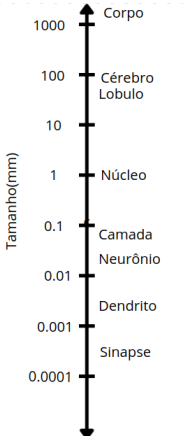
{"model_id":"27ddadc0ee914cdaa29979e9afe4256a","version_major":2,"version_minor":0}

[Anotações de estudos: Introdução à Neurociência/Neurofisiologia]

Introdução à Neurociência

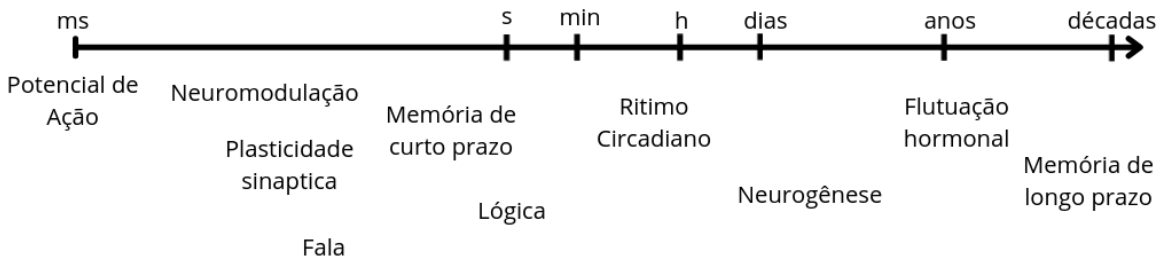
Introdução

- O cérebro humano representa cerca de 2% da massa corporal, porém consome aproximadamente 20% da energia metabólica total.
- As rugas visíveis na superfície cerebral correspondem ao córtex, uma camada bidimensional dobrada, com espessura média de cerca de 2 mm.

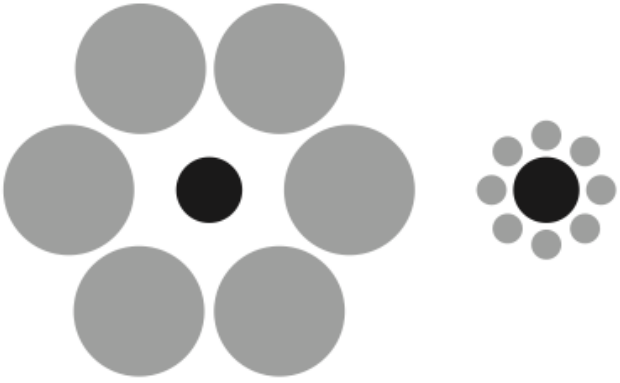


- A organização do sistema nervoso ocorre em **múltiplas escalas espaciais,** desde sinapses com complexidade comparável a pequenas cidades, até a intrincada geometria dos neurônios, com dendritos e axônios extensamente ramificados.
    - Curiosidade: O Gânglio da raiz dorsal pode atingir a altura do próprio organismo.

- A Neurociência de Sistemas investiga como redes neurais coordenam-se para gerar funções superiores, como memória e pensamento.
- Além disso, há dinâmicas em **múltiplas escalas de tempo**



- Neurociência Computacional: modelagem computacional para descrever tais fenômenos.
    - Uma premissa fundamental: “Enxergamos com o cérebro, não com os olhos.”



A vida elétrica do neurônio

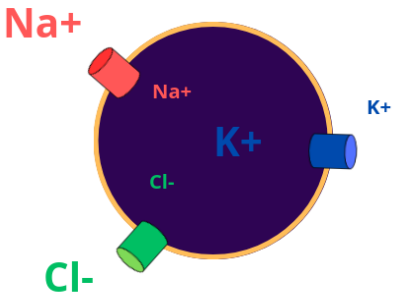
- A atividade elétrica dos neurônios é a base energética fundamental do cérebro, necessária para armazenar e processar informações sobre o mundo.
- O conceito de “eletricidade animal” destaca que a função neuronal depende de processos eletroquímicos.

Doutrina do Neurônio

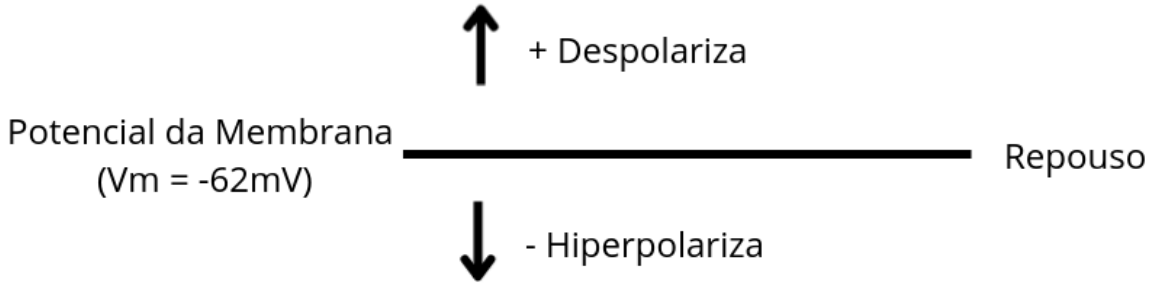
- Os neurônios não formam uma rede contínua; ao contrário, comunicam-se em pontos específicos chamados sinapses.
- A Doutrina do Neurônio, atribuída a pesquisadores como Fridtjof Nansen (primeiro a reportar tal conhecimento), e posteriormente por Camillo Golgi e Santiago Ramón y Cajal (ganhadores do Nobel), estabelece que o **neurônio é a unidade estrutural e funcional do sistema nervoso**.

Potencial Elétrico

- A composição iônica do meio intracelular reflete, em parte, a origem marinha da vida (sopa da vida)
- O potencial de repouso típico de um neurônio está entre -60 e -80 mV, cerca de 1/10 de um volt.
- A permeabilidade ao K+ em repouso é elevada devido a canais iônicos especializados.

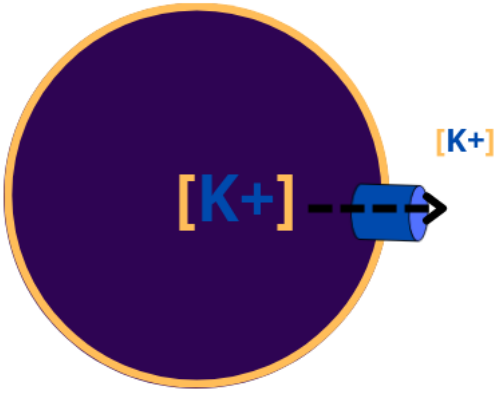


Termos:



Concentração vs Força elétrica

- Existe uma tendência de atingir o equilíbrio homeostático, fazendo com que haja um fluxo de K+ para fora da célula
- A saída de K+ hiperpolariza a membrana
- Fluxo interrompe por volta de -62mV (Potencial de Nernst), onde as forças elétricas e químicas se igualam.

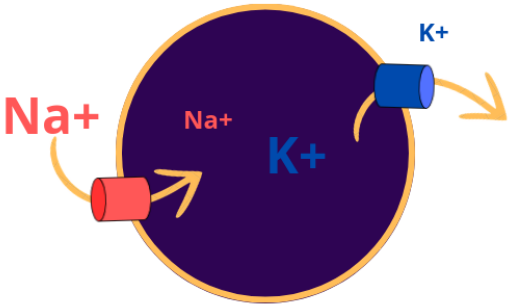


Equação de Nernst

$$E_{ion} = \frac{RT}{zF}\, ln(\frac{[ion_{interno}]}{[ion_{externo}]})$$

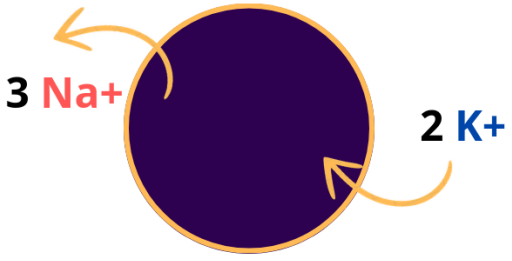
- E_ion é o potencial de equilíbrio do íon,
- R é a constante universal dos gases,
- T é a temperatura absoluta (em Kelvin),
- z é a valência (carga) do íon,
- F é a constante de Faraday,

- Potencial Inerte Na+ = 60mV, K+=-80mV
- Por que não há equilíbrio?
  - Bomba de sódio e potássio



Bomba de Sódio-Potássio e Gradientes Iônicos

- A concentração iônica não se equilibra espontaneamente; a bomba de sódio e potássio (Na+/K+ ATPase) transporta íons contra seus gradientes eletroquímicos, consumindo energia



ATP→ADP+P

.

- Esse gasto energético elevado justifica o alto consumo metabólico do cérebro.
- Caso não haja canais o suficiente, seria mais difícil entrar em repouso, logo os neurônios estariam mais excitáveis.

Equação de Goldman-Hodgkin-Katz (generalização de Nernst)

$$E_m = \frac{RT}{F} \ln \left(\frac{\sum_i^n P_{M_i^+} [M_i^+]_{\text{out}} + \sum_j^m P_{A_j^-} [A_j^-]_{\text{in}}}{\sum_i^n P_{M_i^+} [M_i^+]_{\text{in}} + \sum_j^m P_{A_j^-} [A_j^-]_{\text{out}}} \right)$$

Lei de Ohm

$$I_k = (V_k - E_k) - gK$$

- Ik corrente elétrica (fluxo de ions)
- gK condutividade (proporcional a número de canais)
- Ek Potencial inerte do ion

Equação de cabo

$$V_m = \lambda^2 \frac{d^2 V_m}{dx^2} - \tau_m \frac{dV_m}{dt}$$

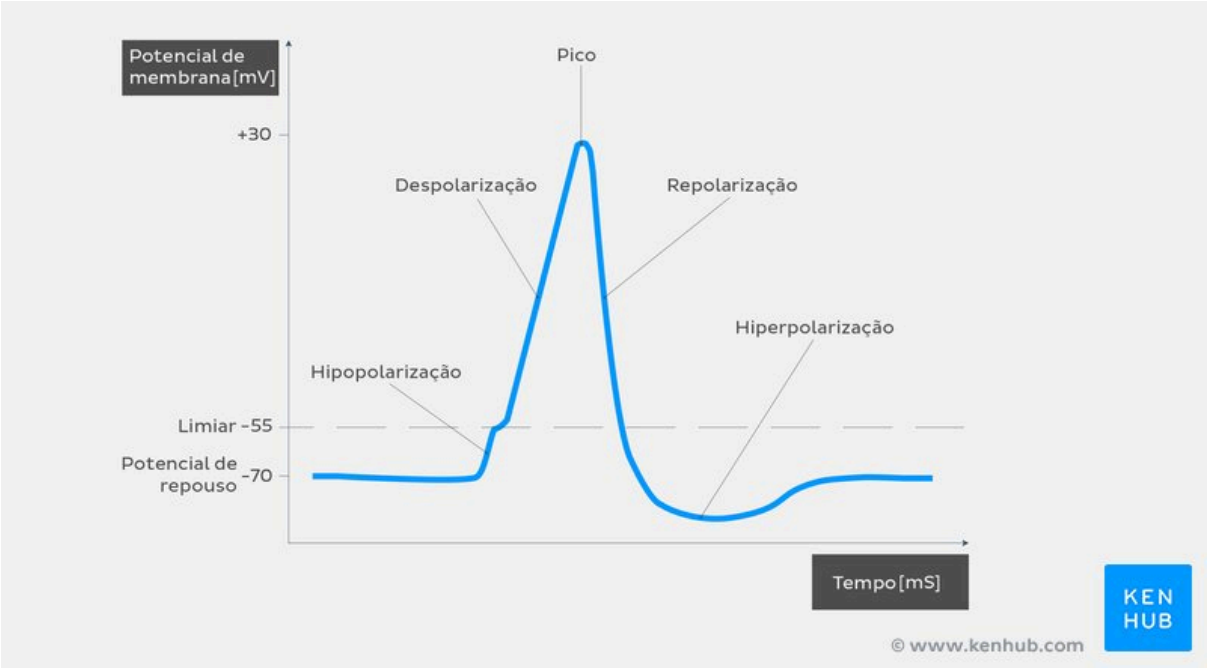
  - Vm: potencial da membana
  - λ: coeficiente de difusão elétrica,
  - τ: constante de tempo da membrana,

- O uso do cabo permite um alongamento que facilita o fluxo de informações
- Um cabo mais grosso seria mais dificil de manter
- A bainha de mielina aumenta o coeficiente de difusão, permitindo maior propagação do sinal para uma mesma distância

Potencial de Ação

- O potencial de ação é uma onda de despolarização auto-reforçada que percorre o neurônio.
- Em repouso, o potencial de membrana aproxima-se do equilíbrio para K+ (Ek).
- Canais iônicos dependentes de voltagem possuem comportamentos complexos, incluindo mecanismos de inativação temporizada.
- O potencial de ação decorre de mudanças na condutância iônica, não de alterações significativas nas concentrações iônicas totais. Essas concentrações são mantidas

pela bomba de Na+/K+.

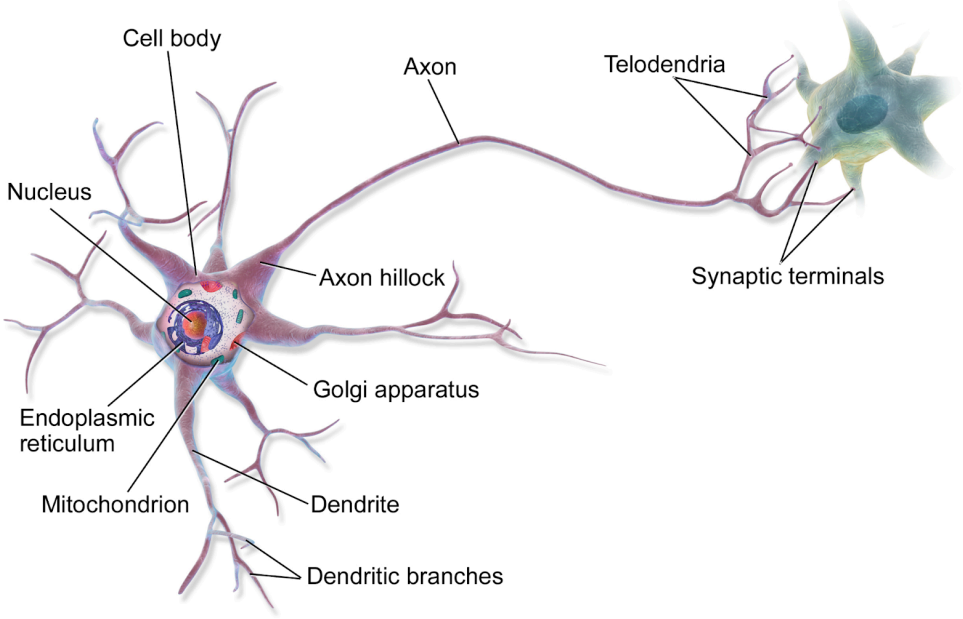


Fases do Potencial de Ação:

1. **Estado de repouso**:
   - Potencial de membrana estável (~ -70 mV).
   - Canais de Na+ e K+ fechados.
2. **Despolarização**:
   - Estímulo atinge o limiar (~ -55 mV).
   - Canais de Na+ se abrem, permitindo entrada rápida de Na+.
   - Potencial se torna menos negativo (e pode ultrapassar 0 mV).
3. **Repolarização**:
   - Canais de Na+ se inativam.
   - Canais de K+ se abrem, permitindo a saída de K+.
   - Potencial de membrana retorna a valores negativos.
4. **Hiperpolarização**:
   - Saída excessiva de K+ devido ao fechamento lento dos canais.
   - Potencial de membrana se torna mais negativo que o estado de repouso.
5. **Restabelecimento do repouso**:
   - Bomba Na+K+-ATP, se restaura gradientes iônicos.
   - Membrana retorna ao potencial de repouso.

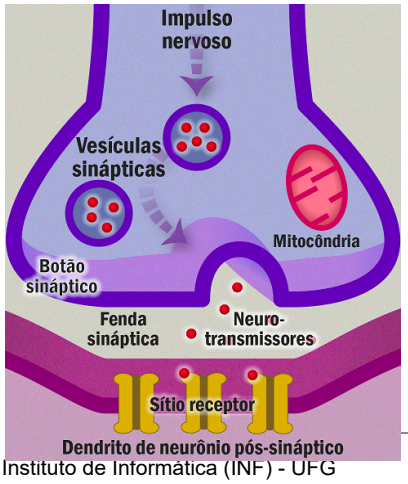
- Certas toxinas, como a do baiacu, bloqueiam canais de Na+ dependentes de voltagem, impedindo a geração do potencial de ação.
- O potencial de ação propaga-se ao longo do axônio, regenerando-se continuamente devido a feedbacks positivos locais.

Neurônio



Colina do axônio → avalia os potenciais pós-sinápticos excitatórios/inibitórios e transmite o sinal (tudo ou nada)

Sinapse



- A sinapse é a região de contato entre neurônios, onde ocorre a transmissão do impulso elétrico do neurônio pré-sináptico ao pós-sináptico por meio de neurotransmissores.
- A fenda sináptica não é vazia; é repleta de proteínas, mitocôndrias e outras estruturas.
- Neurotransmissores liberados do lado pré-sináptico ligam-se a receptores pós-sinápticos, modulando o estado elétrico e químico da célula alvo.

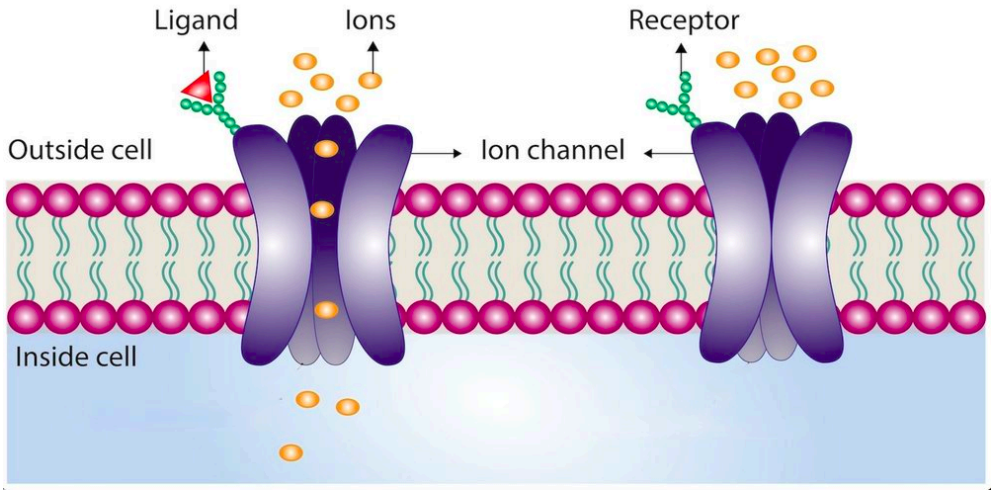
Receptores Pós-Sinápticos

- **Receptores ionotrópicos**: canais iônicos que se abrem diretamente após a ligação do neurotransmissor, alterando imediatamente a permeabilidade iônica.
- **Receptores metabotrópicos**: acoplados a proteínas G (GPCR), desencadeiam cascatas intracelulares complexas que podem regular canais iônicos, modificar a expressão gênica ou alterar a fisiologia celular a longo prazo.

Neurotransmissores de Aminoácidos

- **GABA** é tipicamente inibitório e o **glutamato**, tipicamente excitatório.
- Porém, o efeito depende do tipo de receptor e do contexto; não há natureza intrinsecamente excitatória ou inibitória de um neurotransmissor em si.

Receptor Iônico

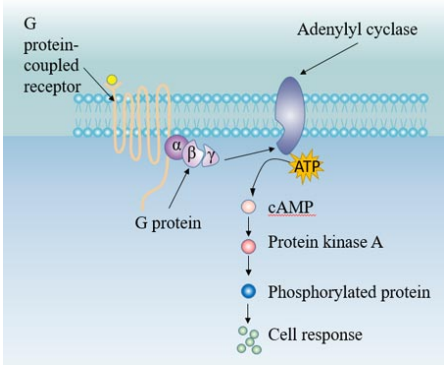


- Controlado pela abertura de um canal dependente de voltagem ou pela ligação de um neurotransmissor no lado extracelular.
- Os canais iônicos controlados por transmissores abrem diretamente um canal iônico.
- O efeito na célula pós-sináptica depende da permeabilidade do canal e da força de condução.
- Exemplos:
    - Receptores de glutamato: geralmente excitatórios.

- Receptor GABA: associado a canais de Cl-, gera hiperpolarização; agonistas incluem barbitúricos e benzodiazepínicos; picrotoxina é um bloqueador.
- Receptor de glicina: principal inibitório na medula espinhal; estricnina o bloqueia.
- Receptor nicotínico de acetilcolina: atua na junção neuromuscular, agonista é a nicotina, curare o bloqueia. A acetilcolinesterase encerra a ação da ACh na fenda sináptica.

Receptor Metabotrópico

- Ativado em dois passos:
  1. Receptor acoplado à proteína G (GCPR).
  2. Acoplamento a um canal iônico.
- Transmissor → GCPR → Canal iônico.
- O efeito na célula pós-sináptica depende da permeabilidade do canal e da força de condução.



Sistemas Efetores Acoplados à Proteína G (cascata)

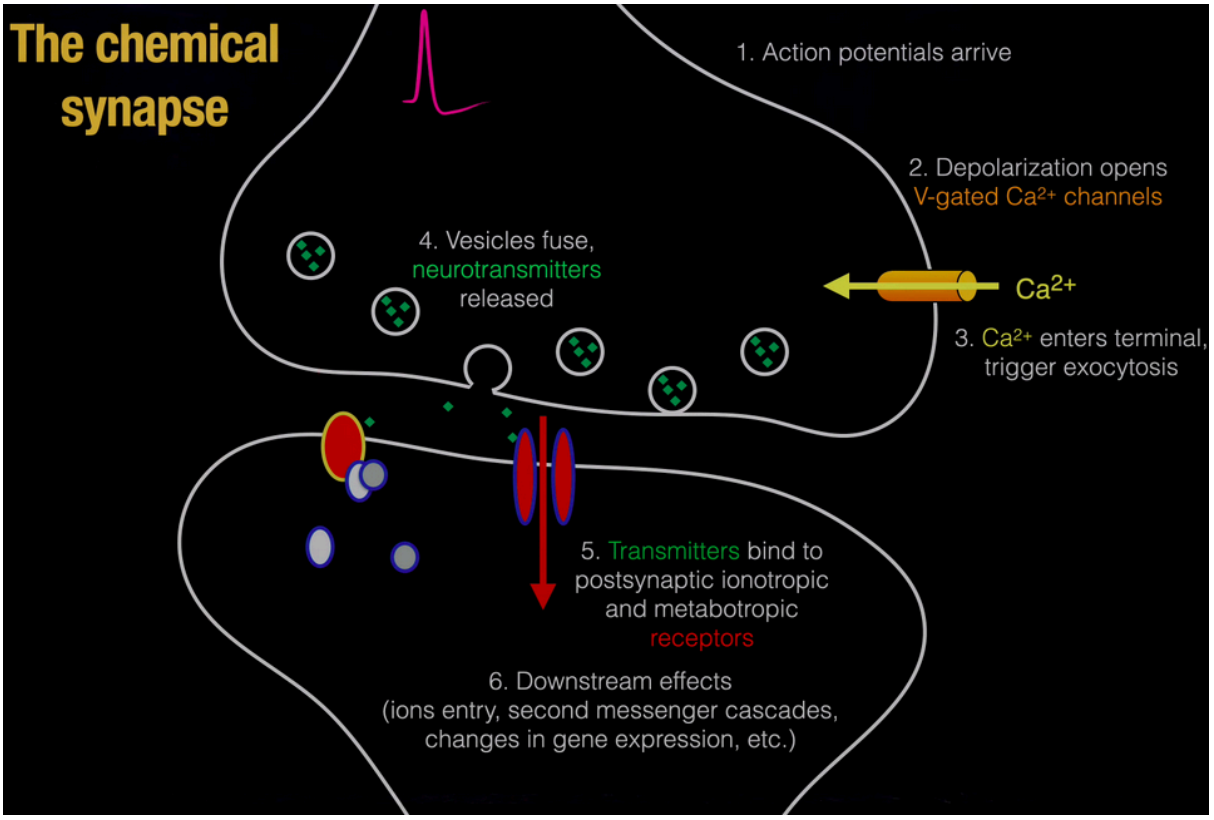
- Transmissor → GCPR → enzimas → cascata bioquímica.
- Os efeitos dependem do contexto e podem causar várias mudanças:
  - Alterar o potencial de membrana.
  - Desencadear mudanças de longo prazo na expressão gênica ou na fisiologia celular.
- Possui diversas funções e tipos, ligando-se a sinais específicos.
- Variedade depende:
  - Tipo de ligante.
  - Tipo de sistema que interage com as proteínas
- Receptores acoplados à proteína G (GPCR) podem reconhecer diversos ligantes e iniciar cascatas intracelulares complexas.
- Exemplo:
  - Adrenalina em receptores β-adrenérgicos aumenta cAMP, ativa PKA e fosforila canais de Ca^{2+}, aumentando a frequência e força das contrações cardíacas.

- Por outro lado, a acetilcolina em receptores muscarínicos do coração abre canais de K+, reduzindo a excitabilidade e diminuindo a frequência cardíaca. A atropina bloqueia esses receptores, elevando novamente a frequência.

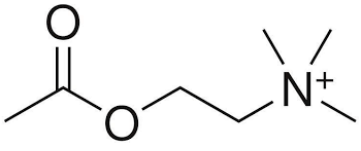
Balanço Entre Inibição e Excitação

- No córtex, cerca de 20% dos neurônios são inibitórios e 80% são excitatórios. Pois, então, por que não há um constante hiperestímulo?.
  - Há uma regulação na própria atividade neural para balancear a importância de cada sinal

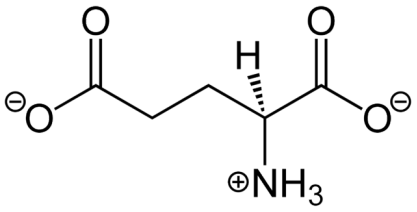
Sinapses Químicas



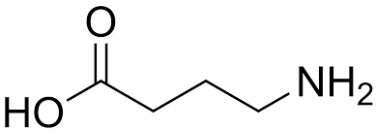
Neurotransmissores comuns



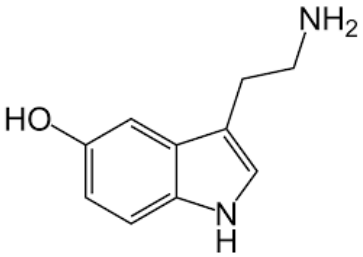
ACETILCOLINA



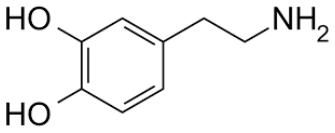
GLUTAMATO



GABA



SEROTONINA



DOPAMINA

Anotações de:
https://www.youtube.com/playlist?list=PLqgZEQsU_8E0I1P9bKR6yKOKPMpoJ_tLR

APÊNDICE 2

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 2 de out. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O planejamento da entrega deste GATE foi estruturado com base em dois tópicos principais:

1. Investigar de maneira mais sistemática a área de estudo.
2. Consolidar os conhecimentos sobre os fundamentos do tema.

Para obter uma visão mais abrangente da área, incluindo os principais autores e os temas mais discutidos, realizei um levantamento um pouco mais sistemático dos materiais, que pode ser consultado aqui: Neurostuff - Screening

Com base em artigos selecionados, consegui destacar as seguintes premissas e teses centrais:

- Premissa
  - Devemos considerar o cérebro como um sistema que pode ser otimizado, de forma análoga a modelos matemáticos, como redes neurais artificiais.
- Tese
  - A utilização de redes neurais artificiais como modelos computacionais de dados comportamentais e neurais pode contribuir para o desenvolvimento de hipóteses falsificáveis sobre o funcionamento cerebral em múltiplos níveis explicativos.

Para consolidar esses conhecimentos fundamentais, investiguei melhor a plataforma BrainScore compreendendo melhor a organização dos dados, conforme descrito neste artigo: https://pubmed.ncbi.nlm.nih.gov/22325196/

Além disso, dei continuidade ao estudo dos conceitos básicos de neurociência, como neurotransmissores. Minhas anotações foram feitas manualmente e digitalizadas para eventual divulgação, que pode ser acessada aqui: Anotações de neurociência

JJ DiCarlo - How the brain solve visual obj recognition

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Com a linha de pesquisa propriamente estruturada, basta dar continuidade a pesquisa de materiais relacionados, investigação de materiais já selecionados, e estudos de fundamentos ainda não dominados.
Proponho dar continuidade aos estudos de fundamentos da área, investigar melhor uma das pesquisas já selecionadas e observar implementações e resultados.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[Anotações Artigo : How Does the Brain Solve Visual Object Recognition? - JJ Dicarlo]

Como o Cérebro Resolve o Reconhecimento Visual de Objetos?

Introdução ao Reconhecimento Visual

- O cérebro identifica objetos em menos de 200 ms, mesmo com variações de posição, escala, iluminação e contexto.
- A habilidade é sustentada por representações neurais no córtex temporal inferior (IT).
- Avanços na compreensão deste processo envolvem neurociência, psicologia e modelagem computacional.

Desafios do Reconhecimento de Objetos

- **Invariância**: Capacidade de reconhecer objetos apesar de variações na aparência.
- A variabilidade das imagens é causada por **transformações de posição, pose, escala e fundo visual**.
- Soluções computacionais precisam "desembaraçar" as informações sensoriais para distinguir objetos.

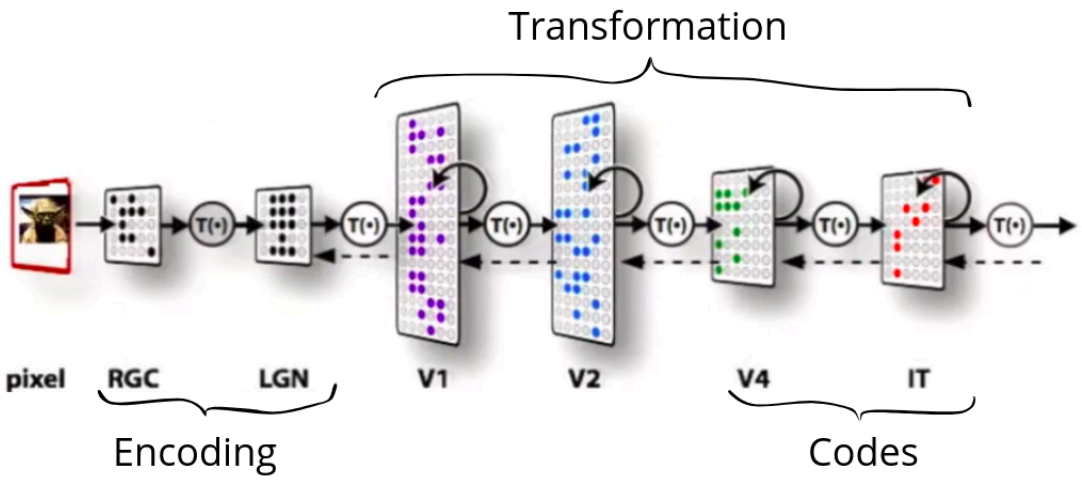
Arquitetura do Sistema Visual

- O **fluxo ventral visual** é responsável pelo reconhecimento de objetos e segue esta hierarquia:
    1. Retina → LGN (núcleo geniculado lateral)
    2. Córtex visual primário (V1) → V2 → V4 → IT.
- Cada área aplica cálculos não-lineares que gradualmente tornam a representação dos objetos mais clara e explícita.

Representação Neuronal

- O IT utiliza **taxas de disparo neuronal** (~50 ms) para codificar objetos.
- Representações populacionais são distribuídas, com vários neurônios contribuindo para descrever objetos.
- As representações mantêm seletividade (reconhecimento de forma) e tolerância (invariância a transformações).

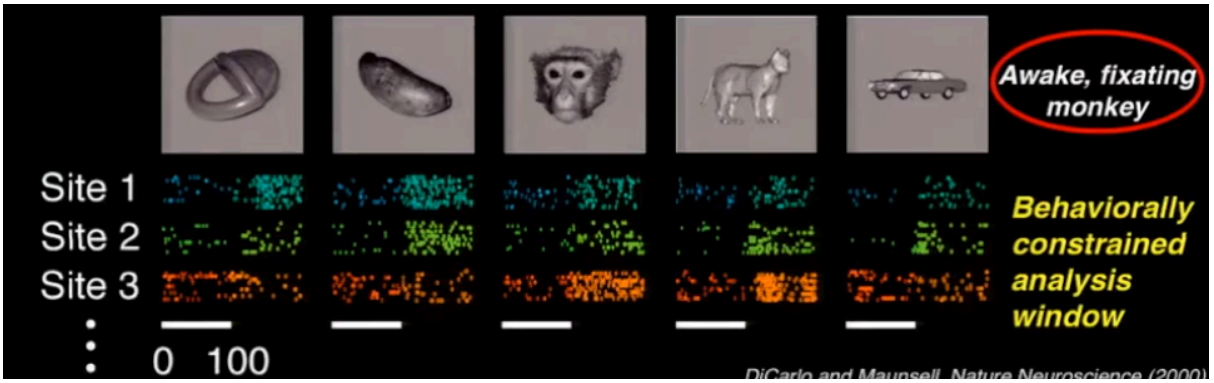
Modelagem Computacional



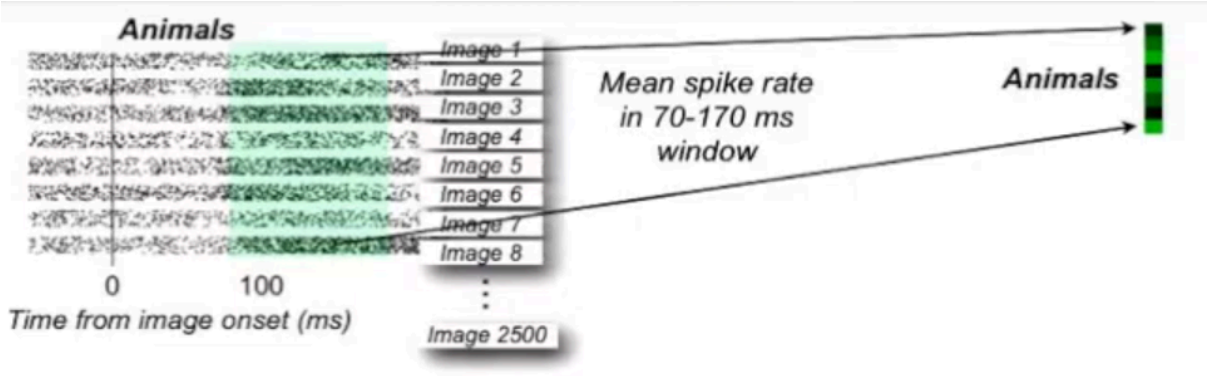
- **Modelos hierárquicos** inspirados no cérebro utilizam operações seletivas e tolerantes para simular o reconhecimento visual.
- Exemplos incluem redes neurais profundas com camadas organizadas para melhorar a discriminação de objetos.
- Embora promissores, esses modelos ainda apresentam limitações em generalização e correspondência com o cérebro.
- Investigar respostas neurais (codes) é uma parte fundamental do estudo da fenomenologia, pois essas respostas estão intimamente ligadas a atividade de detecção de objetos.

Comparação de Atividades Neuronais

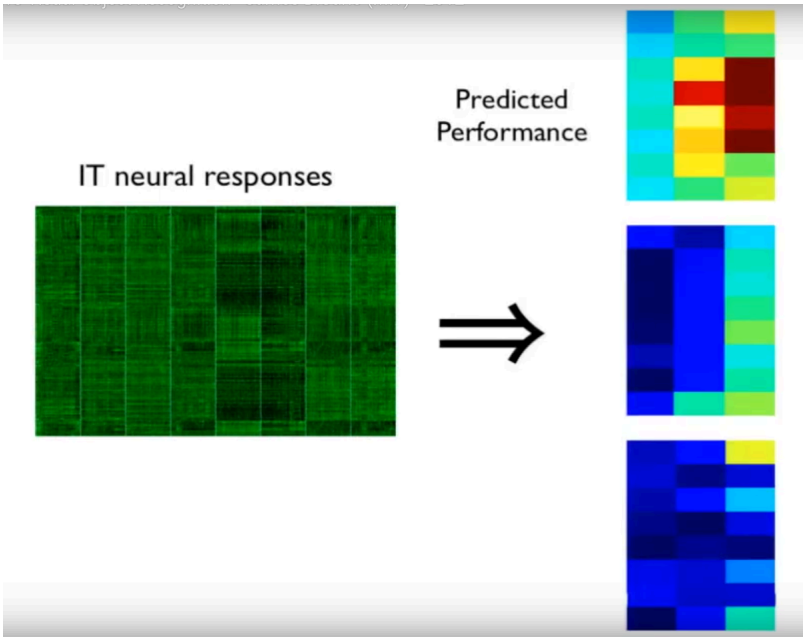
- **Obtenção de dados**
  - Os dados são obtidos a partir de registros de ativações neurais, como fMRI, eletrocorticografia (ECoG) e outras técnicas de neuroimagem e neurofisiologia.



- **Extração de Features**



  - Obter representações claras do comportamento do modelo em resposta aos estímulos apresentados.
  - Utilizar essas representações como base para avaliar a atividade neuronal.
- **Análise Quantitativa**



  - Aplicar decodificadores para transformar as representações em medidas quantitativas, permitindo comparações precisas.

- **Comparação em Espaços de Alta Dimensão**
    - DiCarlo destaca que a análise de respostas neurais em diferentes dias pode revelar padrões consistentes.
    - Mesmo em espaços de alta dimensão, é possível identificar similaridades nas respostas.

Desafios e Direções Futuras

1. **Entendimento dos Algoritmos**:
    - Como as redes neurais no IT "desembaraçam" representações sensoriais complexas.
2. **Conexões de Feedback**:
    - Papel das interações entre áreas corticais na percepção.
3. **Integração Multidisciplinar**:
    - Colaboração entre neurociência, aprendizado de máquina e psicofísica para avançar na compreensão.

Apresentação do artigo:
https://www.youtube.com/watch?v=JxEq7kUnQF8

[Compêndio de artigos relacionados]

Autor	Paper	Citações	Ano publicação	url	**Investigar Mais?**
CF Cadieu	Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition	839	2014	https://arxiv.org/abs/1406.3284	SIMM
Demis Hassabis	Neuroscience-Inspired Artificial Intelligence	1816	2017	https://www.cell.com/neuron/pdf/S0896-6273(17)30509-3.pdf	TALVEZ
DLK Yamins	Using goal-driven deep learning models to understand sensory cortex	1719	2016	https://www.nature.com/articles/nn.4244	SIMM

RPN Rao	Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects	5916	1999	https://www.nature.com/articles/nn0199_79	TALVEZ
A Saxe	If deep learning is the answer, then what is the question?	323	2021	https://www.nature.com/articles/s41583-020-00395-8	SIM
JJ DiCarlo	How does the brain solve visual object recognition?	2005	2012	https://www.cell.com/neuron/fulltext/S0896-6273(12)00092-X	SIMM
A Celeghin	Convolutional neural networks for vision neuroscience: significance, developments, and outstanding issues	14	2023	https://www.frontiersin.org/journals/computational-neuroscience/articles/10.3389/fncom.2023.1153572/full	NÃO

G Tuckute	Many but not all deep neural network audio models capture brain responses and exhibit correspondence between model stages and brain regions	34	2023	https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.3002366	SIM
AH Marblestone	Toward an Integration of Deep Learning and Neuroscience	842	2016	https://www.frontiersin.org/articles/10.3389/fncom.2016.00094/full?source=post_page----------	SIM
M Van Gerven	Computational Foundations of Natural Intelligence	97	2017	https://www.frontiersin.org/articles/10.3389/fncom.2017.00112/full?source=post_page----------	NÃO
GR Yang	Artificial neural networks for neuroscientists: a primer	319	2020	https://www.cell.com/neuron/fulltext/S0896-6273(20)30705-4	NÃO

A Doerig	The neuroconnectionist research programme	109	2023	https://www.nature.com/articles/s41583-023-00705-w	SIMM
DLK Yamins	Performance-optimized hierarchical models predict neural responses in hi	2194	2014	https://www.pnas.org/doi/abs/10.1073/pnas.1403112111	SIMM
BA Richards	A deep learning framework for neuroscience	894	2019	https://idp.nature.com/authorize/casa?redirect_uri=https://www.nature.com/articles/s41593-019-0520-2&casa_token=w8ns6l4cGbQAAAAA:WxoBRGLE5xUSTwibYR1ObfWRE78qk6PvqZTUgzaV_ZLLrcnb5Ys6t9g2vY1JHy3DCGpcRWFkzmlBRnyLKnE	JÁ LI

M Schrimpf	Brain-Score: Which Artificial Neural Network for Object Recognition is most Brain-Like?	545	2018	https://www.biorxiv.org/content/10.1101/407007.abstract	JÁ LI
F Pulvermüller	Biological constraints on neural network models of cognitive function	119	2021	https://www.nature.com/articles/s41583-021-00473-5?fromPaywallRec=false	SIM
T Macpherson	Natural and Artificial Intelligence: A brief introduction to the interplay between AI and neuroscience research	130	2021	https://www.sciencedirect.com/science/article/pii/S0893608021003683#b124	TALVEZ
N Kriegeskorte	Deep Neural Networks: A New Framework for Modeling Biological Vision and Brain Information Processing	1192	2015	https://www.annualreviews.org/content/journals/10.1146/annurev-vision-082114-035447	TALVEZ

SM Khaligh-Razavi	Deep Supervised, but Not Unsupervised, Models May Explain IT Cortical Representation	1323	2014	https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1003915	SIMM
A Défossez	Decoding speech perception from non-invasive brain recording	120	2023	https://arxiv.org/pdf/2208.12266 https://ai.meta.com/blog/ai-speech-brain-activity/	SIM
J Millet	Toward a realistic model of speech processing in the brain with self-supervised learning	78	2022	https://arxiv.org/pdf/2206.01685	TALVEZ
A Gulati	Conformer: Convolution-augmented Transformer for Speech Recognition	3242	2020	https://arxiv.org/pdf/2005.08100	SIM

KC Puvvada	Less is More: Accurate Speech Recognition & Translation without Web-Scale Data	5	2024	https://arxiv.org/pdf/2406.19674	NÃO
D Rekesh	FAST CONFORMER WITH LINEARLY SCALABLE ATTENTION FOR EFFICIENT SPEECH RECOGNITION	64	2023	https://arxiv.org/pdf/2305.05084	SIM
AJE Kell	A Task-Optimized Neural Network Replicates Human Auditory Behavior, Predicts Brain Responses, and Reveals a Cortical Processing Hierarchy	582	2018	https://www.sciencedirect.com/science/article/pii/S0896627318302502	SIMM
G Tuckute	Many but not all deep neural network audio models capture brain responses and exhibit correspondence between model stages	38	2023	https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.3002366	SIMM

	and brain regions				
Y Li	Dissecting neural computations in the human auditory pathway using deep neural networks for speech	41	2023	https://www.nature.com/articles/s41593-023-01468-4	SIMM
K Mahjoory	Convolutional neural networks can identify brain interactions involved in decoding spatial auditory attention	1	2024	https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1012376	TALVEZ
I Higgins	Unsupervised learning of temporal features for word categorization in a spiking neural network model of the auditory brain	18	2017	https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0180174	TALVEZ

APÊNDICE 3

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 9 de out. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O planejamento deste GATE baseou-se na análise detalhada dos trabalhos selecionados e no estudo de tópicos correlatos.

Durante a semana, explorei um novo artigo interessante sobre o mapeamento do cérebro de moscas, que, apesar de ser relevante para a área, não se alinha diretamente ao tema central da pesquisa:
https://www.nature.com/articles/d41586-024-03190-y

O foco principal desta semana foi a leitura e implementação do artigo:
https://www.nature.com/articles/nn.4244

A arquitetura proposta é inspirada em:
https://arxiv.org/abs/1410.0736

Outros trabalhos, como o do brain-score, mencionam a implementação desse artigo:
https://www.biorxiv.org/content/10.1101/407007v2.full

No entanto, não foi possível replicar completamente a implementação mencionada. Para contornar essa limitação, desenvolvi notebooks para testar os pontos principais do artigo:
hdcnn.ipynb
data_brain.ipynb

Estou considerando direcionar minha pesquisa para a investigação de representações de informações cerebrais. Para isso, já identifiquei uma competição interessante de leitura de EEG, na qual comecei a trabalhar:
https://www.kaggle.com/competitions/x-neuroengineering-symposium-challenge/data

Copy of X Symposium Competition notebook

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Estudar representações cerebrais, investigar trabalhos selecionados e implementar soluções viáveis. Continuar o desenvolvimento para a Competição de leitura de EEG.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[data_brain.ipynb]

```
import brainscore_vision
neural_data = brainscore_vision.load_dataset("MajajHong2015.public")

neural_data = neural_data.transpose('presentation', 'neuroid', 'time_bin')
```

```
/opt/conda/lib/python3.11/site-packages/brainscore_core/metrics/__init__.py
:16: FutureWarning: xarray subclass Score should explicitly define
__slots__
  class Score(DataAssembly):
brainio-brainscore/assy_dicarlo_MajajHong2015_public.nc: 100%|██████████|
165M/165M [00:21<00:00, 7.71MB/s]
brainio-brainscore/stimulus_hvm-public.csv: 100%|██████████| 1.33M/1.33M
[00:02<00:00, 521kB/s]
brainio-brainscore/stimulus_hvm-public.zip: 100%|██████████| 132M/132M
[00:14<00:00, 9.08MB/s]
```

Stimulus data

```
neural_data.attrs['stimulus_set']
```

```
        id                                background_id         s  \
0        1  ecd40f3f6d7a4d6d88134d648884e0b9b364efc9  1.000000
1        2  006d66c207c6417574f62f0560c6b2b40a9ec5a1  1.000000
2        3  3b3c1d65865028d0fad0b0bf8f305098db717e7f  1.000000
3        4  687ade2f9ee4d52af9705865395471a24ba38d5f  1.000000
4        5  724e5703cc42aa2c3ff135e3508038a90e4ebcb3  1.000000
...    ...                                       ...       ...
3195  3196  70222407751181c4b7723cb09dbda63e7f9c7333  1.286154
3196  3197  f144e3aabccefb4228b8257506a2aa26ba5b4a6d  1.234000
3197  3198  9b87c8be1440ce9626bcfe656700ff3ac8f909fe  0.895714
3198  3199  c3edce2a8fce8c088605bd27fe1f2e7958939f54  1.187143
3199  3200  91e68c65ff92e5a77f0fb13693bfe01bc429da18  0.795714

                                     image_id  \
0     8a72e2bfdb8c267b57232bf96f069374d5b21832
1     27f69468c9d6019ed0d22b9583c94c5b58198c1c
2     6af1cbb28aacea6c582faa07e92d8325fa7a29d7
3     d0f7a45b377d4920c3466ec7a20dce9437a150d6
4     d4c3b4d4aefd29fd168a2c3c9a9962d99653a715
...                                       ...
3195  caf392d996023d49d9be520da255f865a014f646
3196  4fefbea44756719f5cded1539b1c092e352beab0
3197  b1dbd1fc08edf4dc0814c5578e2c5544ea4d255d
3198  7c5174365cf0ed5a3319250d5913bcd8217f1454
3199  3ade751b115c7813365f97174e75106af2b88310
```

	image_file_name	\
0	astra_rx+00.000_ry+00.000_rz+00.000_tx+00.000_...	
1	_12_rx+00.000_ry+00.000_rz+00.000_tx+00.000_ty...	
2	face0003_rx+00.000_ry+00.000_rz+00.000_tx+00.0...	
3	walnut_obj_rx-90.000_ry+00.000_rz+00.000_tx+00...	
4	walnut_obj_rx-90.000_ry+00.000_rz+00.000_tx+00...	
...	...	
3195	_19_flyingBoat_rx+27.024_ry+104.520_rz+100.874...	
3196	Beetle_rx-06.302_ry-39.833_rz+22.244_tx-00.197...	
3197	_001_rx-13.140_ry-26.381_rz+09.423_tx+00.218_t...	
3198	_004_rx+11.926_ry-44.019_rz+38.652_tx-00.103_t...	
3199	_004_rx-36.603_ry+10.321_rz+16.609_tx+00.024_t...	

	filename	rxy	tz	\
0	astra_rx+00.000_ry+00.000_rz+00.000_tx+00.000_...	-0.000000	0.000	
1	_12_rx+00.000_ry+00.000_rz+00.000_tx+00.000_ty...	-0.000000	0.000	
2	face0003_rx+00.000_ry+00.000_rz+00.000_tx+00.0...	-0.000000	0.000	
3	walnut_obj_rx-90.000_ry+00.000_rz+00.000_tx+00...	-0.000000	0.000	
4	walnut_obj_rx-90.000_ry+00.000_rz+00.000_tx+00...	-0.000000	0.000	
...	...	...	...	
3195	_19_flyingBoat_rx+27.024_ry+104.520_rz+100.874...	27.783925	0.280	
3196	Beetle_rx-06.302_ry-39.833_rz+22.244_tx-00.197...	-39.833000	-0.042	
3197	_001_rx-13.140_ry-26.381_rz+09.423_tx+00.218_t...	-26.381000	0.000	
3198	_004_rx+11.926_ry-44.019_rz+38.652_tx-00.103_t...	-44.019000	0.521	
3199	_004_rx-36.603_ry+10.321_rz+16.609_tx+00.024_t...	10.321000	0.249	

	category_name	rxz_semantic	ty	ryz	object_name	variation	\
0	Cars	0.000000	0.000	-0.000000	car_astra	0	
1	Tables	0.000000	0.000	-0.000000	table3	0	
2	Faces	0.000000	0.000	-0.000000	face2	0	
3	Fruits	0.000000	0.000	-0.000000	walnut	0	
4	Fruits	0.000000	0.000	-0.000000	walnut	0	
...	...	...	...	...	...	...	
3195	Planes	-12.905496	0.096	17.585453	airplane2	3	
3196	Cars	-6.302000	-0.197	22.244000	car_beetle	3	
3197	Chairs	-13.140000	0.218	9.423000	chair0	3	
3198	Chairs	11.926000	-0.103	38.652000	chair1	3	
3199	Chairs	-36.603000	0.024	16.609000	chair1	3	

	size	rxy_semantic	ryz_semantic	rxz	\
0	256.0	90.000000	-0.000000	0.000000	
1	256.0	-0.000000	-0.000000	0.000000	
2	256.0	-0.000000	-0.000000	0.000000	

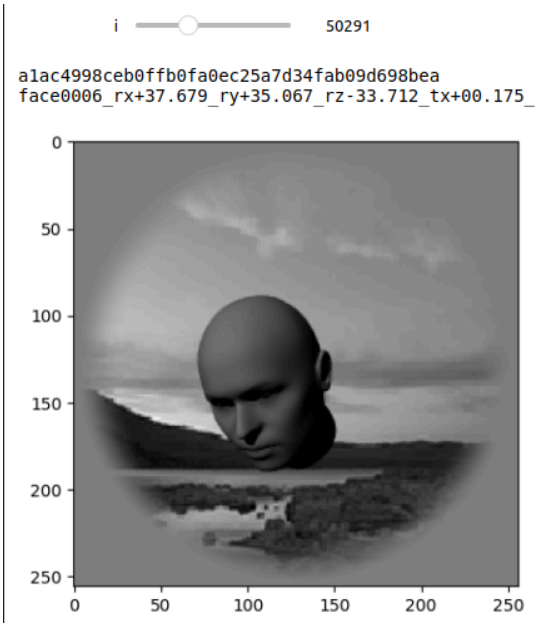
```
3     256.0     -0.000000     -0.000000   0.000000
4     256.0     -0.000000     -0.000000   0.000000
...     ...           ...           ...        ...
3195  256.0    117.783925     17.585453 -12.905496
3196  256.0     50.167000     22.244000  -6.302000
3197  256.0    -26.381000      9.423000 -13.140000
3198  256.0    -44.019000     38.652000  11.926000
3199  256.0     10.321000     16.609000 -36.603000

                                    stimulus_id
0     8a72e2bfdb8c267b57232bf96f069374d5b21832
1     27f69468c9d6019ed0d22b9583c94c5b58198c1c
2     6af1cbb28aacea6c582faa07e92d8325fa7a29d7
3     d0f7a45b377d4920c3466ec7a20dce9437a150d6
4     d4c3b4d4aefd29fd168a2c3c9a9962d99653a715
...                                       ...
3195  caf392d996023d49d9be520da255f865a014f646
3196  4fefbea44756719f5cded1539b1c092e352beab0
3197  b1dbd1fc08edf4dc0814c5578e2c5544ea4d255d
3198  7c5174365cf0ed5a3319250d5913bcd8217f1454
3199  3ade751b115c7813365f97174e75106af2b88310

[3200 rows x 19 columns]
```

```
neural_data.attrs['stimulus_set'].shape
```

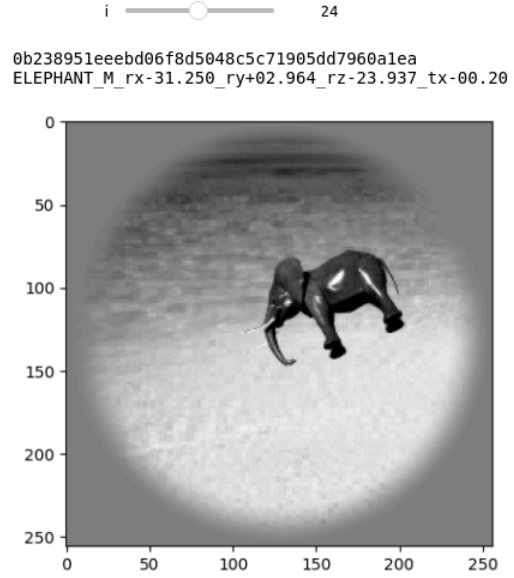
```
(3200, 19)
```

```
%matplotlib inline
from matplotlib import pyplot as plt, image
from ipywidgets import interact

stimulus_set = neural_data.attrs['stimulus_set']
selected_set = stimulus_set
stimulus_id  = neural_data['stimulus_id'].values

def show_image(i):
    id = stimulus_id[i]
    local_path = selected_set.get_stimulus(id)
    print(id)
    print(str(local_path).split('/')[-1])
    img = image.imread(local_path)
    plt.imshow(img)
    plt.show()

# Use o widget interativo para controlar o valor de i
```

```
interact(show_image, i=(0, len(stimulus_id)-1));
```




```
obj  ='ELEPHANT'
selected_set = stimulus_set[stimulus_set['object_name'].str.contains(obj,
case=False)]

stimulus_id  = list(selected_set.stimulus_id)

interact(show_image, i=(0, len(stimulus_id)-1))
```



Neural recording

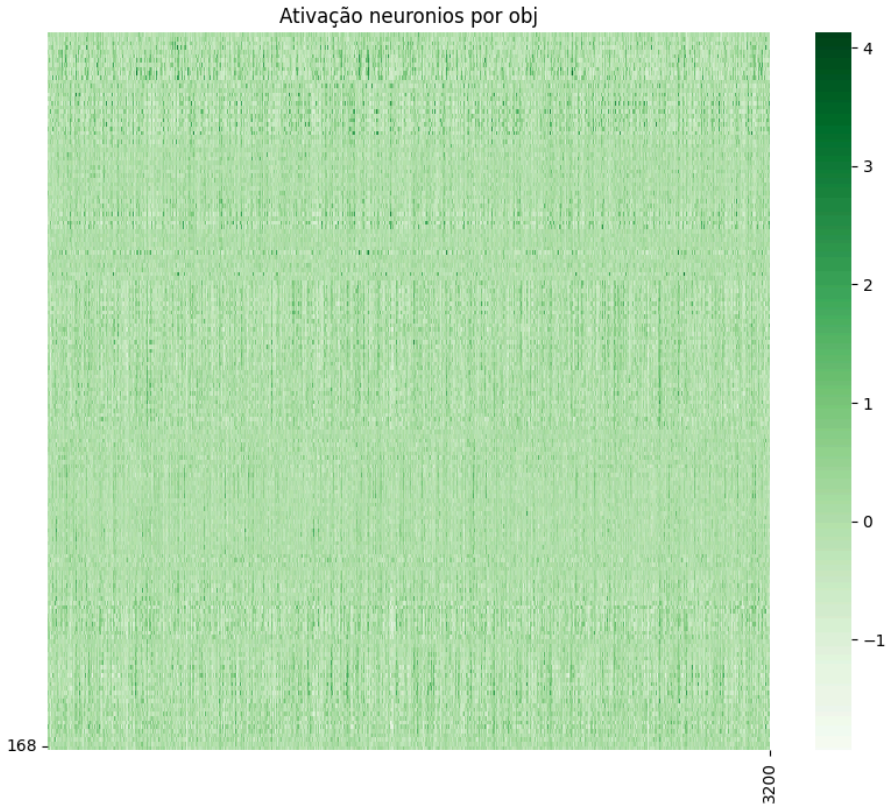
```
from brainscore_vision.benchmark_helpers.neural_common import
average_repetition
benchmark_data = neural_data.sel(region='IT')
benchmark_data = average_repetition(benchmark_data)
benchmark_data = benchmark_data.squeeze('time_bin')
benchmark_data[0].values
```

```
array([ 0.08579675, -0.40017223, …,  0.00478193], dtype=float32)
```

```
benchmark_data.shape
```

```
(3200, 168)
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Supondo que benchmark_data seja o seu xarray
data = benchmark_data.values

# Transpondo a matriz
data_transposed = data.T

# Criando o heatmap com uma coloração de verde mais escura para valores
maiores
plt.figure(figsize=(10, 8))
ax = sns.heatmap(data_transposed, cmap='Greens', cbar=True)

ax.set_xticks([data_transposed.shape[1] - 1])  # Último tick no eixo x
ax.set_xticklabels([data_transposed.shape[1]])  # Exibe o valor
correspondente
ax.set_yticks([data_transposed.shape[0] - 1])  # Último tick no eixo y
ax.set_yticklabels([data_transposed.shape[0]])  # Exibe o valor
correspondente



plt.title('Ativação neuronios por obj')
plt.show()
```


Ativação neuronios por obj
4
3
2
1
0
−1
168
3200

[hdcnn.ipynb]

Dependências

```
import keras as kr
import numpy as np
import tensorflow as tf

from keras.datasets import cifar100

from sklearn.model_selection import train_test_split

from random import randint
import time
import os
```

https://github.com/justinessert/hierarchical-deep-cnn/tree/master

CIFAR 100

```
import ssl

ssl._create_default_https_context = ssl._create_unverified_context

coarse_categories = 20
fine_categories = 100

(X, y_c), (x_test, y_c_test) = cifar100.load_data(label_mode='coarse')
(X, y), (x_test, y_test) = cifar100.load_data(label_mode='fine')

import matplotlib.pyplot as plt

i=0
plt.imshow(X[i])
plt.title(f'Categoria Coarse: {y_c[i]} Categoria Fina: {y[i]}')
plt.axis('off')
plt.show()
```


```
import matplotlib.pyplot as plt

i=9
plt.imshow(X[i])
plt.title(f'Categoria Coarse: {y_c[i]} Categoria Fina: {y[i]}')
plt.axis('off')
plt.show()
```


```
#(Ideally, this would be done through spectral clustering as opposed to
hard-coding)
fine2coarse = np.zeros((fine_categories,coarse_categories))
for i in range(coarse_categories):
    index = np.where(y_c_test[:,0] == i)[0]
    fine_cat = np.unique([y_test[j,0] for j in index])
    for j in fine_cat:
        fine2coarse[j,i] = 1

y_c = 0; # Clear y_c in interest of saving mem
y_c_test=0

def one_hot(y):
    n_values = np.max(y) + 1
    y_new = np.eye(n_values)[y[:,0]]
    return y_new

y=one_hot(y)
y_test=one_hot(y_test)
print(np.shape(y))

(50000, 100)
```

```
import numpy as np
import tensorflow as tf
import time

def zca(x_1, x_2, epsilon=1e-5):
    # aplica ZCA Whitening ao conjunto de imagens
    with tf.name_scope('ZCA'):
        # Converter os dados para o tipo tf.float64
        flatx = tf.cast(tf.reshape(x_1, (-1, np.prod(x_1.shape[-3:]))),
name="reshape_flat"), tf.float64, name="flatx")
        sigma = tf.tensordot(tf.transpose(flatx), flatx, 1, name="sigma") /
tf.cast(tf.shape(flatx)[0], tf.float64)

        # Decomposição SVD
        s, u, v = tf.linalg.svd(sigma, name="svd")
        pc = tf.tensordot(tf.tensordot(u, tf.linalg.diag(1. / tf.sqrt(s +
epsilon)), 1, name="inner_dot"), tf.transpose(u), 1, name="pc")

        # Aplicar ZCA Whitening às imagens
        net1 = tf.tensordot(flatx, pc, 1, name="whiten1")
        net1 = tf.reshape(net1, np.shape(x_1), name="output1")

        flatx2 = tf.cast(tf.reshape(x_2, (-1, np.prod(x_2.shape[-3:]))),
name="reshape_flat2"), tf.float64, name="flatx2")
        net2 = tf.tensordot(flatx2, pc, 1, name="whiten2")
        net2 = tf.reshape(net2, np.shape(x_2), name="output2")

    # Executar os cálculos e retornar os resultados
    return net1.numpy(), net2.numpy()

# Teste da função
time1 = time.time()
X, x_test = zca(X, x_test)
time2 = time.time()
print('Time Elapsed - ZCA Whitening: ' + str(time2 - time1))
```

```
Time Elapsed - ZCA Whitening: 55.6119818687439
```

```
x_train, x_val, y_train, y_val = train_test_split(X, y, test_size=.1,
random_state=0)
X = 0
y = 0


import tensorflow as tf
import time
```

```
def preprocess_img(X, y):
    # Essa função adiciona padding às imagens, faz crops aleatórios e flips
aleatórios
    with tf.name_scope('Preproc'):
        # Aplicar flip horizontal aleatório
        net = tf.map_fn(lambda img: tf.image.flip_left_right(img), X)

        # Aplicar rotação de 90 graus
        net = tf.map_fn(lambda img: tf.image.rot90(img), net)

        # Redimensionar imagem com padding e fazer crops aleatórios
        net = tf.image.resize_with_crop_or_pad(net, 40, 40)
        net = tf.map_fn(lambda img: tf.image.random_crop(img, [32, 32, 3]),
net)

        # Aplicar o mesmo processo ao conjunto original de imagens
        net1 = tf.image.resize_with_crop_or_pad(X, 40, 40)
        net1 = tf.map_fn(lambda img: tf.image.random_crop(img, [32, 32,
3]), net1)

        # Concatenar os dois conjuntos de imagens
        net = tf.concat([net, net1], axis=0)
        net_labels = tf.concat([y, y], axis=0)

        # Shuffle nas imagens e nos rótulos
        indices = tf.range(start=0, limit=tf.shape(net)[0], dtype=tf.int32)
        shuffled_indices = tf.random.shuffle(indices, seed=0)
        net = tf.gather(net, shuffled_indices)
        net_labels = tf.gather(net_labels, shuffled_indices)

        # Aplicar flip vertical aleatório
        net = tf.map_fn(lambda img: tf.image.random_flip_up_down(img), net)

    # Retorna os resultados (as operações já são avaliadas automaticamente)
    return net.numpy(), net_labels.numpy()

# Teste da função
time1 = time.time()
x_train, y_train = preprocess_img(x_train, y_train)
time2 = time.time()
print('Time Elapsed - Img Preprocessing: ' + str(time2 - time1))
```

```
Time Elapsed - Img Preprocessing: 324.2377233505249
```

HDCNN

```
from keras import optimizers
from keras.layers import Input, Conv2D, Dropout, MaxPooling2D, Flatten,
Dense
from keras.models import Model

class HDCNN:
    def __init__(self, input_shape=(32, 32, 3), num_classes=100):
        self.input_shape = input_shape
        self.num_classes = num_classes
        self.model = self.build_model()

    def build_model(self):
        in_layer = Input(shape=self.input_shape, dtype='float32',
name='main_input')

        net = Conv2D(384, 3, strides=1, padding='same',
activation='elu')(in_layer)
        net = MaxPooling2D((2, 2), padding='valid')(net)

        net = Conv2D(384, 1, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(384, 2, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(640, 2, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(640, 2, strides=1, padding='same',
activation='elu')(net)
        net = Dropout(.2)(net)
        net = MaxPooling2D((2, 2), padding='valid')(net)

        net = Conv2D(640, 1, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(768, 2, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(768, 2, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(768, 2, strides=1, padding='same',
activation='elu')(net)
        net = Dropout(.3)(net)
        net = MaxPooling2D((2, 2), padding='valid')(net)

        net = Conv2D(768, 1, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(896, 2, strides=1, padding='same',
```

```
activation='elu')(net)
        net = Conv2D(896, 2, strides=1, padding='same',
activation='elu')(net)
        net = Dropout(.4)(net)
        net = MaxPooling2D((2, 2), padding='valid')(net)

        net = Conv2D(896, 3, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(1024, 2, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(1024, 2, strides=1, padding='same',
activation='elu')(net)
        net = Dropout(.5)(net)
        net = MaxPooling2D((2, 2), padding='valid')(net)

        net = Conv2D(1024, 1, strides=1, padding='same',
activation='elu')(net)
        net = Conv2D(1152, 2, strides=1, padding='same',
activation='elu')(net)
        net = Dropout(.6)(net)
        net = MaxPooling2D((2, 2), padding='same')(net)

        net = Flatten()(net)
        net = Dense(1152, activation='elu')(net)
        net = Dense(self.num_classes, activation='softmax')(net)

        return Model(inputs=in_layer, outputs=net)

    def compile_model(self, optimizer):
        self.model.compile(optimizer= optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])

    def get_model(self):
        return self.model

hdcnn = HDCNN()

sgd_coarse = optimizers.SGD(learning_rate=0.01, decay=1e-6, momentum=0.9,
nesterov=True)
hdcnn.compile_model(sgd_coarse)

model = hdcnn.get_model()
```

Train

```
tbCallBack = kr.callbacks.TensorBoard(log_dir='./data/graph/elu_drop/',
histogram_freq=0, write_graph=True, write_images=True)
batch = 64

index= 0
step = 1
stop = 2

with tf.device('/gpu:0'):
    while index < stop:
        model.fit(x_train, y_train, batch_size=batch, initial_epoch=index,
epochs=index + step,
                  validation_data=(x_val, y_val), callbacks=[tbCallBack])
        index += step
        model.save_weights('data/models/model_coarse' + str(index))

save_index = index
```

```
   2/1407 ━━━━━━━━━━━━━━━━━━━━ 2:30:01 6s/step - accuracy: 0.0000e+00 -
loss: 4.5979
```

```
# Load model


for i in range(len(model.layers)):
    model.layers[i].trainable=False
```

Fine tune Coarser

```
y_train_c = np.dot(y_train,fine2coarse)
y_val_c = np.dot(y_val,fine2coarse)


in_layer = Input(shape=(32, 32, 3), dtype='float32', name='main_input')
net = Conv2D(1024, 1, strides=1, padding='same',
activation='elu')(model.layers[-8].output)
net = Conv2D(1152, 2, strides=1, padding='same', activation='elu')(net)
net = Dropout(.6)(net)
net = MaxPooling2D((2, 2), padding='same')(net)

net = Flatten()(net)
```

```
net = Dense(1152, activation='elu')(net)
out_coarse = Dense(20, activation='softmax')(net)

model_c = Model(inputs=in_layer,outputs=out_coarse)
model_c.compile(optimizer= sgd_coarse, loss='categorical_crossentropy',
metrics=['accuracy'])

for i in range(len(model_c.layers)-1):
    model_c.layers[i].set_weights(model.layers[i].get_weights())

index = 30
step = 10
stop = 40

while index < stop:
    model_c.fit(x_train, y_train_c, batch_size=batch, initial_epoch=index,
epochs=index+step, validation_data=(x_val, y_val_c),
callbacks=[tbCallBack])
    index += step

sgd_fine = optimizers.SGD(lr=0.001, decay=1e-6, momentum=0.9,
nesterov=True)
# troca optimizer pelo fine
model_c.compile(optimizer=sgd_fine, loss='categorical_crossentropy',
metrics=['accuracy'])
stop = 50

while index < stop:
    model_c.fit(x_train, y_train_c, batch_size=batch, initial_epoch=index,
epochs=index+step, validation_data=(x_val, y_val_c),
callbacks=[tbCallBack])
    index += step
```

Fine tune fine

```
def fine_model():
    net = Conv2D(1024, 1, strides=1, padding='same',
activation='elu')(model.layers[-8].output)
    net = Conv2D(1152, 2, strides=1, padding='same', activation='elu')(net)
    net = Dropout(.6)(net)
    net = MaxPooling2D((2, 2), padding='same')(net)

    net = Flatten()(net)
    net = Dense(1152, activation='elu')(net)
    out_fine = Dense(100, activation='softmax')(net)
```

```
    model_fine = Model(inputs=in_layer,outputs=out_fine)
    model_fine.compile(optimizer= sgd_coarse,
              loss='categorical_crossentropy',
              metrics=['accuracy'])

    for i in range(len(model_fine.layers)-1):
        model_fine.layers[i].set_weights(model.layers[i].get_weights())
    return model_fine

fine_models = {'models' : [{} for i in range(coarse_categories)], 'yhf' :
[{} for i in range(coarse_categories)]}
for i in range(coarse_categories):
    model_i = fine_model()
    fine_models['models'][i] = model_i
```

Evaluation

```
def get_error(y,yh):
    # Threshold
    yht = np.zeros(np.shape(yh))
    yht[np.arange(len(yh)), yh.argmax(1)] = 1
    # Evaluate Error
    error = np.count_nonzero(np.count_nonzero(y-yht,1))/len(y)
    return error

for i in range(coarse_categories):
    index= 0
    step = 5
    stop = 5

    # Get all training data for the coarse category
    ix = np.where([(y_train[:,j]==1) for j in [k for k, e in
enumerate(fine2coarse[:,i]) if e != 0]])[1]
    x_tix = x_train[ix]
    y_tix = y_train[ix]

    # Get all validation data for the coarse category
    ix_v = np.where([(y_val[:,j]==1) for j in [k for k, e in
enumerate(fine2coarse[:,i]) if e != 0]])[1]
    x_vix = x_val[ix_v]
    y_vix = y_val[ix_v]

    while index < stop:
        fine_models['models'][i].fit(x_tix, y_tix, batch_size=batch,
initial_epoch=index, epochs=index+step, validation_data=(x_vix, y_vix))
        index += step
```

```
    fine_models['models'][i].compile(optimizer=sgd_fine,
loss='categorical_crossentropy', metrics=['accuracy'])
    stop = 10

    while index < stop:
        fine_models['models'][i].fit(x_tix, y_tix, batch_size=batch,
initial_epoch=index, epochs=index+step, validation_data=(x_vix, y_vix))
        index += step

    yh_f = fine_models['models'][i].predict(x_val[ix_v], batch_size=batch)
    print('Fine Classifier '+str(i)+' Error:
'+str(get_error(y_val[ix_v],yh_f)))


def eval_hdcnn(X, y):
    yh = np.zeros(np.shape(y))

    yh_s = model.predict(X, batch_size=batch)

    print('Single Classifier Error: '+str(get_error(y,yh_s)))

    yh_c = model_c.predict(X, batch_size=batch)
    y_c = np.dot(y,fine2coarse)

    print('Coarse Classifier Error: '+str(get_error(y_c,yh_c)))

    for i in range(coarse_categories):
        if i%5 == 0:
            print("Evaluating Fine Classifier: ", str(i))
        #fine_models['yhf'][i] = fine_models['models'][i].predict(X,
batch_size=batch)
        yh += np.multiply(yh_c[:,i].reshape((len(y)),1),
fine_models['yhf'][i])

    print('Overall Error: '+str(get_error(y,yh)))
    return yh

yh = eval_hdcnn(x_val,y_val)
```

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 17 de out. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

O desenvolvimento desta semana foi dedicado a um estudo mais detalhado do artigo "*Using goal-driven deep learning models to understand sensory cortex*" e de trabalhos correlatos.

Inicialmente, analisei o artigo "*Performance-optimized hierarchical models predict neural responses in higher visual cortex* " do mesmo autor e pioneiro ao anterior, porém não foi fornecido nenhuma informação adicional aparente.

Decidi entrar em contato com um dos colaboradores do projeto Brain-Score, que reporta a implementação em Brain-Score: Which Artificial Neural Network for Object Recognition is most Brain-Like? | bioRxiv , que me sugeriu dialogar diretamente com o autor original.

Durante esse período, dediquei-me ao estudo de trabalhos anteriores e optei por explorar o seguinte artigo do mesmo autor:
"*Hierarchical Modular Optimization of Convolutional Networks Achieves Representations Similar to Macaque IT and Human Ventral Stream*"

Este estudo esclareceu muitos aspectos para mim, no entanto, ainda enfrento desafios para replicar totalmente o primeiro artigo devido a duas pendências principais: a falta de clareza sobre o dataset utilizado e a falta de detalhe dos parâmetros para o processo de otimização da arquitetura.

Apesar dessas dificuldades, decidi prosseguir com a implementação para fins de estudo:
yammins2014.ipynb

Adicionalmente, explorei implementações mais antigas relacionadas às regiões precursoras do processamento visual no cérebro. Esse estudo foi consolidado nesse notebook:

GalborFilters_and_V1.ipynb

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Caso o autor do artigo responda, seguirei suas

instruções para avançar na replicação do estudo. Caso contrário, realizarei uma série de testes experimentais para tentar replicar os resultados apresentados no artigo.

Paralelamente, planejo expandir a pesquisa para incluir implementações mais recentes, comparando-as com outras abordagens aplicadas não apenas ao processamento visual, mas também auditivo, conforme sugerido no artigo como um possível campo de exploração.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[GalborFilters_and_V1.ipynb]

```
# Baseado em
https://www.johancarlin.com/gabor-filter-models-for-visual-neuroscience.html
```

```
%%capture
!pip install gwp
```

```
import matplotlib.pyplot as plt
import numpy as np
import warnings
# TF maintainers need to update their Numpy
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=FutureWarning)
    import tensorflow as tf
import skimage.data
import gwp
plt.rc('image', cmap='gray')
```

Gabor wavelet

Wavelets são muito úteis para análise de sinais não estacionários, pois a incerteza da informação transmitida é minimizada, ou seja, comparado a uma transformada de fourrier, conseguimos melhor informação tanto temporal quanto espectral . "it minimizes the product of its standard deviations in the time and frequency domain".

A gabor wavelet é uma dessas wavelets, e é dada por.

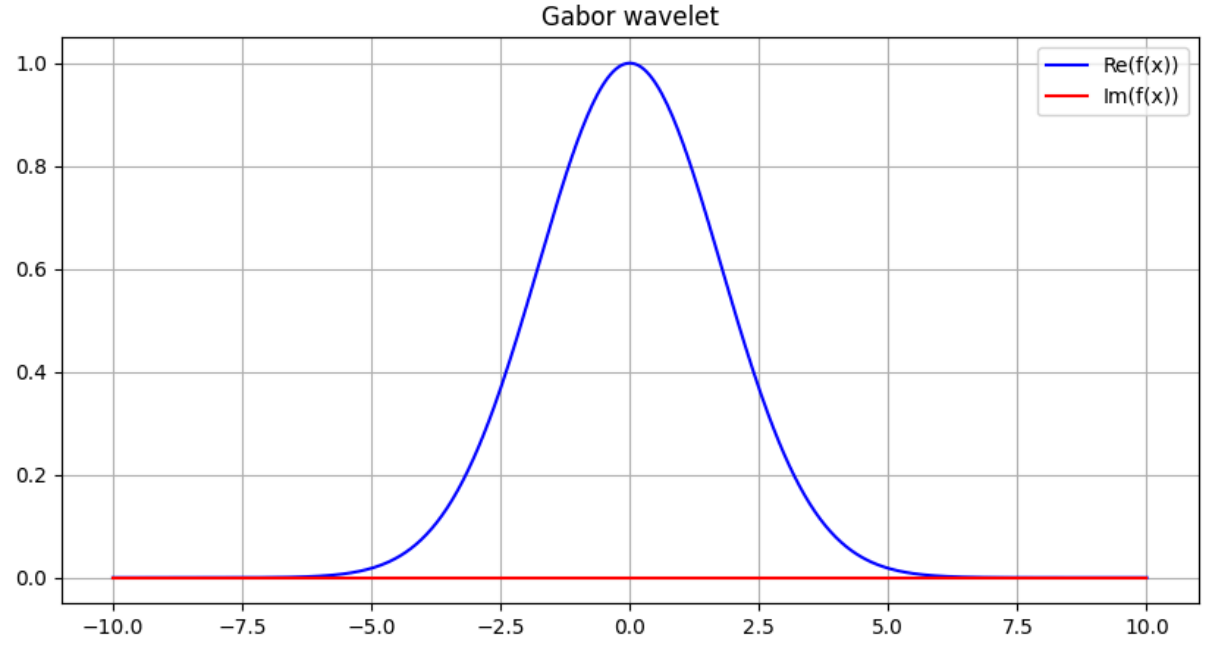
$$f(x) = e^{-\frac{(x-x_0)^2}{a^2}} e^{-ik_0(x-x_0)}$$

Onde:

- x é a posição atual
- x_0 é a posição de referência
- a : Controla a taxa de decaimento exponencial. Um valor maior de a a resulta em uma diminuição mais lenta da função à medida que se afasta de x_0 .
- k_0 : Controla a taxa de modulação (frequência). Este parâmetro está relacionado ao comprimento de onda ou à frequência da oscilação da função.

```
#@title Exemplo
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact
import matplotlib.cm as cm
```

```
def f(x, x0, a, k0):
    return np.exp(-((x - x0)**2) / a**2) * np.exp(-1j * k0 * (x - x0))

def plot_function(x0, a, k0):
    x = np.linspace(-10, 10, 400)
    y = f(x, x0, a, k0)

    plt.figure(figsize=(10, 5))

    plt.plot(x, np.real(y), label="Re(f(x))", color='blue')
    plt.plot(x, np.imag(y), label="Im(f(x))", color='red')
    plt.title(f'Gabor wavelet')
    plt.legend()
    plt.grid(True)
    plt.show()

# Interatividade com sliders
interact(plot_function, x0=(-5.0, 5.0, 0.1), a=(0.1, 5.0, 0.1), k0=(-10.0,
10.0, 0.1));
```



Gabor Filter

Uma **Gabor Filter** é uma extensão de uma gabor wavelet, função composta por uma onda sinusoidal modulada por uma gaussiana. A equação básica é:

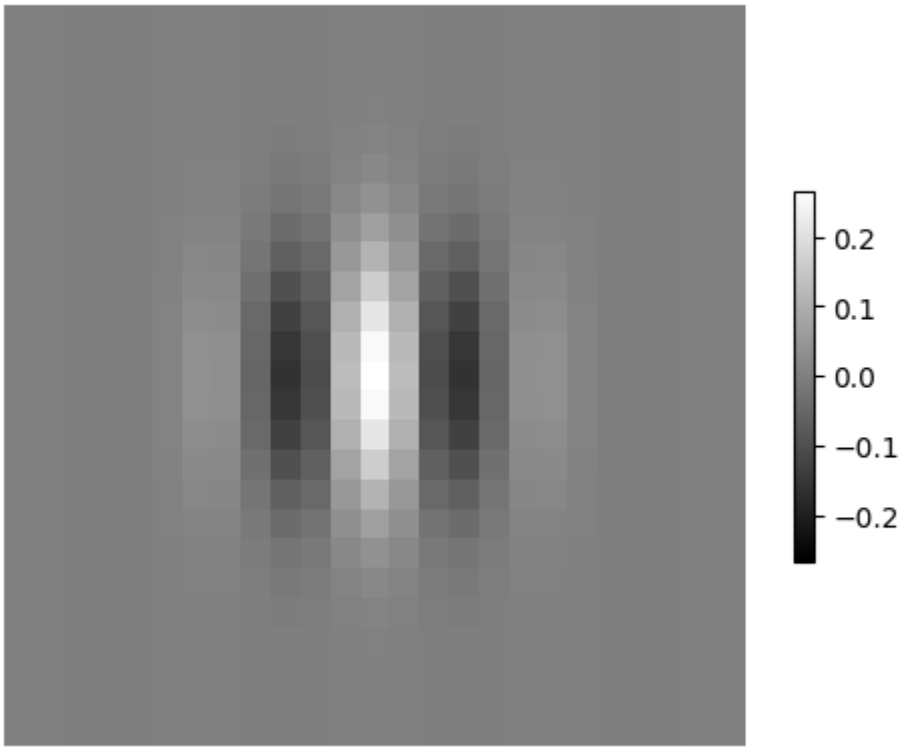
$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\exp\left(i\left(2\pi\frac{x'}{\lambda} + \psi\right)\right)$$

Onde:

- x' e y' são as coordenadas rotacionadas.
- λ é o comprimento de onda da onda sinusoidal (definindo a escala).
- θ é a orientação da wavelet (direção do filtro).
- ψ é o deslocamento de fase da componente sinusoidal.
- σ controla o alcance da gaussiana.
- γ define a excentricidade da elipse, controlando a proporção entre as frequências em x e y.

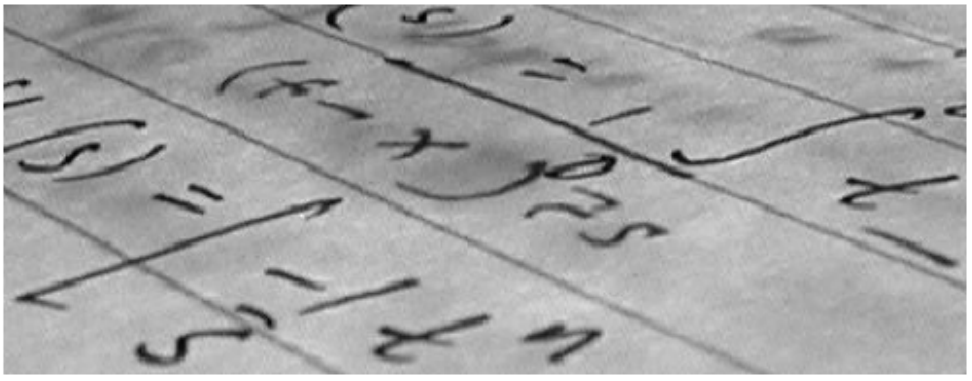
```
#@title Exemplo
sigma = 3.


import numpy as np
import matplotlib.pyplot as plt
import skimage.filters
from ipywidgets import interact

def plot_gabor(sigma=3.0, orientation=0.0, phase=0.0, size_std=4.0,
frequency=0.5 / sigma):
    """
    Função para gerar e plotar o filtro Gabor baseado nos parâmetros.
    """


    # Gera o kernel de Gabor
    gabor = np.real(
                skimage.filters.gabor_kernel(
                    frequency=frequency,
                    theta=orientation,
                    sigma_x=sigma,
                    sigma_y=sigma,
                    offset=phase,
                    n_stds=size_std,
```

```
                )
            )

    # Normaliza o filtro para comparação
    gabor /= np.linalg.norm(gabor)
    absmax = np.max(np.abs(gabor))

    # Plota o filtro Gabor
    fig, ax = plt.subplots(1, 1)
    ax.set_axis_off()
    im = ax.imshow(gabor, vmin=-absmax, vmax=absmax)
    fig.colorbar(im, shrink=.5)
    plt.show()

interact(
    plot_gabor,
    sigma=(0.1, 10.0, 0.1),
    orientation=(0.0, 2*np.pi, 0.1),
    phase=(0.0, 2*np.pi, 0.1),
    size_std=(1.0, 10.0, 0.1)
);
```



Filter banks and energy

```
#@title  imagem
image = skimage.data.text().astype(float)
image /= image.max()
fig, ax = plt.subplots(1, 1)
ax.imshow(image)
ax.set_axis_off()
# reshape array to match convention (index by height by width by channel)
image = image[None, :, :, None]
dc = image.mean()
# zero mean to avoid picking up on edge effects
image -= dc
```



Ampliamos nossa representação de Gabor, construindo um modelo de energia padrão de Adelson & Bergen.

O modelo consiste em um filter bank de Gabor que variam em orientação (8) e fase (2 defasadas em quadratura). Os filtros são então convolvidos com a imagem em um determinado intervalo para obter uma resposta de filtro em cada local amostrado da imagem.

```
# configure gabor model
orientations = gwp.n2orientations(8)
# quadrature-offset filters
sigma = 3
bank = [gwp.gaborbank(sigma=sigma, orientations=orientations, nsigma=4,
phase=thisphase)
                for thisphase in [0, np.pi/2]]
bank[0].shape
```

```
(25, 25, 1, 8)
```

Cada valor de fase aparece como uma lista com [vertical, horizontal, 1, orientação], e a operação chave é calcular a raiz quadrada sobre os mapas de fase somados ao quadrado. Isso cria um **detector de borda invariante à fase**, permitindo detectar bordas dentro da janela Gaussiana, independentemente da fase. Além disso, retifica a saída, já que taxas de disparo neuronais não têm valores negativos. Em Adelson & Bergen (1985), essa operação imita como células complexas do V1 somam respostas de células simples sensíveis à fase para obter invariância à fase.

```
#  the energy representation of the two quadrature-phase filters becomes a
Gaussian.
energy = gwp.v1energy(*bank)
fig, ax = plt.subplots(1, 3, sharex=True, sharey=True)
[this_ax.set_axis_off() for this_ax in ax.ravel()]
bankmax = max(*[np.abs(thisbank).max() for thisbank in bank])
ax[0].imshow(bank[0][:,:,0,0], vmin=-bankmax, vmax=bankmax)
ax[0].set_title('phase=0')
ax[1].imshow(bank[1][:,:,0,0], vmin=-bankmax, vmax=bankmax)
ax[1].set_title('phase=π/2')
ax[2].imshow(energy[:,:,0,0])
ax[2].set_title('energy')
```

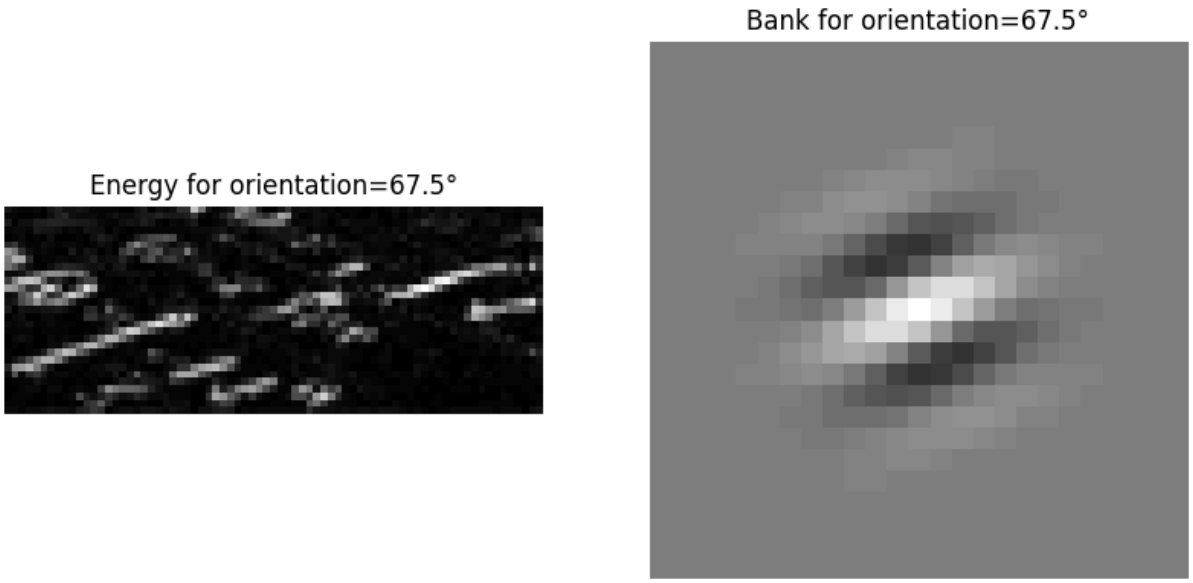
```
Text(0.5, 1.0, 'energy')
```


```
#@title  Como a saída de energia se apresenta para cada orientação.
```

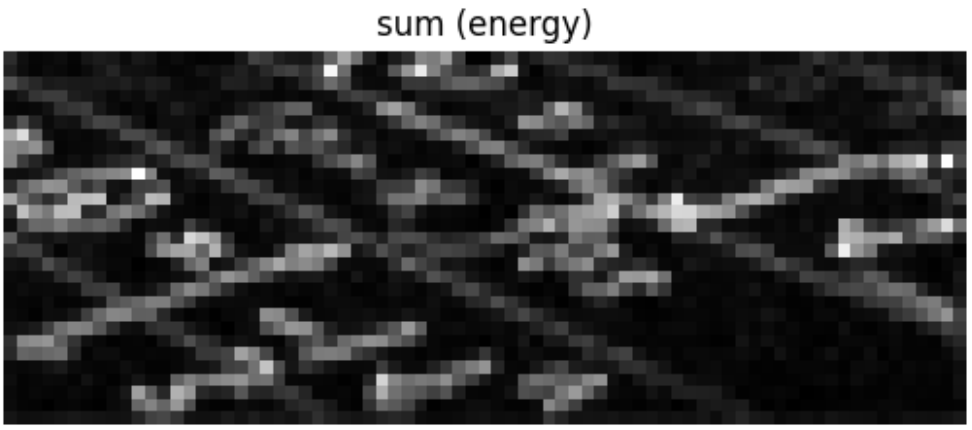
```
# convolve banks with stimulus to obtain raw responses
stride = 2 * sigma
responses = [gwp.convolver(image, thisphasebank, stride) for thisphasebank
in bank]
# calculate V1 energy (sum of squares)
energy = gwp.v1energy(*responses)
```

```
def plot_orientation(index):
    fig, ax = plt.subplots(1, 2, figsize=(10, 5))
```

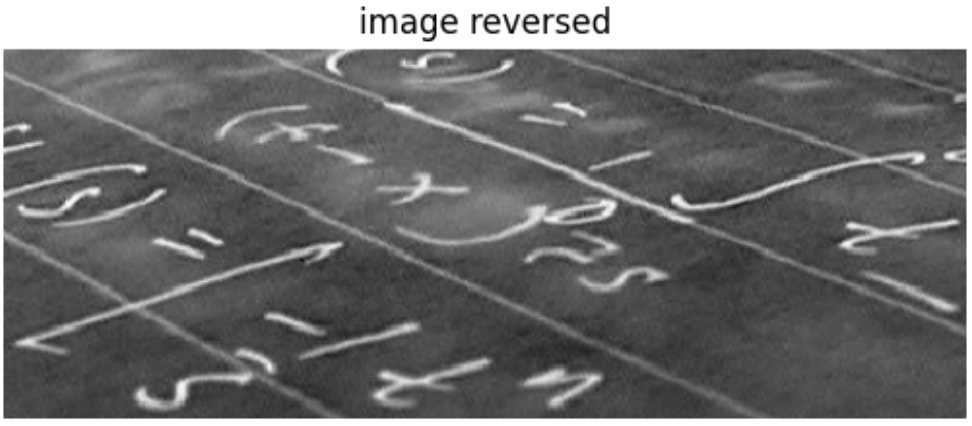
```
    ax[0].imshow(energy[0, :, :, index], vmin=0., cmap='gray')
    ax[0].set_title(f"Energy for orientation={orientations[index] * (180. /
np.pi):.1f}°")

    ax[1].imshow(bank[0][:, :, 0, index], vmin=-bankmax, vmax=bankmax,
cmap='gray')
    ax[1].set_title(f"Bank for orientation={orientations[index] * (180. /
np.pi):.1f}°")

    [axis.set_axis_off() for axis in ax]
    plt.show()

# Create interactive slider for orientation
interact(plot_orientation, index=(0, len(orientations) - 1));
```



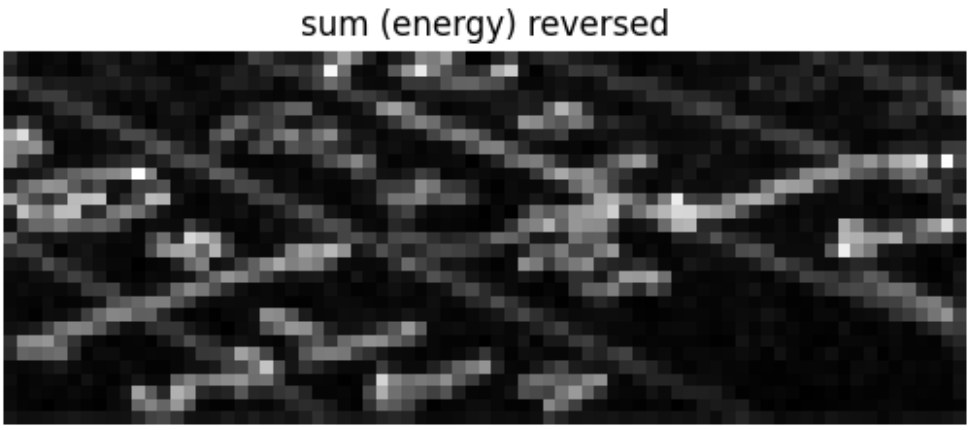

```
# @title Summing all orientations
# or just summing the orientation channels (that is, equal weight on all
orientations)
fig, ax = plt.subplots(1, 1)
ax.imshow(np.sum(energy[0,:,:,:], axis=2))
ax.set_title('sum (energy)')
ax.set_axis_off()
```




```
# @title Inverted image
image_invert = image * -1
fig, ax = plt.subplots(1, 1)
ax.imshow(dc + image_invert[0,:,:,0])
ax.set_title('image reversed')
ax.set_axis_off()
```




```
# @title Summing all orientations
energy_invert = gwp.v1energy(*[gwp.convolver(image_invert, thisphasebank,
stride) for thisphasebank in bank])
fig, ax = plt.subplots(1, 1)
ax.imshow(np.sum(energy_invert[0,:,:,:], axis=2))
ax.set_title('sum (energy) reversed')
ax.set_axis_off()
```



Este modelo de Gabor é sensível à posição, orientação e frequência espacial das bordas na imagem, mas não importa se as bordas são claras sobre um fundo escuro ou o inverso. Isso é geralmente desejável quando se pensa em um processo como a segregação figura-fundo (embora seja possível imaginar contraexemplos interessantes, como nos clássicos algoritmos de detecção de rostos, que dependem do sinal de relações de contraste local específicas na imagem).

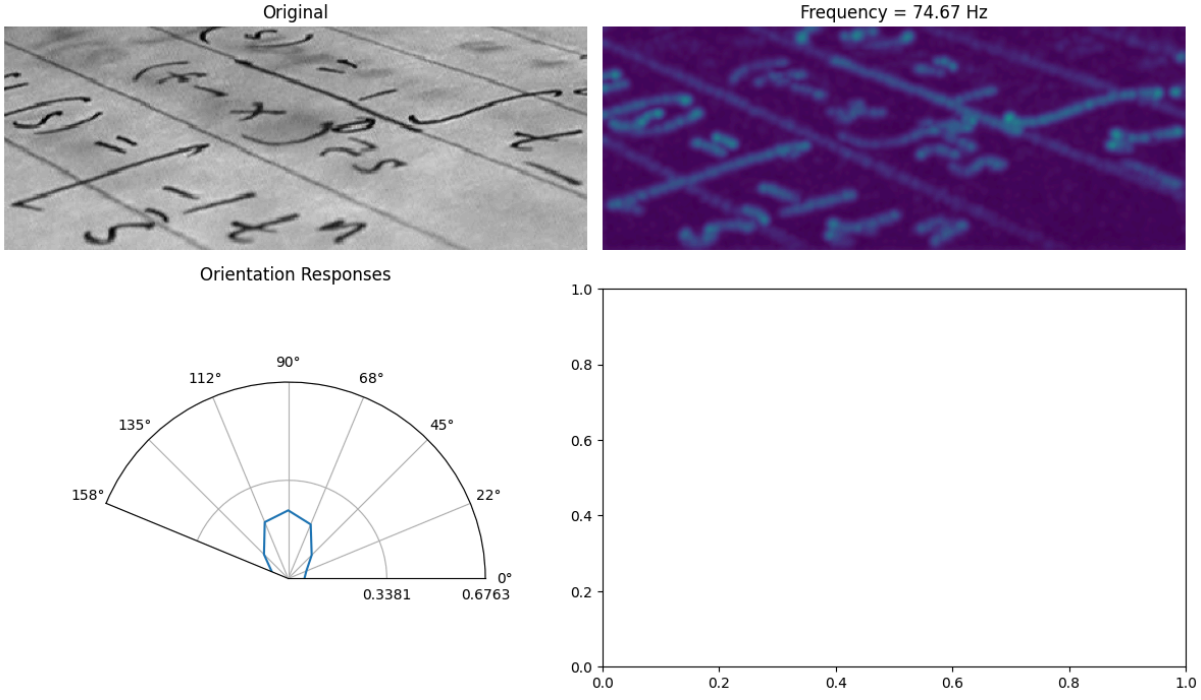
Spatial frequency banks

A parte da pirâmide no GWP vem da disposição típica do modelo em múltiplos bancos, onde cada banco apresenta filtros maiores (sigma) e um passo maior. A mudança no passo resulta em uma organização em pirâmide — o topo da pirâmide possui um pequeno número de filtros muito grandes, enquanto a base tem um grande número de filtros pequenos. Cada nível da pirâmide codifica uma determinada banda de frequência espacial de forma eficiente, usando menos filtros para codificar conteúdos de baixa frequência espacial e mais grosseiros.

```
# so to extend our model, we can add further filters like so
pyramid = {}
sigma_values = np.arange(2., 9.)
frequencies = {this_sigma: (0.5 / this_sigma) * image.shape[2] for
this_sigma in sigma_values}

for this_sigma, this_frequency in frequencies.items():
    pyramid[this_frequency] = [gwp.gaborbank(sigma=this_sigma,
orientations=orientations, nsigma=4, phase=thisphase)
                for thisphase in [0, np.pi/2]]


# energy at each level of the pyramid
```

```
def model_response(pyramid, image):
    pyramid_energy_map = {}
    pyramid_energy_orient = {}
    for this_frequency, this_bank in pyramid.items():
        # for ease of comparison, we set stride=1 here so that each feature
map has the same shape
        # (so not actually a pyramid but an apartment building)
        responses = [gwp.convolver(image, thisphasebank, 1) for
thisphasebank in this_bank]
        energy = gwp.v1energy(*responses)
        pyramid_energy_map[this_frequency] = np.sum(energy[0,:,:,:],
axis=2)
        pyramid_energy_orient[this_frequency] = np.mean(energy[0,:,:,:],
axis=(0,1))
    return pyramid_energy_map, pyramid_energy_orient


pyramid_energy_map, pyramid_energy_orient = model_response(pyramid, image)

# @title Visualizar
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact

def response_visualise(selected_frequency):
    plt.rcParams.update({'figure.autolayout': True})  # Adjust layout to
make room for all plots
    fig, axs = plt.subplots(2, 2, figsize=(12, 8),
subplot_kw={'projection': None})  # Specify no projection by default

    # Set projection for polar plot manually
    axs[1, 0].remove()  # Remove the original axis in this position
    polar_ax = fig.add_subplot(2, 2, 3, projection='polar')  # Add a polar
subplot in the correct position

    # Calculate maximum values for consistent color scales
    absmax = max(np.max(energy) for energy in pyramid_energy_map.values())
    orimax = max(np.max(orient) for orient in
pyramid_energy_orient.values())

    # Original Image
    axs[0, 0].imshow(dc + image[0, :, :, 0], cmap='gray')
    axs[0, 0].set_title('Original')
    axs[0, 0].axis('off')

    # Energy map for selected frequency
```

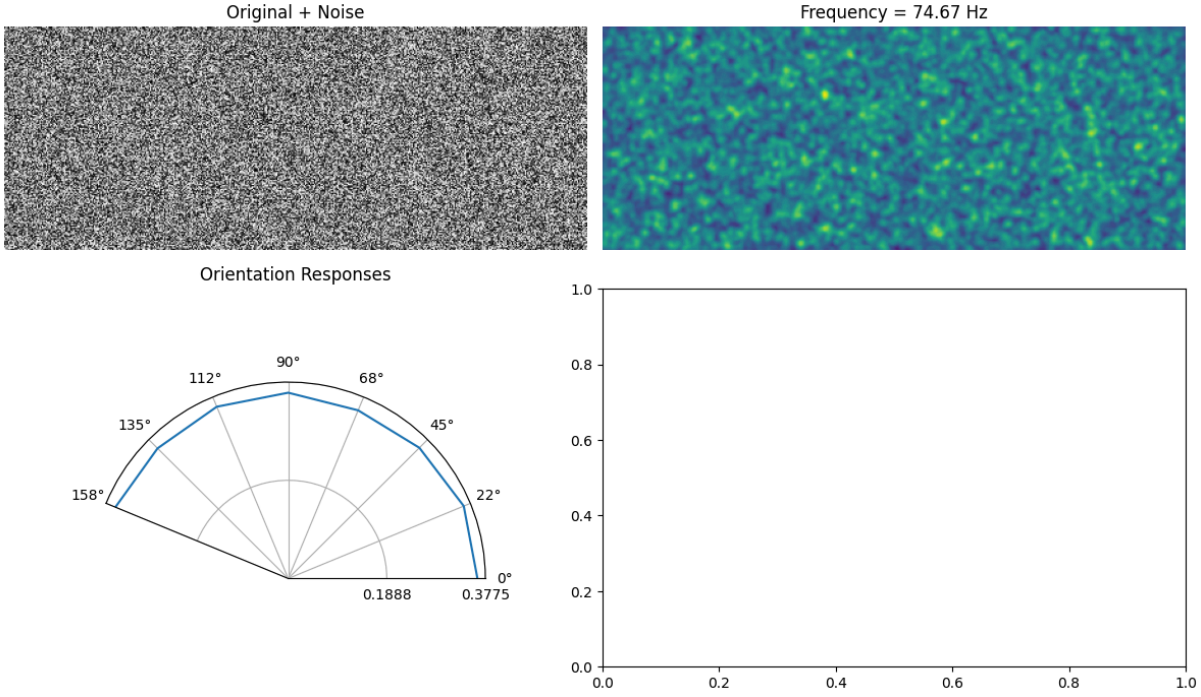
```
    axs[0, 1].imshow(pyramid_energy_map[selected_frequency], vmin=0,
vmax=absmax, cmap='viridis')
    axs[0, 1].set_title(f"Frequency = {selected_frequency:.2f} Hz")
    axs[0, 1].axis('off')

    # Polar plot for orientations
    polar_ax.plot(orientations, pyramid_energy_orient[selected_frequency])
    polar_ax.set_thetamin(np.degrees(np.min(orientations)))
    polar_ax.set_thetamax(np.degrees(np.max(orientations)))
    polar_ax.set_xticks(orientations)
    polar_ax.set_xticklabels([f"{np.degrees(angle):.0f}°" for angle in
orientations])
    polar_ax.set_rgrids([orimax / 2, orimax], angle=315)  # Example radii
labels at meaningful positions
    polar_ax.set_title('Orientation Responses')

    # Ensure the subplot spaces are utilized well
    plt.tight_layout()
    plt.show()

# Create an interactive slider to select frequency
interact(response_visualise,
selected_frequency=list(pyramid_energy_map.keys()));
```



A orientação dominante está em cerca de 90 graus. Além disso, a energia da borda parece aumentar à medida que a frequência espacial do filtro (em ciclos por largura da imagem) diminui.

```
# @title Verificar se artefatos existem em ruído
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact

def response_visualise(selected_frequency, pyramid_energy_map,
pyramid_energy_orient, image):
    plt.rcParams.update({'figure.autolayout': True})
    fig, axs = plt.subplots(2, 2, figsize=(12, 8),
subplot_kw={'projection': None})

    # Set projection for polar plot manually
    axs[1, 0].remove()  # Remove the original axis in this position
    polar_ax = fig.add_subplot(2, 2, 3, projection='polar')

    # Calculate maximum values for consistent color scales
    absmax = max(np.max(energy) for energy in pyramid_energy_map.values())
    orimax = max(np.max(orient) for orient in
pyramid_energy_orient.values())

    # Original Image
    axs[0, 0].imshow(image[0, :, :, 0] + 0.5, cmap='gray')  # Adjust the
range back to [0,1] for display
    axs[0, 0].set_title('Original + Noise')
    axs[0, 0].axis('off')

    # Energy map for selected frequency
    axs[0, 1].imshow(pyramid_energy_map[selected_frequency], vmin=0,
vmax=absmax, cmap='viridis')
    axs[0, 1].set_title(f"Frequency = {selected_frequency:.2f} Hz")
    axs[0, 1].axis('off')

    # Polar plot for orientations
    polar_ax.plot(orientations, pyramid_energy_orient[selected_frequency])
    polar_ax.set_thetamin(np.degrees(np.min(orientations)))
    polar_ax.set_thetamax(np.degrees(np.max(orientations)))
    polar_ax.set_xticks(orientations)
    polar_ax.set_xticklabels([f"{np.degrees(angle):.0f}°" for angle in
orientations])
    polar_ax.set_rgrids([orimax / 2, orimax], angle=315)
    polar_ax.set_title('Orientation Responses')
```

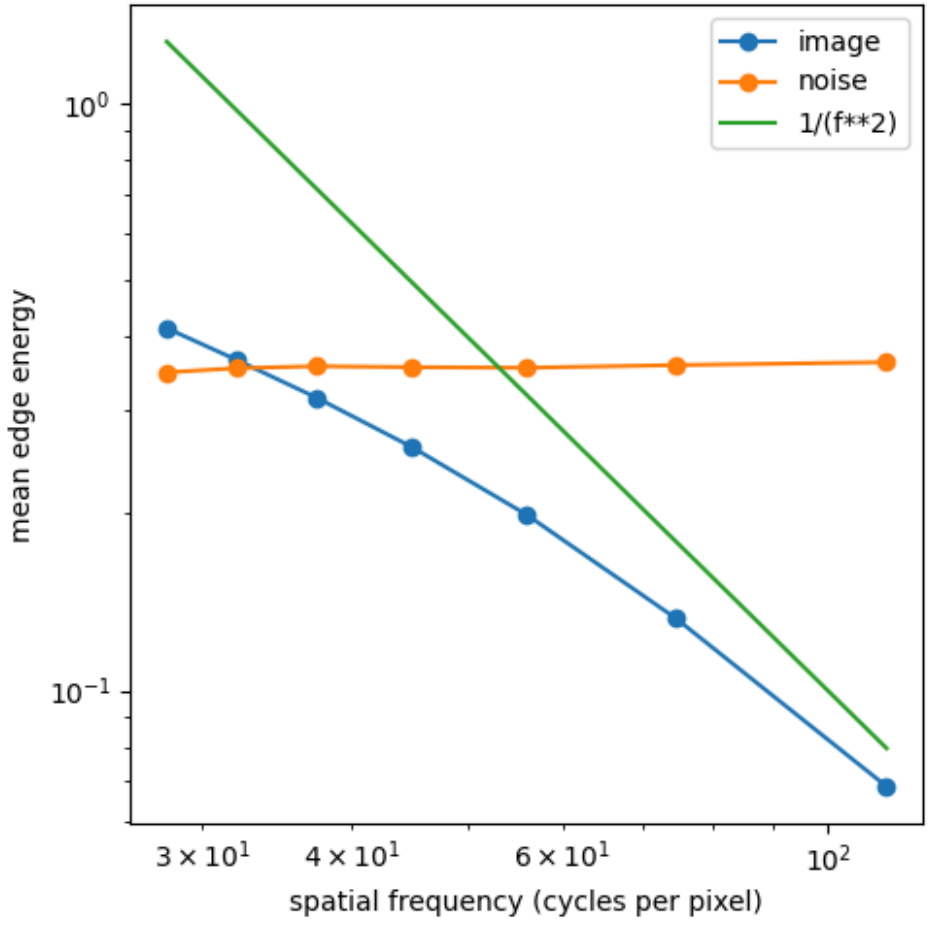
```
    plt.tight_layout()
    plt.show()

# Assuming `pyramid` is your model object and you have a way to get
responses from an image
noise_image = np.random.rand(*image.shape) - 0.5  # Adjusting noise to have
zero mean
noise_pyramid_energy_map, noise_pyramid_energy_orient =
model_response(pyramid, noise_image)

# Create an interactive slider to select frequency and visualize the
response with noise
interact(lambda selected_frequency: response_visualise(
    selected_frequency,
    noise_pyramid_energy_map,
    noise_pyramid_energy_orient,
    noise_image
), selected_frequency=list(noise_pyramid_energy_map.keys()));
```




```
#@title Colapsando o canal de orientação também para obter um único valor
médio de energia de borda por escala espacial
fig,ax = plt.subplots(1, 1, figsize=(5,5))
ax.plot(*zip(*[(this_key, this_ori.mean()) for (this_key, this_ori) in
```

```
pyramid_energy_orient.items()]), 'o-', label='image')
ax.plot(*zip(*[(this_key, this_ori.mean()) for (this_key, this_ori) in
noise_pyramid_energy_orient.items()]), 'o-', label='noise')
f2 = {this_key: 1000*(1/(this_key**2)) for this_key in
frequencies.values()}
ax.plot(*zip(*f2.items()), label='1/(f**2)')
ax.set_ylabel('mean edge energy')
ax.set_xlabel('spatial frequency (cycles per pixel)')
ax.set_xscale('log')
ax.set_yscale('log')
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7e88af346aa0>
```



A imagem de ruído possui um espectro uniforme, como se poderia esperar de um ruído não estruturado. Mas neste gráfico log-log, o espectro da imagem natural decresce de maneira quase linear. Essa é uma observação clássica na análise de

imagens naturais. O espectro de amplitude para a maioria das imagens naturais exibe essa relação do tipo $1/f^2$.

Conclusão

Os filtros Gabor não são apenas um ponto de partida razoável para modelar o V1 (costuma-se dizer que isso pode explicar cerca de 50% da variância em células responsivas do V1), eles também são incrivelmente interpretáveis em comparação com modelos mais contemporâneos de neurociência visual (como as CNNs). Cobriremos como ajustar o modelo aos dados na próxima parte, mas talvez você já perceba que, uma vez obtidas as estimativas de parâmetros para este tipo de modelo, torna-se bastante simples caracterizar a codificação neural em termos de suas dimensões fundamentais - posição, frequência espacial e orientação.

[yammins2014.ipynb]

```
import torch
import torch.nn as nn
import torch.nn.functional as F

# Hierarchical Modular Optimization of Convolutional Networks Achieves
Representations Similar to Macaque IT and Human Ventral Stream
#
https://proceedings.neurips.cc/paper_files/paper/2013/file/9a1756fd0c741126
d7bbd4b692ccbd91-Paper.pdf
```

1. **Camada Simples**:

$$N_\Theta = \text{Pool}_{\theta_p}\left(\text{Normalize}_{\theta_N}\left(\text{Threshold}_{\theta_T}\left(\text{Filter}_{\theta_F}(\text{Input})\right)\right)\right)$$

Onde:

- $(\theta_F, \theta_T, \theta_N, \theta_p)$ são os parâmetros para a filtragem, thresholding, normalização e pooling, respectivamente.

```
class RandomFilter(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, mean=0.0,
std=1.0):
        super(RandomFilter, self).__init__()
        # Randomly initialize filter weights with specified mean and std
        self.weight = nn.Parameter(
            torch.randn(out_channels, in_channels, kernel_size,
kernel_size) * std + mean
        )
        self.weight.requires_grad = False
        self.bias = None

    def forward(self, x):
        padding = (self.weight.shape[2] - 1) // 2
        return F.conv2d(x, self.weight, bias=self.bias, padding=padding)

class ThresholdActivation(nn.Module):
    def __init__(self, threshold=0.0):
        super(ThresholdActivation, self).__init__()
        self.threshold = threshold

    def forward(self, x):
        return F.threshold(x, threshold=self.threshold, value=0.0)

class CustomNormalization(nn.Module):
```

```
    def __init__(self, num_features, eps=1e-5, affine=True):
        super(CustomNormalization, self).__init__()
        self.num_features = num_features
        self.eps = eps
        self.affine = affine
        if self.affine:
            self.gamma = nn.Parameter(torch.ones(1, num_features, 1, 1))
            self.beta = nn.Parameter(torch.zeros(1, num_features, 1, 1))
        else:
            self.register_parameter('gamma', None)
            self.register_parameter('beta', None)


    def forward(self, x):
        # Compute mean and variance across spatial dimensions

        mean = x.mean(dim=(2, 3), keepdim=True)
        var = x.var(dim=(2, 3), keepdim=True, unbiased=False)
        x_hat = (x - mean) / torch.sqrt(var + self.eps)
        if self.affine:
            x_hat = self.gamma * x_hat + self.beta
        return x_hat

class CustomPooling(nn.Module):
    def __init__(self, kernel_size=2, stride=2, pooling_type='max'):
        super(CustomPooling, self).__init__()
        if pooling_type == 'max':
            self.pool = nn.MaxPool2d(kernel_size=kernel_size,
stride=stride)
        elif pooling_type == 'avg':
            self.pool = nn.AvgPool2d(kernel_size=kernel_size,
stride=stride)
        else:
            raise ValueError("pooling_type must be 'max' or 'avg'")

    def forward(self, x):
        return self.pool(x)


# Hierarchical Convolutional layer
class HHCMLayer(nn.Module):
    def __init__(
        self,
        in_channels,
        out_channels,
        kernel_size,
```

```
        θF={'mean': 0.0, 'std': 1.0},
        θT={'threshold': 0.0},
        θN={'eps': 1e-5, 'affine': True},
        θp={'kernel_size': 2, 'stride': 2, 'pooling_type': 'max'},
    ):
        super(HHCMLayer, self).__init__()
        self.filter = RandomFilter(
            in_channels, out_channels, kernel_size, mean=θF['mean'],
std=θF['std']
        )
        self.activation = ThresholdActivation(threshold=θT['threshold'])
        self.normalization = CustomNormalization(
            out_channels, eps=θN['eps'], affine=θN['affine']
        )
        self.pooling = CustomPooling(
            kernel_size=θp['kernel_size'], stride=θp['stride'],
pooling_type=θp['pooling_type']
        )

    def forward(self, x):
        x = self.filter(x)
        x = self.activation(x)
        x = self.normalization(x)
        x = self.pooling(x)
        return x
```

1. **Camada Hierárquica**:

$$N_{\Theta}^{(l)} = \text{Pool}_{\theta_p^{(l)}}\left(\text{Normalize}_{\theta_N^{(l)}}\left(\text{Threshold}_{\theta_T^{(l)}}\left(\text{Filter}_{\theta_F^{(l)}}\left(N_{\Theta}^{(l-1)}\right)\right)\right)\right)$$

Onde:

- l representa a l-ésima camada.

1. **Combinação Heterogênea**:

$$N \boxtimes \boxtimes_{i=1}^{k} N\left(\Theta_{i1}, \Theta_{i2}, \ldots, \Theta_{ini}\right)$$

Onde:

- $\boxtimes$ denota a combinação das saídas das redes ao longo da dimensão convolucional.
- k é o número de componentes combinados.

```
class SingleStackNetwork(nn.Module):
    def __init__(self, layers_params):
        super(SingleStackNetwork, self).__init__()
        layers = []
        for params in layers_params:
            layer = HHCMLayer(
                in_channels=params['in_channels'],
                out_channels=params['out_channels'],
                kernel_size=params['kernel_size'],
                θF=params.get('θF', {'mean': 0.0, 'std': 1.0}),
                θT=params.get('θT', {'threshold': 0.0}),
                θN=params.get('θN', {'eps': 1e-5, 'affine': True}),
                θp=params.get('θp', {'kernel_size': 2, 'stride': 2,
'pooling_type': 'max'}),
            )
            layers.append(layer)
        self.network = nn.Sequential(*layers)

    def forward(self, x):
        return self.network(x)

class CombinedNetwork(nn.Module):
    def __init__(self, networks):
        super(CombinedNetwork, self).__init__()
        self.networks = nn.ModuleList(networks)

    def forward(self, x):
        outputs = [net(x) for net in self.networks]
        return torch.cat(outputs, dim=1)


layers_params_1 = [
    {
        'in_channels': 3,
        'out_channels': 16,
        'kernel_size': 3,
        'θF': {'mean': 0.0, 'std': 1.0},
        'θT': {'threshold': 0.0},
        'θN': {'eps': 1e-5, 'affine': True},
        'θp': {'kernel_size': 2, 'stride': 2, 'pooling_type': 'max'},
    },
    {
        'in_channels': 16,
        'out_channels': 32,
        'kernel_size': 3,
```

```
        'θF': {'mean': 0.0, 'std': 1.0},
        'θT': {'threshold': 0.0},
        'θN': {'eps': 1e-5, 'affine': True},
        'θp': {'kernel_size': 2, 'stride': 2, 'pooling_type': 'max'},
    },
]

layers_params_2 = [
    {
        'in_channels': 3,
        'out_channels': 8,
        'kernel_size': 5,
        'θF': {'mean': 0.0, 'std': 0.5},
        'θT': {'threshold': 0.1},
        'θN': {'eps': 1e-5, 'affine': True},
        'θp': {'kernel_size': 2, 'stride': 2, 'pooling_type': 'avg'},
    },
    {
        'in_channels': 8,
        'out_channels': 16,
        'kernel_size': 5,
        'θF': {'mean': 0.0, 'std': 0.5},
        'θT': {'threshold': 0.1},
        'θN': {'eps': 1e-5, 'affine': True},
        'θp': {'kernel_size': 2, 'stride': 2, 'pooling_type': 'avg'},
    },
]

net1 = SingleStackNetwork(layers_params_1)
net2 = SingleStackNetwork(layers_params_2)
combined_net = CombinedNetwork([net1, net2])

x = torch.randn(1, 3, 64, 64)  # Batch size 1, 3 channels, 64x64 image
output = combined_net(x)
output.shape
```

```
torch.Size([1, 48, 16, 16])
```

Otimização

Hierarchical Modular Optimization (HMO)

O objetivo da **Hierarchical Modular Optimization (HMO)** é encontrar modelos dentro da classe $ N $ que sejam eficazes em modelar respostas neurais para uma

variedade de imagens. Para isso, a estratégia é realizar uma otimização em larga escala (high-throughput) com base em uma tarefa de triagem que representa os desafios do problema de reconhecimento de objetos.

OBJETIVO

Reconhecimento de objetos de forma supervisionada. Foi utilizado um **Maximum Correlation Classifier (MCC)** com validação cruzada 3-fold, onde a função objetivo era a porcentagem de acertos na classificação.

1. Hierarchical Modular Optimization (HMO)

Dado que $ \cal{N} $ é um espaço muito vasto, a HMO busca uma rede multi-stack eficaz, composta por componentes de pilha única especializados em diferentes partes do problema. O procedimento de otimização HMO ocorre em várias etapas:

- **Etapa 1: Otimização** — O espaço de redes de pilha única é otimizado usando o estimador Hyperparameter Tree Parzen, que funciona bem em grandes espaços de parâmetros contínuos e discretos.
- **Etapa 2: Boosting** — As redes otimizadas são vistas como "weak learners", e um algoritmo Adaboost é aplicado para identificar redes cujos padrões de erro sejam complementares.
- **Etapa 3: Combinação** — As redes selecionadas são combinadas para formar uma rede multi-stack $ N_1 $, cujas saídas são avaliadas na tarefa de triagem.
- **Etapa 4: Reponderação por Erros** — A função de pontuação é reponderada com base nos erros do primeiro modelo $ N_1 $, e a otimização é repetida para encontrar redes adicionais $ N_2, N_3, \dots $.

2. Ciclos de Otimização

Esse ciclo de otimização, boosting, combinação e reponderação é repetido $ K $ vezes para obter uma rede final composta $ N = \bigoplus_{i=1}^{K} \bigoplus_{j=1}^{k_i} N_{ij} $. O processo pode ser encerrado ou reiniciado com a rede $ N $ como entrada para outro ciclo de empilhamento.

3. Meta-Parâmetros

Os meta-parâmetros controlam o número de componentes selecionados em cada rodada de boosting $ l_1, l_2, \dots $, o número de repetições $ K $, e quantas vezes o processo de empilhamento $ M $ é realizado. Esses parâmetros foram fixados como $ l_1 = l_2 = 10 $, $ K = 3 $, e $ M \leq 2 $. Com esses parâmetros e o conjunto de

triagem, foi gerada uma rede final $ NHMO $, que produz vetores de características de 1250 dimensões para cada estímulo.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import hyperopt
from hyperopt import fmin, tpe, hp, Trials, STATUS_OK
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
from torch.utils.data import DataLoader, Dataset
from sklearn.model_selection import train_test_split


class HMO:
    def __init__(self, K=3, l=10, M=2, max_evals=50):
        self.K = K  # boosting rounds
        self.l = l  # networks selected per round
        self.M = M  # times the procedure is stacked

        self.max_evals = max_evals  # Max eval para hyperparameter
optimization

    def fit(self, X_train, y_train, X_test, y_test, num_classes):
        self.selected_networks = []
        sample_weights = np.ones(len(y_train)) / len(y_train)
        for m in range(self.M):
            for k in range(self.K):
                # Step 1: Optimization with reweighted scoring
                networks, performances = self.hyperparameter_optimization(
                    X_train, y_train, sample_weights, num_classes
                )
                # Step 2: Boosting to select networks
                selected_nets = self.boosting_selection(
                    networks, performances, X_train, y_train,
sample_weights, num_classes
                )
                self.selected_networks.extend(selected_nets)
                # Step 3: Form the combined network and evaluate
                combined_net = CombinedNetwork(self.selected_networks)
                combined_net.eval()
                with torch.no_grad():
                    train_features =
combined_net(torch.tensor(X_train).float()).numpy()
```

```
                # clf pode mudar
                clf = LogisticRegression(max_iter=1000)
                clf.fit(train_features, y_train,
sample_weight=sample_weights)
                train_preds = clf.predict(train_features)

                # Step 4: Update sample weights
                incorrect = (train_preds != y_train).astype(float)
                sample_weights = incorrect / incorrect.sum()
                print(f"Boosting Round {k+1}/{self.K}, Stack
{m+1}/{self.M}")

        self.combined_net = CombinedNetwork(self.selected_networks)

        # train final classifier with combined features
        self.combined_net.eval()
        with torch.no_grad():
            train_features =
self.combined_net(torch.tensor(X_train).float()).numpy()
            test_features =
self.combined_net(torch.tensor(X_test).float()).numpy()
        self.classifier = LogisticRegression(max_iter=1000)
        self.classifier.fit(train_features, y_train)
        test_preds = self.classifier.predict(test_features)
        accuracy = accuracy_score(y_test, test_preds)
        print(f"Final Test Accuracy: {accuracy * 100:.2f}%")
        return accuracy

    def hyperparameter_optimization(self, X_train, y_train, sample_weights,
num_classes):
        def objective(params):
            layers_params = []
            in_channels = X_train.shape[1]
            for _ in range(params['num_layers']):
                layer_param = {
                    'in_channels': in_channels,
                    'out_channels': int(params['out_channels']),
                    'kernel_size': int(params['kernel_size']),
                    'θF': {'mean': 0.0, 'std': params['θF_std']},
                    'θT': {'threshold': params['θT_threshold']},
                    'θN': {'eps': 1e-5, 'affine': True},
                    'θp': {
                        'kernel_size': int(params['θp_kernel_size']),
                        'stride': int(params['θp_stride']),
                        'pooling_type': params['θp_pooling_type'],
```

```
                    },
                }
                in_channels = int(params['out_channels'])
                layers_params.append(layer_param)

            net = SingleStackNetwork(layers_params)
            net.eval()
            with torch.no_grad():
                features = net(torch.tensor(X_train).float()).numpy()

            features = features.reshape(features.shape[0], -1)
            clf = LogisticRegression(max_iter=1000)
            clf.fit(features, y_train, sample_weight=sample_weights)
            preds = clf.predict(features)


            error = np.sum(sample_weights * (preds !=
y_train).astype(float))
            return {'loss': error, 'status': STATUS_OK, 'net': net}

        # espaço de busca
        space = {
            'num_layers': hp.choice('num_layers', [1, 2, 3]),
            'out_channels': hp.quniform('out_channels', 8, 64, 8),
            'kernel_size': hp.choice('kernel_size', [3, 5, 7]),
            'θF_std': hp.uniform('θF_std', 0.5, 1.5),
            'θT_threshold': hp.uniform('θT_threshold', 0.0, 0.2),
            'θp_kernel_size': hp.choice('θp_kernel_size', [2, 3]),
            'θp_stride': hp.choice('θp_stride', [1, 2]),
            'θp_pooling_type': hp.choice('θp_pooling_type', ['max',
'avg']),
        }
        trials = Trials()
        best = fmin(
            fn=objective,
            space=space,
            algo=tpe.suggest,
            max_evals=self.max_evals,
            trials=trials,
            # rstate=np.random.RandomState(42),
        )
        networks = [trial['result']['net'] for trial in trials.trials]
        performances = [trial['result']['loss'] for trial in trials.trials]
        return networks, performances

    def boosting_selection(self, networks, performances, X_train, y_train,
```

```
sample_weights, num_classes):
        # Initialize variables
        selected_networks = []
        n_samples = len(y_train)
        weights = sample_weights.copy()


        for _ in range(self.l):
            min_error = float('inf')
            best_net = None

            # precorre ate achar rede de menor erro
            for net, perf in zip(networks, performances):
                net.eval()
                with torch.no_grad():
                    features = net(torch.tensor(X_train).float()).numpy()
                features = features.reshape(features.shape[0], -1)
                clf = LogisticRegression(max_iter=1000)
                clf.fit(features, y_train, sample_weight=weights)
                preds = clf.predict(features)
                error = np.sum(weights * (preds != y_train).astype(float))

                if error < min_error:
                    min_error = error
                    best_net = net
                    best_preds = preds


            incorrect = (best_preds != y_train).astype(float)
            weights = weights * incorrect
            if weights.sum() == 0:
                weights = np.ones(n_samples) / n_samples
            else:
                weights /= weights.sum()
            selected_networks.append(best_net)


            print(f"Erro de rede selecionada: {min_error}")
        return selected_networks

    def predict(self, X):
        self.combined_net.eval()
        with torch.no_grad():
            features = self.combined_net(torch.tensor(X).float()).numpy()
        preds = self.classifier.predict(features)
```

```
        return preds


  from torchvision import datasets, transforms

  transform = transforms.Compose([transforms.ToTensor()])
  train_dataset = datasets.CIFAR10(root='./data', train=True,
download=True, transform=transform)
  test_dataset = datasets.CIFAR10(root='./data', train=False,
download=True, transform=transform)


  n_samples = 1000  # Use 1000 samples for training and testing
  X_train, y_train = zip(*[(x.numpy(), y) for x, y in
resample(train_dataset, n_samples=n_samples)])
  X_test, y_test = zip(*[(x.numpy(), y) for x, y in resample(test_dataset,
n_samples=n_samples)])
  X_train = np.array(X_train).squeeze()
  y_train = np.array(y_train)
  X_test = np.array(X_test).squeeze()
  y_test = np.array(y_test)
  num_classes = 10

  X_train = X_train.transpose(0, 2, 3, 1)  # (N, H, W, C)
  X_test = X_test.transpose(0, 2, 3, 1)

Files already downloaded and verified
Files already downloaded and verified

hmo = HMO(K=1, l=1, M=1, max_evals=1)
hmo.fit(X_train, y_train, X_test, y_test, num_classes)

preds = hmo.predict(X_test)
test_accuracy = accuracy_score(y_test, preds)

100%|██████████| 1/1 [00:01<00:00,  1.02s/trial, best loss:
0.45600000000000007]
Erro de rede selecionada: 0.45600000000000007
```

APÊNDICE 4

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 30 de out. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

A entrega desse GATE foi centrada nos estudos dos princípios, exploração de trabalhos e testes.

Em relação aos estudos:

Terminei meus estudos de Neurofisiologia básica (primeira metade dessa playlist):
https://www.youtube.com/playlist?list=PLqgZEQsU_8E0I1P9bKR6yKOKPMpoJ_tLR

Li um artigo recente abordando a intersecção da IA e neurociência:
https://www.cell.com/cell/fulltext/S0092-8674(24)00980-2

Assisti outra palestra com pesquisadores sobre o tema de minha pesquisa e aplicações possíveis:
CBMM10 Panel: Neuroscience to AI and back again

Assisti a apresentação de uma das pesquisadoras abordando o tema em uma aplicação específica de estudo:
Functional Specificity in the Human Brain: What, Whether, and Why?

Conheci o trabalho de outros pesquisadores do grupo, pos docs, principalmente na área de audio:
https://www.cell.com/neuron/fulltext/S0896-6273(18)30250-2
https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.3002366

Já em relação aos testes:
Investiguei as arquiteturas dos ganhadores BrainScore 2024, e testei dentro do framework:

Brain-Score Benchmarking - CCN 2024

Explorei uma biblioteca para visualização de dados neurais:
https://github.com/MouseLand/rastermap

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

Durante a semana, concentrei boa parte de meu tempo estudando fundamentos e investigando possíveis abordagens para minha pesquisa. Notei que dentre as arquiteturas mais recentes em visão, nenhuma aborda sobre a arquitetura YOLO, apesar de ser popular, e a área de aprendizado não supervisionado ainda não parece tão popular. Pretendo realizar, portanto, testes dentro dessas possíveis abordagens.

Notei, também, que a área de aúdio ainda parece mal explorada, e foca mais em arquiteturas convolucionais não tão conhecidas, pretendo investigar o porquê disso.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

LEONARDO ALVES: Em análise!

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 31 de out. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

No GATE anterior, foram definidos experimentos e pesquisas para direcionar novas aplicações que aumentem a similaridade de modelos com o funcionamento do cérebro. A partir disso, busquei responder algumas questões:

1. Por que o YOLO não é um modelo amplamente utilizado neste contexto?
O YOLO não é amplamente empregado porque é de uma API diferente do Pytorch e é supervisionado. Além disso, abordagens que combinam convoluções e Transformers demonstram resultados promissores:
- Hugging Face - CVT
- Brain-Score - Vision Model

Exemplo do uso de um modelo no framework:
test.ipynb

Observei também, que o uso de framework é limitado pelo poder computacional. Realizei tentativas de rodar modelos com menor gasto de memória mas fui mal sucedido, ainda.

2. Qual é o cenário de abordagem não supervisionada?
A abordagem não supervisionada é uma área em crescimento, com algumas implementações voltadas para emular o sistema visual. Um exemplo é o projeto disponível em:
-unsup_vvs

3. Por que arquiteturas convolucionais são amplamente utilizadas em tarefas de áudio?
O uso de arquiteturas convolucionais em áudio segue princípios básicos de circuitos cerebrais e imita o processamento hierárquico observado no sistema visual. Confira um exemplo:
- auditory_brain_dnn

Além disso, disponibilizo anotações dos estudos e pontos pendentes da entrega passada:
- Anotações e Pendências

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega, planejo resolver os problemas técnicos enfrentados neste GATE e apresentar as informações de forma mais acessível e organizada.

As principais ações incluem:

1. Solucionar os problemas técnicos encontrados no framework;
2. Disponibilizar os conceitos estudados em um formato mais acessível, possivelmente na forma de um artigo, para facilitar a compreensão e a consulta;
3. Iniciar experimentos com abordagens não supervisionadas para a classificação de áudio, explorando arquiteturas pouco utilizadas, como o Conformer.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[test.ipynb]

Install

```
!git clone https://github.com/brain-score/vision.git

cd vision

!python -m pip install --upgrade pip .
!python -m pip install -e .
```

Test a model

```
from brainscore_vision.model_helpers.activations.core import Defaults


Defaults.batch_size = 16

from brainscore_vision import score

model_score =
score(model_identifier='cvt_cvt-w24-384-in22k_finetuned-in1k_4',
benchmark_identifier='MajajHong2015public.IT-pls')
print(model_score)
```

Via terminal

```
!python brainscore_vision score
--model_identifier='cvt_cvt-w24-384-in22k_finetuned-in1k_4'
--benchmark_identifier='MajajHong2015public.IT-pls'
```

[Anotações estudos sobre Visão]

Percepção Visual

→ Recepção da Luz na Retina

1 Estrutura do Olho

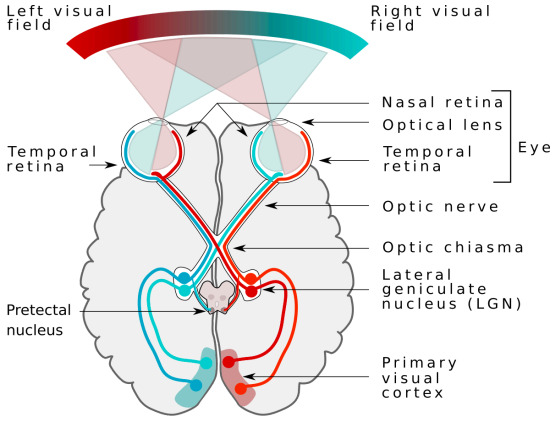
O olho humano funciona como uma câmera fotográfica sofisticada. A luz entra pela córnea, passa pela pupila (cujo tamanho é regulado pela íris) e é focalizada pela lente (cristalino) na retina, localizada no fundo do olho.

2 A Retina e os Fotorreceptores

A retina é composta por vários tipos de células, mas as principais responsáveis pela detecção da luz são os fotorreceptores: os cones e os bastonetes.

- **Cones**: Responsáveis pela visão em cores e pela acuidade visual. Funcionam melhor em condições de iluminação intensa e estão concentrados na fóvea, a região central da retina.
- **Bastonetes**: Sensíveis à luz fraca, permitem a visão em condições de pouca iluminação, mas não distinguem cores. Estão distribuídos principalmente na periferia da retina.

3 Transdução de Sinal



Quando a luz atinge os fotorreceptores, provoca uma mudança na conformação de moléculas fotossensíveis (como a rodopsina nos bastonetes), iniciando uma cascata bioquímica que resulta na hiperpolarização da célula. Esse processo transforma o sinal luminoso em um impulso elétrico.

→ Processamento Inicial na Retina

Após a transdução, o sinal elétrico é modulado por outras células da retina:

- **Células Bipolares**: Recebem sinais dos fotorreceptores e transmitem para as células ganglionares.
- **Células Horizontais e Amácrinas**: Modulam a atividade entre fotorreceptores e células bipolares, contribuindo para o contraste e detecção de movimento.
- **Células Ganglionares**: Seus axônios formam o nervo óptico, que leva a informação visual ao cérebro.

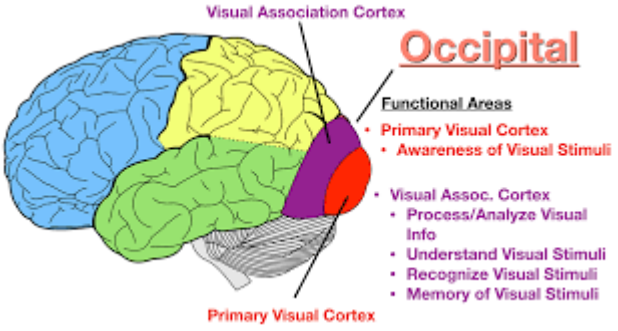
As células ganglionares também têm campos receptivos organizados em centros e periferias (on-center/off-surround e vice-versa), o que é crucial para a detecção de contrastes e bordas.

→ Transmissão para o Cérebro

Os axônios das células ganglionares convergem para formar o nervo óptico. No quiasma óptico, ocorre um cruzamento parcial das fibras nervosas: as informações do campo visual esquerdo de ambos os olhos são processadas no hemisfério direito e vice-versa.

Após o quiasma, os sinais viajam pelo trato óptico até o **Corpo Geniculado Lateral (CGL)** no tálamo, onde ocorre uma primeira integração significativa dos sinais visuais. Dali, as projeções seguem para o **Córtex Visual Primário (V1)**, localizado no lobo occipital.

→ Processamento no Cérebro



1 Córtex Visual Primário (V1)

No V1, as informações visuais são processadas em termos de características básicas, como orientação, frequência espacial, cor e movimento. Neurônios nessa área respondem a

estímulos específicos, como linhas em determinada orientação ou movimento em certa direção.

2 Vias Visuais

- **Via Ventral ("O quê")**: Estende-se para o lobo temporal e é responsável pelo reconhecimento de objetos e cores.
- **Via Dorsal ("Onde/Como")**: Estende-se para o lobo parietal e é responsável pela percepção de movimento e localização espacial.

→ Experimentos de Hubel e Wiesel

Descobertas Principais

David Hubel e Torsten Wiesel, nos anos 1960, realizaram uma série de experimentos que elucidaram como os neurônios do córtex visual respondem a estímulos visuais.

- **Neurônios Simples**: Descobriram que certos neurônios no V1 respondem a barras ou bordas com orientação específica.
- **Neurônios Complexos**: Identificaram neurônios que respondem a estímulos com orientação específica em movimento.
- **Organização em Colunas**: Demonstraram que neurônios com preferências similares de orientação estão organizados em colunas perpendiculares à superfície cortical.

Usando eletrodos para registrar a atividade elétrica de neurônios individuais em gatos e macacos anestesiados, apresentaram diferentes estímulos visuais (como pontos de luz, linhas e padrões em movimento) e observaram as respostas neuronais.

As descobertas de Hubel e Wiesel foram revolucionárias por:

- **Revelar a especificidade neuronal**: Mostraram que neurônios individuais são especializados em processar características específicas do estímulo visual.
- **Contribuir para o entendimento da plasticidade cerebral**: Seus trabalhos também exploraram como a privação visual em períodos críticos do desenvolvimento pode afetar permanentemente a organização do córtex visual.
- **Fundamentar modelos de processamento visual**: Suas descobertas formaram a base para teorias sobre como o cérebro constrói a percepção visual a partir de elementos simples.

Em reconhecimento ao seu trabalho pioneiro, receberam o Prêmio Nobel de Fisiologia ou Medicina em 1981.

A percepção visual é resultado de um complexo processamento que se inicia com a recepção da luz na retina e culmina na interpretação sofisticada pelo cérebro. Os estudos de Hubel e Wiesel aprofundaram significativamente nossa compreensão sobre como o córtex visual processa informações, demonstrando a existência de neurônios altamente especializados e a importância da organização cortical. Esses conhecimentos não apenas avançaram a neurociência básica, mas também têm implicações em áreas como a inteligência artificial, neurocirurgia e reabilitação visual.

APÊNDICE 5

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 7 de out. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

As atividades desse GATE foram centradas na exploração de arquiteturas como o Conformer, da Nvidia.

Fiz uma coleção de modelos asr da nvidia estilo conformer
NvdiaASR.ipynb

Função que capta as funções de ativações das camadas da rede:
hook_test.ipynb

Baixei um dataset e fiz um pré-processamento para utilizar nos experimentos e comparar com artigos selecionados que o utiliza:
neural_data.ipynb

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Prosseguir o desenvolvimentos das atividades mencionadas e terminar a comparação desses modelos.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[neural_data.ipynb]

Install

```
%%capture
!wget https://mcdermottlab.mit.edu//tuckute_feather_2023/data.tar
!tar -xvf /content/data.tar
```

EDA

wavs

```
import os

data_path = '/content/data'
os.listdir(data_path)
```

```
['stimuli', 'neural', 'package_anatomical_roi_labels.py']
```

```
import pickle

with open('/content/data/stimuli/df_stimuli_meta.pkl', 'rb') as file:
    data = pickle.load(file)

data.head()
```

	category_label
stim5_alarm_clock	Mechanical
stim7_applause	HumNonVoc
stim11_baby_crying	HumVoc
stim12_background_speech	EngSpeech
stim18_basketball_dribbling	EnvSound

```
from IPython import display
sounds_path = os.path.join(data_path , 'stimuli/165_natural_sounds_16kHz')
wav_files = [os.path.join(sounds_path, f) for f in os.listdir(sounds_path)
```

```
if f.endswith('.wav')]

i=0
sample = wav_files[i]
display.Audio(sample)
```

```
<IPython.lib.display.Audio object>
```

```
label = sample.split('.wav')[0]
data.loc[label]
```

```
category_label    Mechanical
Name: stim527_ringtone, dtype: object
```

Neural

NH2015

Norman-Haignere, Sam et al.

```
import pickle

with open('/content/data/neural/B2021/df_roi_meta.pkl', 'rb') as file:
    neural_data = pickle.load(file)

neural_data.head()
```

subj_idx	hemi	x_ras	y_ras	voxel_id	kell_r_reliability	pearson_r_reliability	voxel_variability_mean_reps	voxel_variability_std
0	1	rh	-20	-73	0	0.344148	0.178875	0.287731
1	1	rh	-18	-75	1	0.348526	0.168776	0.354286
2	1	rh	-18	-73	2	0.338838	0.207209	0.334681
3	1	rh	-16	-75	3	0.327738	0.157635	0.418364
4	1	rh	-6	-43	4	0.353526	0.078773	0.394613

...

coord_id	shared_by	roi_label_fname	roi_label_specific	roi_label_general	roi_anat_hemi
0.392985	-20_-73	1.0	NaN	NaN	NaN
0.526748	-18_-75	1.0	NaN	NaN	NaN
0.480094	-18_-73	1.0	NaN	NaN	NaN
0.632466	-16_-75	2.0	NaN	NaN	NaN
0.622458	-6_-43	2.0	NaN	NaN	NaN

neural_data

	subj_idx	hemi	x_ras	y_ras	voxel_id	kell_r_reliability	pearson_r_reliability	voxel_variability_mean_reps	voxel_variability_std
0	1	rh	-20	-73	0	0.344148	0.178875	0.287731	0.392985
1	1	rh	-18	-75	1	0.348526	0.168776	0.354286	0.526748
2	1	rh	-18	-73	2	0.338838	0.207209	0.334681	0.480094
3	1	rh	-16	-75	3	0.327738	0.157635	0.418364	0.632466
4	1	rh	-6	-43	4	0.353526	0.078773	0.394613	0.622458
...	...	...	...	...	...	...	...	...	...
26787	20	lh	91	-12	26787	0.332566	0.155260	0.303427	0.433818
26788	20	lh	91	-10	26788	0.415720	0.277475	0.317263	0.438018
26789	20	lh	91	-8	26789	0.434968	0.339678	0.323597	0.452083
26790	20	lh	93	-12	26790	0.333928	0.172713	0.309483	0.415619
26791	20	lh	93	-10	26791	0.365284	0.256224	0.307256	0.423518

...

coord_id	shared_by	roi_label_fname	roi_label_specific	roi_label_general	roi_anat_hemi
-20_-73	1.0	NaN	NaN	NaN	NaN
-18_-75	1.0	NaN	NaN	NaN	NaN
-18_-73	1.0	NaN	NaN	NaN	NaN
-16_-75	2.0	NaN	NaN	NaN	NaN
-6_-43	2.0	NaN	NaN	NaN	NaN
...	...	...	...	...	...
91_-12	6.0	NaN	NaN	NaN	NaN
91_-10	7.0	NaN	NaN	NaN	NaN
91_-8	4.0	NaN	NaN	NaN	NaN
93_-12	3.0	NaN	NaN	NaN	NaN
93_-10	3.0	NaN	NaN	NaN	NaN

B2021

Boebinger, Dana et al.

```
import pickle

with open('/content/data/neural/B2021/df_roi_meta.pkl', 'rb') as file:
    neural_data = pickle.load(file)

neural_data.head()
```

subj_idx	hemi	x_ras	y_ras	voxel_id	kell_r_reliability	pearson_r_reliability	voxel_variability_mean_reps	voxel_variability_std
0	1	rh	-20	-73	0	0.344148	0.178875	0.287731
1	1	rh	-18	-75	1	0.348526	0.168776	0.354286
2	1	rh	-18	-73	2	0.338838	0.207209	0.334681
3	1	rh	-16	-75	3	0.327738	0.157635	0.418364
4	1	rh	-6	-43	4	0.353526	0.078773	0.394613

...

coord_id	shared_by	roi_label_fname	roi_label_specific	roi_label_general	roi_anat_hemi
0.392985	-20_-73	1.0	NaN	NaN	NaN
0.526748	-18_-75	1.0	NaN	NaN	NaN
0.480094	-18_-73	1.0	NaN	NaN	NaN
0.632466	-16_-75	2.0	NaN	NaN	NaN
0.622458	-6_-43	2.0	NaN	NaN	NaN

```
import scipy.io as sio
mat_file_path = '/content/data/neural/B2021/stim_info_v4.mat'
mat_data = sio.loadmat(mat_file_path)
mat_data
```

```
{'__header__': b'MATLAB 5.0 MAT-file, Platform: GLNXA64, Created on: Thu
Oct 24 19:08:08 2019',
 '__version__': '1.0',
 '__globals__': [],
 'stim_info': array([[(array([[array(['EngSpeech'], dtype='<U9')],
                [array(['ForSpeech'], dtype='<U9')],
                [array(['HumVoc'], dtype='<U6')],
                [array(['HumNonVoc'], dtype='<U9')],
                [array(['AniVoc'], dtype='<U6')],
                [array(['AniNonVoc'], dtype='<U9')],
```

```
                [array(['Nature'], dtype='<U6')],
                [array(['Mechanical'], dtype='<U10')],
                [array(['Song'], dtype='<U4')],
                [array(['Music'], dtype='<U5')],
                [array(['EnvSound'], dtype='<U8')],
                [array(['Drum'], dtype='<U4')],
                [array(['ForMusic'], dtype='<U8')],
                [array(['ForSong'], dtype='<U7')]], dtype=object), array([[
8],..., [14]], dtype=uint8), array([[0.  , 0.  , 0.  , ..., 0.  , 0.  , 0.
],

                ...,
                [0.02, 0.  , 0.  , ..., 0.  , 0.  , 0.52]]), array([[  5],
                [  7],
                ...,
                [927]], dtype=uint16), array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 1, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 1, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 1]], dtype=uint8), array([[0.    , 0.4
, 0.112 ],
                [0.2198, 0.7   , 0.21  ],
                [0.375 , 0.    , 0.75  ],
                [0.7   , 0.    , 0.    ],
                [0.592 , 0.4   , 1.    ],
                [1.    , 0.5   , 0.95  ],
                [1.    , 0.96  , 0.    ],
                [0.9   , 0.54  , 0.    ],
                [0.2   , 0.76  , 1.    ],
                [0.    , 0.06  , 0.6   ],
                [0.5   , 0.5   , 0.5   ],
                [0.6   , 0.976 , 1.    ],
                [0.    , 0.25  , 1.    ],
                [0.    , 0.52  , 1.    ]]), array([[10, 13, 12,  9, 14,  1,
2,  3,  5,  4,  6,  7,  8, 11]],
               dtype=uint8), array([[array(['stim5_alarm_clock'],
dtype='<U17')],
                [array(['stim7_applause'], dtype='<U14')],
                  …
                [array(['stim926_acapella'], dtype='<U16')],
                [array(['stim927_chorus3'], dtype='<U15')]], dtype=object))
]],
       dtype=[('category_labels', 'O'), ('category_assignments', 'O'),
```

```
('continuous_scores', 'O'), ('ids', 'O'), ('category_regressors', 'O'),
('colors', 'O'), ('plotting_order', 'O'), ('stim_names', 'O')])}
```

[hook_test.ipynb]

PyTorch

```
from PIL import Image
import torch
from torchvision.models import resnet18
from torchvision import transforms as T

device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')

model = resnet18(pretrained=True)
model.to(device)

ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2),
bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
```

```
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)

activation = {}
def get_activation(name):
    def hook(model, input, output):
      activation[name] = output.detach()
    return hook

model.layer3[0].conv1.register_forward_hook(get_activation('layer3'))
```

```
<torch.utils.hooks.RemovableHandle at 0x7bff8aaf58a0>
```

```
X = torch.randn(1, 3, 224, 224).to(device)
out = model(X)

activation['layer3'].shape
```

```
torch.Size([1, 256, 14, 14])
```

```
model.conv1.register_forward_hook(get_activation('conv1'))
```

```
<torch.utils.hooks.RemovableHandle at 0x7bff8aeaf310>
```

```
X = torch.randn(1, 3, 224, 224).to(device)
out = model(X)

activation['conv1'].shape
```

```
torch.Size([1, 64, 112, 112])
```

```
activation
```

```
{'layer3': tensor([[[[ 0.1246, -0.3928, -0.5942,  ..., -0.5392, -0.4535,
0.2980],
          [ 0.1499, -0.4062, -0.5364,  ..., -0.2857, -0.3547,  0.3066],
           ...,
          [-0.2416, -1.1494, -0.7769,  ...,  0.5568, -0.6813, -0.6782],
          [ 0.2016, -0.6411, -1.0342,  ..., -0.7255, -0.8721, -0.4021],
          [ 0.5816, -0.5737, -0.2668,  ..., -1.0331, -1.4976, -0.2959]]]],
       device='cuda:0'),
 'conv1': tensor([[[[-5.6076e-01, -1.5122e+00, -4.2310e-01,  ...,
5.7826e-01,
           5.7518e-01, -1.0200e+00],
          [-8.3768e-01,  2.8897e+00,  1.2708e+00,  ..., -4.1427e+00,
           -2.6348e+00, -1.5601e+00],...,
          [ 3.8215e-07, -5.6201e-07, -5.3995e-07,  ..., -6.1194e-09,
           -1.5378e-06, -7.9621e-07]]]], device='cuda:0')}
```

NeMo

```
%%capture
!pip install wget
!apt-get install sox libsndfile1 ffmpeg
!pip install text-unidecode

# ## Install NeMo
BRANCH = 'main'
!python -m pip install
git+https://github.com/NVIDIA/NeMo.git@$BRANCH#egg=nemo_toolkit[asr]

## Install TorchAudio
!pip install torchaudio -f
https://download.pytorch.org/whl/torch_stable.html

from nemo.collections.asr.models import EncDecMultiTaskModel

model = EncDecMultiTaskModel.from_pretrained('nvidia/canary-1b')
```

```
[NeMo I 2024-11-08 04:07:22 mixins:200] _setup_tokenizer: detected an
aggregate tokenizer
[NeMo I 2024-11-08 04:07:22 mixins:339] Tokenizer SentencePieceTokenizer
initialized with 32 tokens
[NeMo I 2024-11-08 04:07:22 mixins:339] Tokenizer SentencePieceTokenizer
initialized with 1024 tokens
[NeMo I 2024-11-08 04:07:22 mixins:339] Tokenizer SentencePieceTokenizer
initialized with 1024 tokens
[NeMo I 2024-11-08 04:07:22 mixins:339] Tokenizer SentencePieceTokenizer
initialized with 1024 tokens
[NeMo I 2024-11-08 04:07:22 mixins:339] Tokenizer SentencePieceTokenizer
initialized with 1024 tokens
[NeMo I 2024-11-08 04:07:22 aggregate_tokenizer:73] Aggregate vocab size:
4128

[NeMo W 2024-11-08 04:07:23 modelPT:176] If you intend to do training or
fine-tuning, please call the ModelPT.setup_training_data() method and
provide a valid configuration file to setup the train data loader.
    Train config :
    tarred_audio_filepaths: null
    manifest_filepath: null
    sample_rate: 16000
    shuffle: true
    batch_size: null
    num_workers: 8
    use_lhotse: true
    max_duration: 40
```

```
    pin_memory: true
    use_bucketing: false
    bucket_duration_bins: null
    num_buckets: 1
    text_field: answer
    lang_field: target_lang
    batch_duration: 360
    quadratic_duration: 15
    bucket_buffer_size: 20000
    shuffle_buffer_size: 10000

[NeMo W 2024-11-08 04:07:23 modelPT:183] If you intend to do validation,
please call the ModelPT.setup_validation_data() or
ModelPT.setup_multiple_validation_data() method and provide a valid
configuration file to setup the validation data loader(s).
    Validation config :
    manifest_filepath: null
    sample_rate: 16000
    batch_size: 8
    shuffle: false
    num_workers: 0
    pin_memory: true
    tarred_audio_filepaths: null
    use_lhotse: true
    text_field: answer
    lang_field: target_lang
    use_bucketing: false

[NeMo W 2024-11-08 04:07:23 modelPT:189] Please call the
ModelPT.setup_test_data() or ModelPT.setup_multiple_test_data() method and
provide a valid configuration file to setup the test data loader(s).
    Test config :
    manifest_filepath: null
    sample_rate: 16000
    batch_size: 32
    shuffle: false
    num_workers: 0
    pin_memory: true
    tarred_audio_filepaths: null
    use_lhotse: true
    text_field: answer
    lang_field: target_lang
    use_bucketing: false


[NeMo I 2024-11-08 04:07:23 features:305] PADDING: 0
[NeMo I 2024-11-08 04:07:54 save_restore_connector:275] Model
```

```
EncDecMultiTaskModel was successfully restored from
/root/.cache/huggingface/hub/models--nvidia--canary-1b/snapshots/dd32c0c709
e2bfc79f583e16b9df4b3a160f7e86/canary-1b.nemo.

# # update dcode params
# decode_cfg = model.cfg.decoding
# decode_cfg.beam.beam_size = 1
# model.change_decoding_strategy(decode_cfg)

model

EncDecMultiTaskModel(
  (preprocessor): AudioToMelSpectrogramPreprocessor(
    (featurizer): FilterbankFeatures()
  )
  (encoder): ConformerEncoder(
    (pre_encode): ConvSubsampling(
      (out): Linear(in_features=4096, out_features=1024, bias=True)
      (conv): Sequential(
        (0): Conv2d(1, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (4): ReLU(inplace=True)
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (6): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (7): ReLU(inplace=True)
      )
    )
    (pos_enc): RelPositionalEncoding(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (layers): ModuleList(
      (0-23): 24 x ConformerLayer(
        (norm_feed_forward1): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward1): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (norm_conv): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (conv): ConformerConvolution(
```

```
          (pointwise_conv1): Conv1d(1024, 2048, kernel_size=(1,),
stride=(1,))
          (depthwise_conv): CausalConv1D(1024, 1024, kernel_size=(9,),
stride=(1,), groups=1024)
          (batch_norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
          (activation): Swish()
          (pointwise_conv2): Conv1d(1024, 1024, kernel_size=(1,),
stride=(1,))
        )
        (norm_self_att): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (self_attn): RelPositionMultiHeadAttention(
          (linear_q): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_k): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_v): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_out): Linear(in_features=1024, out_features=1024,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear_pos): Linear(in_features=1024, out_features=1024,
bias=False)
        )
        (norm_feed_forward2): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward2): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (norm_out): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
  (encoder_decoder_proj): Identity()
  (transf_decoder): TransformerDecoderNM(
    (_embedding): TransformerEmbedding(
      (token_embedding): Embedding(4128, 1024, padding_idx=0)
      (position_embedding): FixedPositionalEncoding()
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
```

```
    (_decoder): TransformerDecoder(
      (final_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
      (layers): ModuleList(
        (0-23): 24 x TransformerDecoderBlock(
          (layer_norm_1): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
          (first_sub_layer): MultiHeadAttention(
            (query_net): Linear(in_features=1024, out_features=1024,
bias=True)
            (key_net): Linear(in_features=1024, out_features=1024,
bias=True)
            (value_net): Linear(in_features=1024, out_features=1024,
bias=True)
            (out_projection): Linear(in_features=1024, out_features=1024,
bias=True)
            (attn_dropout): Dropout(p=0.1, inplace=False)
            (layer_dropout): Dropout(p=0.1, inplace=False)
          )
          (layer_norm_2): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
          (second_sub_layer): MultiHeadAttention(
            (query_net): Linear(in_features=1024, out_features=1024,
bias=True)
            (key_net): Linear(in_features=1024, out_features=1024,
bias=True)
            (value_net): Linear(in_features=1024, out_features=1024,
bias=True)
            (out_projection): Linear(in_features=1024, out_features=1024,
bias=True)
            (attn_dropout): Dropout(p=0.1, inplace=False)
            (layer_dropout): Dropout(p=0.1, inplace=False)
          )
          (layer_norm_3): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
          (third_sub_layer): PositionWiseFF(
            (dense_in): Linear(in_features=1024, out_features=4096,
bias=True)
            (dense_out): Linear(in_features=4096, out_features=1024,
bias=True)
            (layer_dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
```

```
  (log_softmax): TokenClassifier(
    (dropout): Dropout(p=0.0, inplace=False)
    (mlp): MultiLayerPerceptron(
      (layer0): Linear(in_features=1024, out_features=4128, bias=True)
    )
  )
  (loss): SmoothedCrossEntropyLoss()
  (spec_augmentation): SpectrogramAugmentation(
    (spec_augment): SpecAugment()
  )
  (val_loss): GlobalAverageLossMetric()
  (wer): WER()
  (bleu): BLEU()
)
```

```
activation = {}
def get_activation(name):
    def hook(model, input, output):
      activation[name] = output
    return hook

model.encoder.pos_enc.register_forward_hook(get_activation('encoder'))

model.transcribe('/content/download.wav')
```

```
[NeMo W 2024-11-08 04:14:21 dataloader:206] You are using a non-tarred
dataset and requested tokenization during data sampling (pretokenize=True).
This will cause the tokenization to happen in the main (GPU) process,
possibly impacting the training speed if your tokenizer is very large. If
the impact is noticable, set pretokenize=False in dataloader config. (note:
that will disable token-per-second filtering and 2D bucketing features)
Transcribing: 1it [00:00,  2.21it/s]

['']
```

```
activation['encoder'][0]
```

```
tensor([[[ 0.3048,  0.0901,  0.2628,  ..., -0.2037,  0.1171, -0.1511],
         [ 0.7563,  0.1873,  0.2673,  ..., -0.1476, -0.0885,  0.3239],
         [-0.2568, -0.3061, -0.6333,  ..., -0.2973, -0.0297, -0.0792],
         [-0.1511, -0.4235,  0.2578,  ...,  0.2666, -0.0660, -0.3310],
         [ 0.3596,  0.1087,  0.5955,  ...,  0.3024,  0.1373,  0.4369]]],
       device='cuda:0')
```

[NvdiaASR.ipynb]

Install

```
%%capture
!pip install wget
!apt-get install sox libsndfile1 ffmpeg
!pip install text-unidecode
```

```
# ## Install NeMo
  BRANCH = 'main'
  !python -m pip install
git+https://github.com/NVIDIA/NeMo.git@$BRANCH#egg=nemo_toolkit[asr]

## Install TorchAudio
!pip install torchaudio -f
https://download.pytorch.org/whl/torch_stable.html

!wget https://dldata-public.s3.us-east-2.amazonaws.com/2086-149220-0033.wav
```

Disponível em :

https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/asr/models.html

Catálogo de modelos

```
from IPython.display import Audio
Audio('/content/2086-149220-0033.wav')
```

```
<IPython.lib.display.Audio object>
```

Canary 1b

```
from nemo.collections.asr.models import EncDecMultiTaskModel

model = EncDecMultiTaskModel.from_pretrained('nvidia/canary-1b')
model
```

Parakeet

Parakeet CTC 0.6B (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="nvidia/parakeet-ctc-0.6b")
model
```

```
{"model_id":"9a62be7744434b3b8288f663fc8dcd1d","version_major":2,"version_minor":0}
```

```
[NeMo I 2024-11-25 17:47:05 mixins:176] Tokenizer SentencePieceTokenizer
initialized with 1024 tokens
```

```
[NeMo W 2024-11-25 17:47:06 modelPT:176] If you intend to do training or
fine-tuning, please call the ModelPT.setup_training_data() method and
provide a valid configuration file to setup the train data loader.
    Train config :
    manifest_filepath:
/disk1/NVIDIA/datasets/LibriSpeech_NeMo/librivox-train-all.json
    sample_rate: 16000
    batch_size: 16
    shuffle: true
    num_workers: 8
    pin_memory: true
    use_start_end_token: false
    trim_silence: false
    max_duration: 16.7
    min_duration: 0.1
    is_tarred: false
    tarred_audio_filepaths: null
    shuffle_n: 2048
    bucketing_strategy: fully_randomized
    bucketing_batch_size: null

[NeMo W 2024-11-25 17:47:06 modelPT:183] If you intend to do validation,
please call the ModelPT.setup_validation_data() or
ModelPT.setup_multiple_validation_data() method and provide a valid
configuration file to setup the validation data loader(s).
    Validation config :
    manifest_filepath:
/disk1/NVIDIA/datasets/LibriSpeech_NeMo/librivox-dev-clean.json
    sample_rate: 16000
    batch_size: 16
    shuffle: false
    use_start_end_token: false
    num_workers: 8
    pin_memory: true

[NeMo W 2024-11-25 17:47:06 modelPT:189] Please call the
ModelPT.setup_test_data() or ModelPT.setup_multiple_test_data() method and
provide a valid configuration file to setup the test data loader(s).
    Test config :
    manifest_filepath: null
    sample_rate: 16000
    batch_size: 16
    shuffle: false
    use_start_end_token: false
    num_workers: 8
```

```
    pin_memory: true


[NeMo I 2024-11-25 17:47:06 features:305] PADDING: 0
[NeMo I 2024-11-25 17:47:29 save_restore_connector:275] Model
EncDecCTCModelBPE was successfully restored from
/root/.cache/huggingface/hub/models--nvidia--parakeet-ctc-0.6b/snapshots/16
ca39445465932bfbaeb5126933d5ce8bd43a77/parakeet-ctc-0.6b.nemo.

EncDecCTCModelBPE(
  (preprocessor): AudioToMelSpectrogramPreprocessor(
    (featurizer): FilterbankFeatures()
  )
  (encoder): ConformerEncoder(
    (pre_encode): ConvSubsampling(
      (out): Linear(in_features=2560, out_features=1024, bias=True)
      (conv): Sequential(
        (0): Conv2d(1, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (4): ReLU(inplace=True)
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (6): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (7): ReLU(inplace=True)
      )
    )
    (pos_enc): RelPositionalEncoding(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (layers): ModuleList(
      (0-23): 24 x ConformerLayer(
        (norm_feed_forward1): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward1): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (norm_conv): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (conv): ConformerConvolution(
          (pointwise_conv1): Conv1d(1024, 2048, kernel_size=(1,),
stride=(1,))
```

```
          (depthwise_conv): CausalConv1D(1024, 1024, kernel_size=(9,),
stride=(1,), groups=1024)
          (batch_norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
          (activation): Swish()
          (pointwise_conv2): Conv1d(1024, 1024, kernel_size=(1,),
stride=(1,))
        )
        (norm_self_att): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (self_attn): RelPositionMultiHeadAttention(
          (linear_q): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_k): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_v): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_out): Linear(in_features=1024, out_features=1024,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear_pos): Linear(in_features=1024, out_features=1024,
bias=False)
        )
        (norm_feed_forward2): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward2): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (norm_out): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
  (decoder): ConvASRDecoder(
    (decoder_layers): Sequential(
      (0): Conv1d(1024, 1025, kernel_size=(1,), stride=(1,))
    )
  )
  (loss): CTCLoss()
  (spec_augmentation): SpectrogramAugmentation(
    (spec_augment): SpecAugment()
  )
```

```
  (wer): WER()
)
```

Parakeet CTC 1.1B (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="nvidia/parake
et-ctc-1.1b")
model
```

```
[NeMo I 2024-11-25 17:56:43 mixins:176] Tokenizer SentencePieceTokenizer
initialized with 1024 tokens

[NeMo W 2024-11-25 17:56:43 modelPT:176] If you intend to do training or
fine-tuning, please call the ModelPT.setup_training_data() method and
provide a valid configuration file to setup the train data loader.
    Train config :
    manifest_filepath:
/disk1/NVIDIA/datasets/LibriSpeech_NeMo/librivox-train-all.json
    sample_rate: 16000
    batch_size: 16
    shuffle: true
    num_workers: 8
    pin_memory: true
    use_start_end_token: false
    trim_silence: false
    max_duration: 16.7
    min_duration: 0.1
    is_tarred: false
    tarred_audio_filepaths: null
    shuffle_n: 2048
    bucketing_strategy: fully_randomized
    bucketing_batch_size: null

[NeMo W 2024-11-25 17:56:43 modelPT:183] If you intend to do validation,
please call the ModelPT.setup_validation_data() or
ModelPT.setup_multiple_validation_data() method and provide a valid
configuration file to setup the validation data loader(s).
    Validation config :
    manifest_filepath:
/disk1/NVIDIA/datasets/LibriSpeech_NeMo/librivox-dev-clean.json
    sample_rate: 16000
    batch_size: 16
    shuffle: false
    use_start_end_token: false
    num_workers: 8
    pin_memory: true
```

```
[NeMo W 2024-11-25 17:56:44 modelPT:189] Please call the
ModelPT.setup_test_data() or ModelPT.setup_multiple_test_data() method and
provide a valid configuration file to setup the test data loader(s).
    Test config :
    manifest_filepath: null
    sample_rate: 16000
    batch_size: 16
    shuffle: false
    use_start_end_token: false
    num_workers: 8
    pin_memory: true


[NeMo I 2024-11-25 17:56:44 features:305] PADDING: 0
[NeMo I 2024-11-25 17:57:19 save_restore_connector:275] Model
EncDecCTCModelBPE was successfully restored from
/root/.cache/huggingface/hub/models--nvidia--parakeet-ctc-1.1b/snapshots/08
5a3de63c7598065b072cd8f2182e6a5fa593eb/parakeet-ctc-1.1b.nemo.

EncDecCTCModelBPE(
  (preprocessor): AudioToMelSpectrogramPreprocessor(
    (featurizer): FilterbankFeatures()
  )
  (encoder): ConformerEncoder(
    (pre_encode): ConvSubsampling(
      (out): Linear(in_features=2560, out_features=1024, bias=True)
      (conv): Sequential(
        (0): Conv2d(1, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (4): ReLU(inplace=True)
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (6): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (7): ReLU(inplace=True)
      )
    )
    (pos_enc): RelPositionalEncoding(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (layers): ModuleList(
      (0-41): 42 x ConformerLayer(
        (norm_feed_forward1): LayerNorm((1024,), eps=1e-05,
```

```
elementwise_affine=True)
        (feed_forward1): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (norm_conv): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (conv): ConformerConvolution(
          (pointwise_conv1): Conv1d(1024, 2048, kernel_size=(1,),
stride=(1,))
          (depthwise_conv): CausalConv1D(1024, 1024, kernel_size=(9,),
stride=(1,), groups=1024)
          (batch_norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
          (activation): Swish()
          (pointwise_conv2): Conv1d(1024, 1024, kernel_size=(1,),
stride=(1,))
        )
        (norm_self_att): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (self_attn): RelPositionMultiHeadAttention(
          (linear_q): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_k): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_v): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_out): Linear(in_features=1024, out_features=1024,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear_pos): Linear(in_features=1024, out_features=1024,
bias=False)
        )
        (norm_feed_forward2): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward2): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (norm_out): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      )
    )
```

```
  )
  (decoder): ConvASRDecoder(
    (decoder_layers): Sequential(
      (0): Conv1d(1024, 1025, kernel_size=(1,), stride=(1,))
    )
  )
  (loss): CTCLoss()
  (spec_augmentation): SpectrogramAugmentation(
    (spec_augment): SpecAugment()
  )
  (wer): WER()
)
```

Parakeet RNNT 0.6B (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="nvidia/parak
eet-rnnt-0.6b")
model
```

```
{"model_id":"fa516b85cbee4f22b6effc173764b7a7","version_major":2,"version_m
inor":0}
```

```
[NeMo I 2024-11-25 18:21:13 mixins:176] Tokenizer SentencePieceTokenizer
initialized with 1024 tokens

[NeMo W 2024-11-25 18:21:14 modelPT:176] If you intend to do training or
fine-tuning, please call the ModelPT.setup_training_data() method and
provide a valid configuration file to setup the train data loader.
    Train config :
    manifest_filepath:
/disk1/NVIDIA/datasets/LibriSpeech_NeMo/librivox-train-all.json
    sample_rate: 16000
    batch_size: 16
    shuffle: true
    num_workers: 8
    pin_memory: true
    use_start_end_token: false
    trim_silence: false
    max_duration: 16.7
    min_duration: 0.1
    is_tarred: false
    tarred_audio_filepaths: null
    shuffle_n: 2048
    bucketing_strategy: fully_randomized
    bucketing_batch_size: null
```

```
[NeMo W 2024-11-25 18:21:14 modelPT:183] If you intend to do validation,
please call the ModelPT.setup_validation_data() or
ModelPT.setup_multiple_validation_data() method and provide a valid
configuration file to setup the validation data loader(s).
    Validation config :
    manifest_filepath:
/disk1/NVIDIA/datasets/LibriSpeech_NeMo/librivox-dev-clean.json
    sample_rate: 16000
    batch_size: 16
    shuffle: false
    use_start_end_token: false
    num_workers: 8
    pin_memory: true

[NeMo W 2024-11-25 18:21:14 modelPT:189] Please call the
ModelPT.setup_test_data() or ModelPT.setup_multiple_test_data() method and
provide a valid configuration file to setup the test data loader(s).
    Test config :
    manifest_filepath: null
    sample_rate: 16000
    batch_size: 16
    shuffle: false
    use_start_end_token: false
    num_workers: 8
    pin_memory: true


[NeMo I 2024-11-25 18:21:14 features:305] PADDING: 0
[NeMo I 2024-11-25 18:21:19 rnnt_models:226] Using RNNT Loss :
warprnnt_numba
    Loss warprnnt_numba_kwargs: {'fastemit_lambda': 0.0, 'clamp': -1.0}
[NeMo I 2024-11-25 18:21:20 rnnt_models:226] Using RNNT Loss :
warprnnt_numba
    Loss warprnnt_numba_kwargs: {'fastemit_lambda': 0.0, 'clamp': -1.0}

[NeMo W 2024-11-25 18:21:20 rnnt_loop_labels_computer:270] No conditional
node support for Cuda.
    Cuda graphs with while loops are disabled, decoding speed will be
slower
    Reason: CUDA is not available

[NeMo I 2024-11-25 18:21:20 rnnt_models:226] Using RNNT Loss :
warprnnt_numba
    Loss warprnnt_numba_kwargs: {'fastemit_lambda': 0.0, 'clamp': -1.0}

[NeMo W 2024-11-25 18:21:20 rnnt_loop_labels_computer:270] No conditional
node support for Cuda.
```

```
    Cuda graphs with while loops are disabled, decoding speed will be
slower
    Reason: CUDA is not available

[NeMo I 2024-11-25 18:21:36 save_restore_connector:275] Model
EncDecRNNTBPEModel was successfully restored from
/root/.cache/huggingface/hub/models--nvidia--parakeet-rnnt-0.6b/snapshots/9
2452edc2d210f9bad2d69b83e44582d5b7e4c12/parakeet-rnnt-0.6b.nemo.

EncDecRNNTBPEModel(
  (preprocessor): AudioToMelSpectrogramPreprocessor(
    (featurizer): FilterbankFeatures()
  )
  (encoder): ConformerEncoder(
    (pre_encode): ConvSubsampling(
      (out): Linear(in_features=2560, out_features=1024, bias=True)
      (conv): Sequential(
        (0): Conv2d(1, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (4): ReLU(inplace=True)
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (6): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (7): ReLU(inplace=True)
      )
    )
    (pos_enc): RelPositionalEncoding(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (layers): ModuleList(
      (0-23): 24 x ConformerLayer(
        (norm_feed_forward1): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward1): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (norm_conv): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (conv): ConformerConvolution(
          (pointwise_conv1): Conv1d(1024, 2048, kernel_size=(1,),
stride=(1,))
```

```
          (depthwise_conv): CausalConv1D(1024, 1024, kernel_size=(9,),
stride=(1,), groups=1024)
          (batch_norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
          (activation): Swish()
          (pointwise_conv2): Conv1d(1024, 1024, kernel_size=(1,),
stride=(1,))
        )
        (norm_self_att): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (self_attn): RelPositionMultiHeadAttention(
          (linear_q): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_k): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_v): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_out): Linear(in_features=1024, out_features=1024,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear_pos): Linear(in_features=1024, out_features=1024,
bias=False)
        )
        (norm_feed_forward2): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward2): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (norm_out): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
  (decoder): RNNTDecoder(
    (prediction): ModuleDict(
      (embed): Embedding(1025, 640, padding_idx=1024)
      (dec_rnn): LSTMDropout(
        (lstm): LSTM(640, 640, num_layers=2, dropout=0.2)
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
  )
  (joint): RNNTJoint(
```

```
    (pred): Linear(in_features=640, out_features=640, bias=True)
    (enc): Linear(in_features=1024, out_features=640, bias=True)
    (joint_net): Sequential(
      (0): ReLU(inplace=True)
      (1): Dropout(p=0.2, inplace=False)
      (2): Linear(in_features=640, out_features=1025, bias=True)
    )
    (_loss): RNNTLoss(
      (_loss): RNNTLossNumba()
    )
    (_wer): WER()
  )
  (loss): RNNTLoss(
    (_loss): RNNTLossNumba()
  )
  (spec_augmentation): SpectrogramAugmentation(
    (spec_augment): SpecAugment()
  )
  (wer): WER()
)
```

Parakeet RNNT 1.1B (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="nvidia/parak
eet-rnnt-1.1b")
model
```

```
[NeMo I 2024-11-25 18:31:53 mixins:176] Tokenizer SentencePieceTokenizer
initialized with 1024 tokens

[NeMo W 2024-11-25 18:31:54 modelPT:176] If you intend to do training or
fine-tuning, please call the ModelPT.setup_training_data() method and
provide a valid configuration file to setup the train data loader.
    Train config :
    manifest_filepath:
/disk1/NVIDIA/datasets/LibriSpeech_NeMo/librivox-train-all.json
    sample_rate: 16000
    batch_size: 16
    shuffle: true
    num_workers: 8
    pin_memory: true
    use_start_end_token: false
    trim_silence: false
    max_duration: 16.7
    min_duration: 0.1
    is_tarred: false
```

```
    tarred_audio_filepaths: null
    shuffle_n: 2048
    bucketing_strategy: fully_randomized
    bucketing_batch_size: null

[NeMo W 2024-11-25 18:31:54 modelPT:183] If you intend to do validation,
please call the ModelPT.setup_validation_data() or
ModelPT.setup_multiple_validation_data() method and provide a valid
configuration file to setup the validation data loader(s).
    Validation config :
    manifest_filepath:
/disk1/NVIDIA/datasets/LibriSpeech_NeMo/librivox-dev-clean.json
    sample_rate: 16000
    batch_size: 16
    shuffle: false
    use_start_end_token: false
    num_workers: 8
    pin_memory: true

[NeMo W 2024-11-25 18:31:54 modelPT:189] Please call the
ModelPT.setup_test_data() or ModelPT.setup_multiple_test_data() method and
provide a valid configuration file to setup the test data loader(s).
    Test config :
    manifest_filepath: null
    sample_rate: 16000
    batch_size: 16
    shuffle: false
    use_start_end_token: false
    num_workers: 8
    pin_memory: true


[NeMo I 2024-11-25 18:31:54 features:305] PADDING: 0
[NeMo I 2024-11-25 18:32:07 rnnt_models:226] Using RNNT Loss :
warprnnt_numba
    Loss warprnnt_numba_kwargs: {'fastemit_lambda': 0.0, 'clamp': -1.0}
[NeMo I 2024-11-25 18:32:07 rnnt_models:226] Using RNNT Loss :
warprnnt_numba
    Loss warprnnt_numba_kwargs: {'fastemit_lambda': 0.0, 'clamp': -1.0}

[NeMo W 2024-11-25 18:32:07 rnnt_loop_labels_computer:270] No conditional
node support for Cuda.
    Cuda graphs with while loops are disabled, decoding speed will be
slower
    Reason: CUDA is not available
```

```
[NeMo I 2024-11-25 18:32:07 rnnt_models:226] Using RNNT Loss :
warprnnt_numba
    Loss warprnnt_numba_kwargs: {'fastemit_lambda': 0.0, 'clamp': -1.0}

[NeMo W 2024-11-25 18:32:07 rnnt_loop_labels_computer:270] No conditional
node support for Cuda.
    Cuda graphs with while loops are disabled, decoding speed will be
slower
    Reason: CUDA is not available

[NeMo I 2024-11-25 18:32:29 save_restore_connector:275] Model
EncDecRNNTBPEModel was successfully restored from
/root/.cache/huggingface/hub/models--nvidia--parakeet-rnnt-1.1b/snapshots/2
326ee79c014e914d01b12595eacc243d2816a69/parakeet-rnnt-1.1b.nemo.

EncDecRNNTBPEModel(
  (preprocessor): AudioToMelSpectrogramPreprocessor(
    (featurizer): FilterbankFeatures()
  )
  (encoder): ConformerEncoder(
    (pre_encode): ConvSubsampling(
      (out): Linear(in_features=2560, out_features=1024, bias=True)
      (conv): Sequential(
        (0): Conv2d(1, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
        (1): ReLU(inplace=True)
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (4): ReLU(inplace=True)
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (6): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (7): ReLU(inplace=True)
      )
    )
    (pos_enc): RelPositionalEncoding(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (layers): ModuleList(
      (0-41): 42 x ConformerLayer(
        (norm_feed_forward1): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward1): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
```

```
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (norm_conv): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (conv): ConformerConvolution(
          (pointwise_conv1): Conv1d(1024, 2048, kernel_size=(1,),
stride=(1,))
          (depthwise_conv): CausalConv1D(1024, 1024, kernel_size=(9,),
stride=(1,), groups=1024)
          (batch_norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
          (activation): Swish()
          (pointwise_conv2): Conv1d(1024, 1024, kernel_size=(1,),
stride=(1,))
        )
        (norm_self_att): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (self_attn): RelPositionMultiHeadAttention(
          (linear_q): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_k): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_v): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_out): Linear(in_features=1024, out_features=1024,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
          (linear_pos): Linear(in_features=1024, out_features=1024,
bias=False)
        )
        (norm_feed_forward2): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward2): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (norm_out): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
  (decoder): RNNTDecoder(
    (prediction): ModuleDict(
      (embed): Embedding(1025, 640, padding_idx=1024)
      (dec_rnn): LSTMDropout(
```

```
        (lstm): LSTM(640, 640, num_layers=2, dropout=0.2)
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
  )
  (joint): RNNTJoint(
    (pred): Linear(in_features=640, out_features=640, bias=True)
    (enc): Linear(in_features=1024, out_features=640, bias=True)
    (joint_net): Sequential(
      (0): ReLU(inplace=True)
      (1): Dropout(p=0.2, inplace=False)
      (2): Linear(in_features=640, out_features=1025, bias=True)
    )
    (_loss): RNNTLoss(
      (_loss): RNNTLossNumba()
    )
    (_wer): WER()
  )
  (loss): RNNTLoss(
    (_loss): RNNTLossNumba()
  )
  (spec_augmentation): SpectrogramAugmentation(
    (spec_augment): SpecAugment()
  )
  (wer): WER()
)
```

Parakeet TDT 1.1B (en)

```
%%capture
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="nvidia/parak
eet-tdt-1.1b")

model

EncDecRNNTBPEModel(
  (preprocessor): AudioToMelSpectrogramPreprocessor(
    (featurizer): FilterbankFeatures()
  )
  (encoder): ConformerEncoder(
    (pre_encode): ConvSubsampling(
      (out): Linear(in_features=2560, out_features=1024, bias=True)
      (conv): Sequential(
        (0): Conv2d(1, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1))
        (1): ReLU(inplace=True)
```

```
        (2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (3): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (4): ReLU(inplace=True)
        (5): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), groups=256)
        (6): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        (7): ReLU(inplace=True)
      )
    )
    (pos_enc): RelPositionalEncoding(
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (layers): ModuleList(
      (0-41): 42 x ConformerLayer(
        (norm_feed_forward1): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward1): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (norm_conv): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (conv): ConformerConvolution(
          (pointwise_conv1): Conv1d(1024, 2048, kernel_size=(1,),
stride=(1,))
          (depthwise_conv): CausalConv1D(1024, 1024, kernel_size=(9,),
stride=(1,), groups=1024)
          (batch_norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
          (activation): Swish()
          (pointwise_conv2): Conv1d(1024, 1024, kernel_size=(1,),
stride=(1,))
        )
        (norm_self_att): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (self_attn): RelPositionMultiHeadAttention(
          (linear_q): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_k): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_v): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_out): Linear(in_features=1024, out_features=1024,
bias=True)
```

```
          (dropout): Dropout(p=0.1, inplace=False)
          (linear_pos): Linear(in_features=1024, out_features=1024,
bias=False)
        )
        (norm_feed_forward2): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (feed_forward2): ConformerFeedForward(
          (linear1): Linear(in_features=1024, out_features=4096, bias=True)
          (activation): Swish()
          (dropout): Dropout(p=0.1, inplace=False)
          (linear2): Linear(in_features=4096, out_features=1024, bias=True)
        )
        (dropout): Dropout(p=0.1, inplace=False)
        (norm_out): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
  (decoder): RNNTDecoder(
    (prediction): ModuleDict(
      (embed): Embedding(1025, 640, padding_idx=1024)
      (dec_rnn): LSTMDropout(
        (lstm): LSTM(640, 640, num_layers=2, dropout=0.2)
        (dropout): Dropout(p=0.2, inplace=False)
      )
    )
  )
  (joint): RNNTJoint(
    (pred): Linear(in_features=640, out_features=640, bias=True)
    (enc): Linear(in_features=1024, out_features=640, bias=True)
    (joint_net): Sequential(
      (0): ReLU(inplace=True)
      (1): Dropout(p=0.2, inplace=False)
      (2): Linear(in_features=640, out_features=1030, bias=True)
    )
    (_loss): RNNTLoss(
      (_loss): TDTLossNumba()
    )
    (_wer): WER()
  )
  (loss): RNNTLoss(
    (_loss): TDTLossNumba()
  )
  (spec_augmentation): SpectrogramAugmentation(
    (spec_augment): SpecAugment()
  )
```

```
  (wer): WER()
)
```

Parakeet TDT-CTC 1.1B PnC(en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.ASRModel.from_pretrained(model_name="nvidia/parakeet-tdt_ctc-1.1b")
model
```

Parakeet TDT-CTC 110M PnC(en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.ASRModel.from_pretrained(model_name="nvidia/parakeet-tdt_ctc-110m")
model
```

Conformer

NVIDIA Conformer-CTC Small (en-US)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained("nvidia/stt_en_conformer_ctc_small")
model
```

NVIDIA Conformer-CTC Large (en-US)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained("nvidia/stt_en_conformer_ctc_large")
model
```

NVIDIA Conformer-Transducer Large (en-US)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained("nvidia/stt_en_conformer_transducer_large")
model
```

NVIDIA Conformer-Transducer X-Large (en-US)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained("nvidia/stt_en_conformer
```

```
_transducer_xlarge")
model
```

FastConformer

NVIDIA FastConformer-CTC Large (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="nvidia/stt_en
_fastconformer_ctc_large")
model
```

NVIDIA FastConformer-CTC XLarge (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecCTCTBPEModel.from_pretrained(model_name="nvidia/stt_e
n_fastconformer_ctc_xlarge")
model
```

NVIDIA FastConformer-CTC XXLarge (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecCTCTBPEModel.from_pretrained(model_name="nvidia/stt_e
n_fastconformer_ctc_xxlarge")
model
```

NVIDIA FastConformer-Transducer Large (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="nvidia/stt_e
n_fastconformer_transducer_large")
model
```

NVIDIA FastConformer-Transducer XLarge (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="nvidia/stt_e
n_fastconformer_transducer_xlarge")
model
```

NVIDIA FastConformer-Transducer XXLarge (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="nvidia/stt_e
```

```
n_fastconformer_transducer_xxlarge")
model
```

NVIDIA FastConformer-Hybrid Large (en)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecHybridRNNTCTCBPEModel.from_pretrained(model_name="nvi
dia/stt_en_fastconformer_hybrid_large_pc")
model
```

NVIDIA Streaming Conformer-Hybrid Large (en-US)

```
import nemo.collections.asr as nemo_asr
model =
nemo_asr.models.EncDecHybridRNNTCTCBPEModel.from_pretrained(model_name="nvi
dia/stt_en_fastconformer_hybrid_large_streaming_multi")

# Optional: change the default latency. Default latency is 1040ms.
Supported latencies: {0: 0ms, 1: 80ms, 16: 480ms, 33: 1040ms}.
# Note: These are the worst latency and average latency would be half of
these numbers.
model.encoder.set_default_att_context_size([70,13])

#Optional: change the default decoder. Default decoder is Transducer
(RNNT). Supported decoders: {ctc, rnnt}.
model.change_decoding_strategy(decoder_type='rnnt')

model.transcribe(['2086-149220-0033.wav'])
```

Jasper

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModel.from_pretrained(model_name="stt_en_jasper10x
5dr")
```

Quartznet

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModel.from_pretrained(model_name="stt_en_quartznet
15x5")
```

ContextNet

STT En ContextNet 256

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="stt_en_conte
xtnet_256")
```

STT En ContextNet 512

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="stt_en_conte
xtnet_512")
```

STT En ContextNet 1024

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="stt_en_conte
xtnet_1024")
```

STT En ContextNet 256 MLS

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="stt_en_conte
xtnet_256_mls")
```

STT En ContextNet 512 MLS

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="stt_en_conte
xtnet_512_mls")
```

STT En ContextNet 1024 MLS

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecRNNTBPEModel.from_pretrained(model_name="stt_en_conte
xtnet_1024_mls")
```

Citrinet

NVIDIA Citrinet CTC 256 Librispeech (en-US)

```
import nemo.collections.asr as nemo_asr
asr_model =
```

```
nemo_asr.models.EncDecCTCModelBPE.from_pretrained("nvidia/stt_en_citrinet_256_ls")
```

NVIDIA Citrinet CTC 384 Librispeech (en-US)

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained("nvidia/stt_en_citrinet_384_ls")
```

NVIDIA Citrinet CTC 512 Librispeech (en-US)

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained("nvidia/stt_en_citrinet_512_ls")
```

NVIDIA Citrinet CTC 768 Librispeech (en-US)

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained("nvidia/stt_en_citrinet_768_ls")
```

NVIDIA Citrinet CTC 1024 Librispeech (en-US)

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained("nvidia/stt_en_citrinet_1024_ls")
```

NVIDIA Streaming Citrinet 1024 (en-US)

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained("nvidia/stt_en_citrinet_1024_gamma_0_25")
```

Squeezeformer

STT En Squeezeformer Large Small Librispeech

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="stt_en_squeezeformer_ctc_large_ls")
```

STT En Squeezeformer CTC Small Librispeech

```
import nemo.collections.asr as nemo_asr
asr_model =
```

```
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="stt_en_squeez
eformer_ctc_small_ls")
```

STT En Squeezeformer CTC Medium Librispeech

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="stt_en_squeez
eformer_ctc_medium_ls")
```

STT En Squeezeformer CTC Medium-Large Librispeech

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="stt_en_squeez
eformer_ctc_medium_large_ls")
```

STT En Squeezeformer CTC Small-Medium Librispeech

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="stt_en_squeez
eformer_ctc_small_medium_ls")
```

STT En Squeezeformer CTC XSmall Librispeech

```
import nemo.collections.asr as nemo_asr
asr_model =
nemo_asr.models.EncDecCTCModelBPE.from_pretrained(model_name="stt_en_squeez
eformer_ctc_xsmall_ls")
```

Outros

SSL En Conformer Large

```
# %%capture
# import nemo.collections.asr as nemo_asr
# ssl_model =
nemo_asr.models.ssl_models.SpeechEncDecSelfSupervisedModel.from_pretrained(
model_name='ssl_en_conformer_large')
```

Wav2Vec2-Conformer

```
from transformers import Wav2Vec2ConformerConfig, Wav2Vec2ConformerModel

configuration = Wav2Vec2ConformerConfig()
model = Wav2Vec2ConformerModel(configuration)
configuration = model.config
```

```
%%capture
from transformers import AutoProcessor, Wav2Vec2ConformerModel


model =
Wav2Vec2ConformerModel.from_pretrained("facebook/wav2vec2-conformer-rope-la
rge-960h-ft")

# inputs = processor(dataset[0]["audio"]["array"],
sampling_rate=sampling_rate, return_tensors="pt")
# with torch.no_grad():
#     outputs = model(**inputs)

# last_hidden_states = outputs.last_hidden_state
# list(last_hidden_states.shape)

model

Wav2Vec2ConformerModel(
  (feature_extractor): Wav2Vec2ConformerFeatureEncoder(
    (conv_layers): ModuleList(
      (0): Wav2Vec2ConformerLayerNormConvLayer(
        (conv): Conv1d(1, 512, kernel_size=(10,), stride=(5,))
        (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (activation): GELUActivation()
      )
      (1-4): 4 x Wav2Vec2ConformerLayerNormConvLayer(
        (conv): Conv1d(512, 512, kernel_size=(3,), stride=(2,))
        (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (activation): GELUActivation()
      )
      (5-6): 2 x Wav2Vec2ConformerLayerNormConvLayer(
        (conv): Conv1d(512, 512, kernel_size=(2,), stride=(2,))
        (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
        (activation): GELUActivation()
      )
    )
  )
  (feature_projection): Wav2Vec2ConformerFeatureProjection(
    (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (projection): Linear(in_features=512, out_features=1024, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): Wav2Vec2ConformerEncoder(
    (embed_positions): Wav2Vec2ConformerRotaryPositionalEmbedding()
    (pos_conv_embed): Wav2Vec2ConformerPositionalConvEmbedding(
```

```
      (conv): ParametrizedConv1d(
        1024, 1024, kernel_size=(128,), stride=(1,), padding=(64,),
groups=16
        (parametrizations): ModuleDict(
          (weight): ParametrizationList(
            (0): _WeightNorm()
          )
        )
      )
      (padding): Wav2Vec2ConformerSamePadLayer()
      (activation): GELUActivation()
    )
    (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (layers): ModuleList(
      (0-23): 24 x Wav2Vec2ConformerEncoderLayer(
        (ffn1_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (ffn1): Wav2Vec2ConformerFeedForward(
          (intermediate_dropout): Dropout(p=0.1, inplace=False)
          (intermediate_dense): Linear(in_features=1024, out_features=4096,
bias=True)
          (intermediate_act_fn): SiLU()
          (output_dense): Linear(in_features=4096, out_features=1024,
bias=True)
          (output_dropout): Dropout(p=0.1, inplace=False)
        )
        (self_attn_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (self_attn_dropout): Dropout(p=0.1, inplace=False)
        (self_attn): Wav2Vec2ConformerSelfAttention(
          (linear_q): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_k): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_v): Linear(in_features=1024, out_features=1024,
bias=True)
          (linear_out): Linear(in_features=1024, out_features=1024,
bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (conv_module): Wav2Vec2ConformerConvolutionModule(
          (layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
          (pointwise_conv1): Conv1d(1024, 2048, kernel_size=(1,),
stride=(1,), bias=False)
```

```
          (glu): GLU(dim=1)
          (depthwise_conv): Conv1d(1024, 1024, kernel_size=(31,),
stride=(1,), padding=(15,), groups=1024, bias=False)
          (batch_norm): BatchNorm1d(1024, eps=1e-05, momentum=0.1,
affine=True, track_running_stats=True)
          (activation): SiLU()
          (pointwise_conv2): Conv1d(1024, 1024, kernel_size=(1,),
stride=(1,), bias=False)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (ffn2_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
        (ffn2): Wav2Vec2ConformerFeedForward(
          (intermediate_dropout): Dropout(p=0.1, inplace=False)
          (intermediate_dense): Linear(in_features=1024, out_features=4096,
bias=True)
          (intermediate_act_fn): SiLU()
          (output_dense): Linear(in_features=4096, out_features=1024,
bias=True)
          (output_dropout): Dropout(p=0.1, inplace=False)
        )
        (final_layer_norm): LayerNorm((1024,), eps=1e-05,
elementwise_affine=True)
      )
    )
  )
)
```

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião (“gate”) de aprovação: 28 de out. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

As atividades deste STAGE concentraram-se na continuidade do desenvolvimento das análises relacionadas à comparação de modelos de áudio com as ativações do córtex auditivo humano.

Resumi brevemente o funcionamento de todos os modelos dos quais as ativações foram extraídas, com destaque para os modelos ASR da Nvidia:
Nvidia ASR Models

Fiz um script para guardar as ativações de cada um dos modelos selecionados durante a inferência do dataset de Sam Norman-Haignere et al :

get_activations.ipynb

Realizei a comparação entre o dataset de fmri e as ativações colhidas do modelo canary-1b, baseado em Many but not all deep neural network audio models capture brain responses and exhibit correspondence between model stages and brain regions:
regression_analysis.ipynb

Expliquei o princípio dessa comparação baseado em:
Análise regressão

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

Pretendo aplicar o mesmo procedimento aos demais modelos selecionados, realizar a análise dos resultados obtidos e conduzir testes adicionais.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

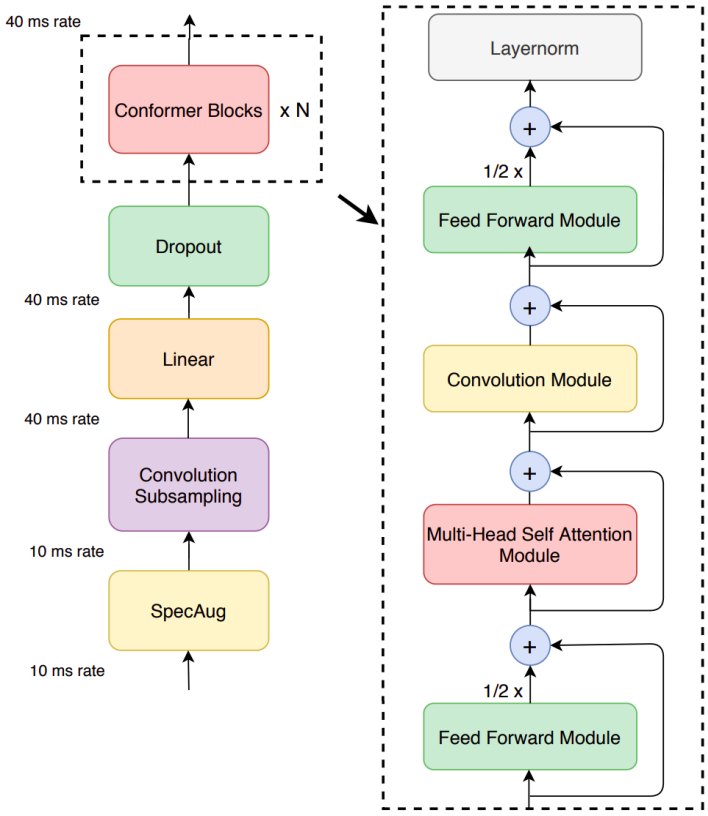
CEDRIC LUIZ DE CARVALHO: Go!

[Nvidia ASR Models]

Modelos de ASR NVIDIA

→ Canary

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182



- Funcionamento:
  Modelo encoder-decoder que utiliza FastConformer Encoder (eficiente para extrair representações) e Transformer Decoder (geração robusta de texto).
  Trabalha com reconhecimento automático de fala (ASR) e tradução.

- Características:

  - Multilíngue: Suporte para Inglês, Alemão, Francês e Espanhol.
  - Multi-tarefa: Reconhecimento de fala e tradução entre os idiomas suportados.
  - Destaque: Topo do OpenASR Leaderboard na HuggingFace.

Parakeet

- Funcionamento:
  Família de modelos baseada no FastConformer Encoder, com diferentes decodificadores:

  - CTC: Loss e decodificação direta, não autoregressiva.
  - RNNT: Loss transducer, modelo autoregressivo.

    - TDT: Decodificação baseada em tarefas específicas.
- Características:
    - Diferentes tamanhos de modelo (0.6B e 1.1B).
    - Modelos especializados para CTC, RNNT e TDT.
    - Disponível em espaços interativos para testes.

→ Conformer

1. Conformer-CTC

- Funcionamento:
  Baseado em loss CTC (não-autoregressivo), combina atenção e convoluções para capturar interações globais e locais.
    - Linear decoder substitui LSTM, reduzindo a complexidade.
- Características:
    - Codificação em nível de caracteres ou sub-palavras (BPE).
    - Rápido e eficiente para inferência sem comprometer a qualidade.

2. Conformer-Transducer

- Funcionamento:
  Utiliza loss RNNT (autoregressivo), preservando o encoder do Conformer-CTC.
    - Adequado para aplicações que exigem geração sequencial.
- Características:
    - Codificação em sub-palavras ou caracteres.
    - Alta precisão para tarefas dependentes de contexto.

3. Conformer-HAT (Hybrid Autoregressive Transducer)

- Funcionamento:
  Separação de previsões de tokens e pontuações de "blanks".
    - Permite subtração de probabilidades internas do modelo durante a decodificação com LM externo.

- Características:
    - Melhora a eficiência de modelos de linguagem externos.
    - Útil para adaptação em novos domínios.

Fast-Conformer

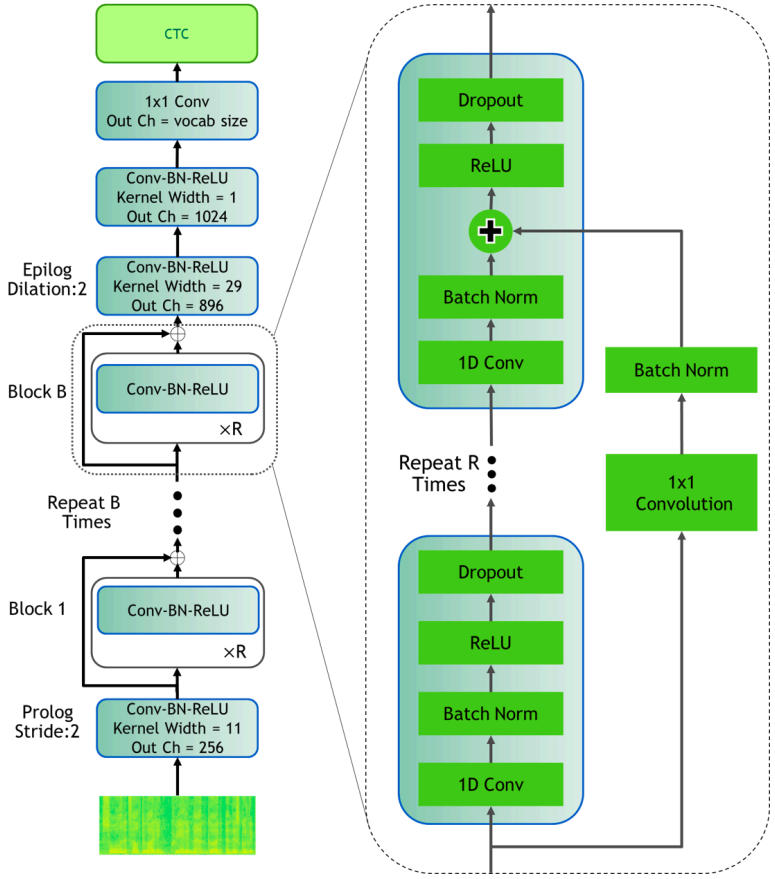
- Funcionamento:
  Versão otimizada do Conformer com encoder mais rápido, usando:
    - Subamostragem por convoluções profundidade-8.
    - Redução do tamanho do kernel para acelerar processamento.
- Características:
    - Até 2.4x mais rápido que o Conformer padrão.
    - Suporte para áudio longo (até 70 minutos).
    - Codificação baseada em sub-palavras (BPE).

→ Hybrid-Transducer-CTC

- Funcionamento:
  Combina decodificadores RNNT e CTC no mesmo modelo.
    - O RNNT é o padrão, mas pode alternar para CTC quando necessário.
- Características:
    - Treinamento unificado acelera a convergência.
    - Versões para codificação em caracteres ou sub-palavras.

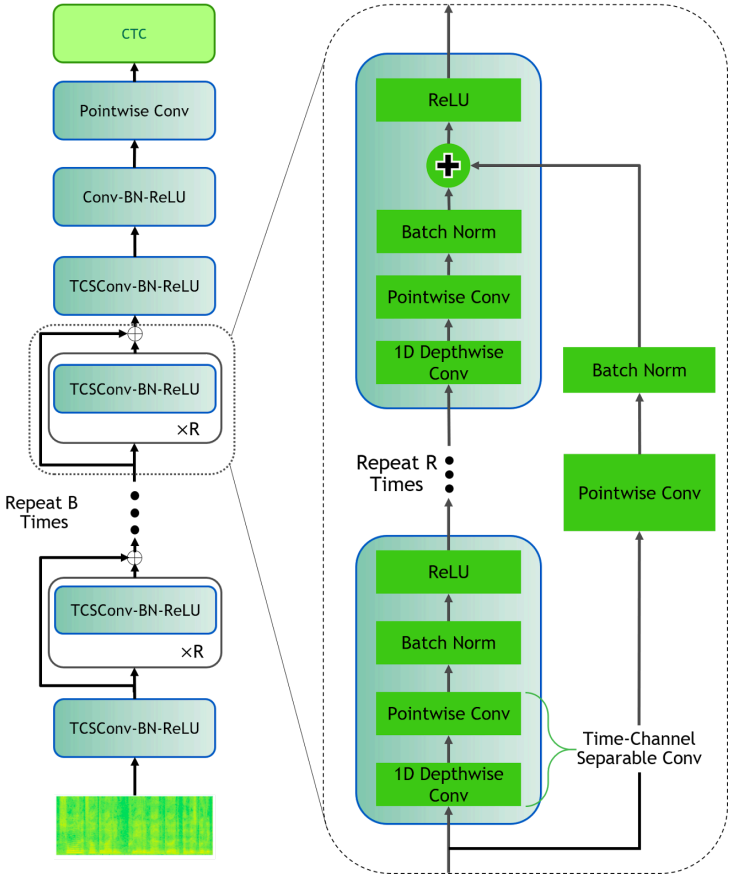
→ Jasper e QuartzNet

- Jasper:



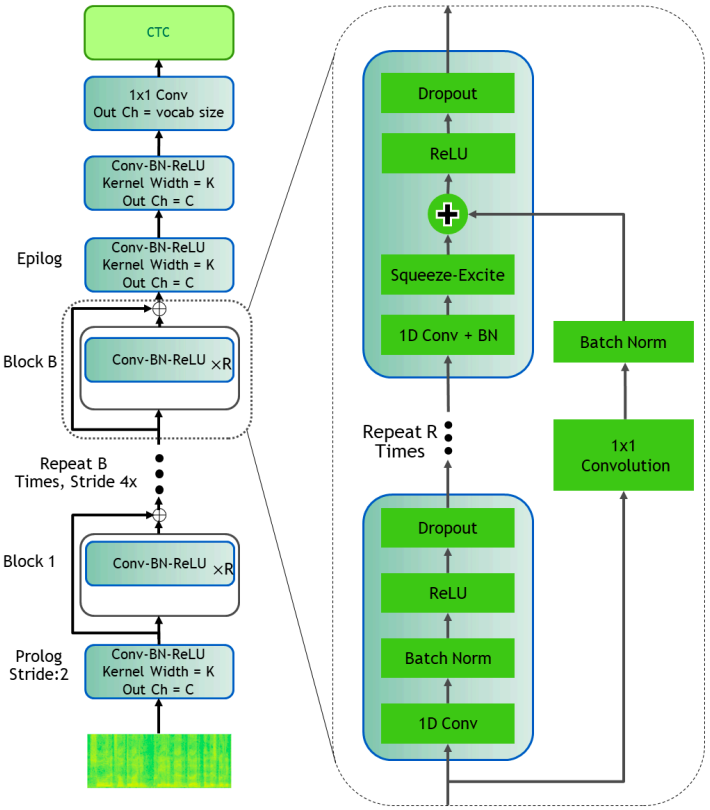
- Baseado em redes neurais convolucionais profundas (TDNN).
- Estrutura modular com blocos de convolução 1D.

- QuartzNet:



- Variante mais leve do Jasper, com convoluções separáveis.
- Mantém a qualidade com menos parâmetros.

→ Citrinet



- Funcionamento:
  Versão avançada do QuartzNet com codificação BPE e mecanismo Squeeze-and-Excitation.
    - Decodificação baseada em CTC, garantindo eficiência.
- Características:
    - Alta precisão para transcrição de áudio.
    - Arquitetura otimizada para inferência rápida.

→ ContextNet

- Funcionamento:
  Modelo transducer com Squeeze-and-Excitation para contexto expandido.
    - Decodificação autoregressiva.

- Características:
    - Ideal para tarefas com forte dependência de contexto.
    - Codificação em caracteres ou sub-palavras.

→ Squeezeformer-CTC

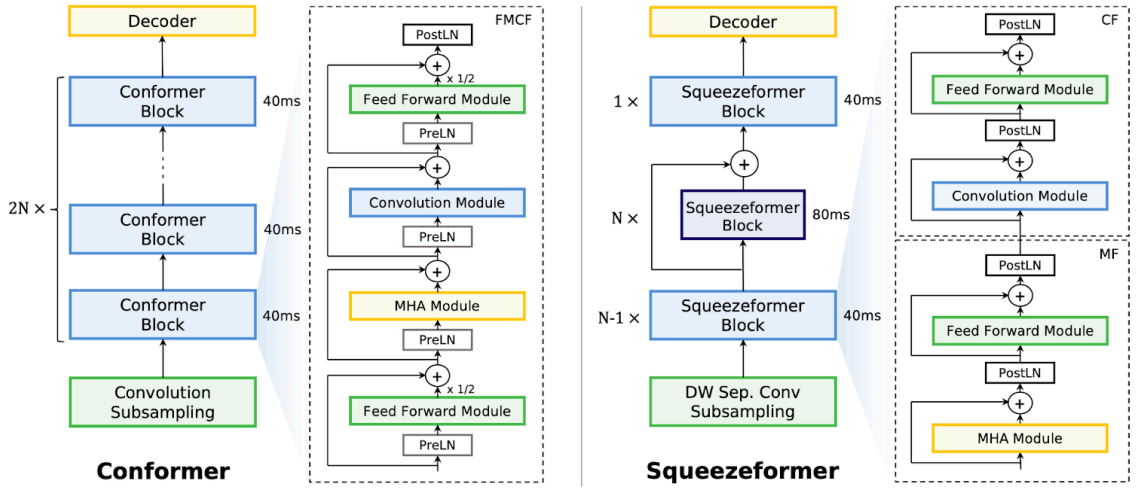


Fig. 2: *The Conformer architecutre (Left) and the Squeezeformer architecture (Right) which consists of the Temporal U-Net structure for downsampling and upsampling of the sampling rate, the standard Transformer-style block structure that only uses post-Layer Normalization, and the depth-wise separable subsampling layer.*

- Funcionamento:
  Variante CTC do Squeezeformer com redução temporal em estilo U-Net.
    - Uso eficiente de memória e FLOPs.
- Características:
    - Arquitetura simplificada e unificada.
    - Combinação de ativação temporal e convoluções para transcrição precisa.

[get_activations.ipynb]

```
!apt install build-essential -y
# ## Install NeMo
BRANCH = 'main'
!python -m pip install
git+https://github.com/NVIDIA/NeMo.git@$BRANCH#egg=nemo_toolkit[asr]
```

Model

```
from nemo.collections.asr.models import EncDecMultiTaskModel
model_name = 'nvidia/canary-1b'
model = EncDecMultiTaskModel.from_pretrained(model_name)
```

```
[name for name, module in model.named_children()]
```

```
['preprocessor',
 'encoder',
 'encoder_decoder_proj',
 'transf_decoder',
 'log_softmax',
 'loss',
 'spec_augmentation',
 'val_loss',
 'wer',
 'bleu']
```

Data

```
ls
```

```
2086-149220-0033.wav  auditory_brain_dnn/    model_actv/     results/
Untitled.ipynb        get_activations.ipynb  model_actv.tar  results.tar
```

```
import pickle
data_path = 'auditory_brain_dnn/data'
with open('auditory_brain_dnn/data/stimuli/df_stimuli_meta.pkl', 'rb') as
file:
    data = pickle.load(file)
```

```
data.head()
```

```
                            category_label
stim5_alarm_clock               Mechanical
stim7_applause                   HumNonVoc
stim11_baby_crying                  HumVoc
stim12_background_speech         EngSpeech
stim18_basketball_dribbling       EnvSound
```

```
from IPython import display
import os
sounds_path = os.path.join(data_path , 'stimuli/165_natural_sounds_16kHz')
wav_files = [os.path.join(sounds_path, f) for f in os.listdir(sounds_path)
if f.endswith('.wav')]

i=0
sample = wav_files[i]
display.Audio(sample)
```

```
<IPython.lib.display.Audio object>
```

```
sample
```

```
'auditory_brain_dnn/data/stimuli/165_natural_sounds_16kHz/stim515_sports_an
ouncer.wav'
```

```
[name for name, module in model.named_children()]
```

```
['preprocessor',
 'encoder',
 'encoder_decoder_proj',
 'transf_decoder',
 'log_softmax',
 'loss',
 'spec_augmentation',
 'val_loss',
 'wer',
 'bleu']
```

Get activations

```
activation = {}
def get_activation(name):
    def hook(model, input, output):
      activation[name] = output
    return hook


main_layers = [name for name, module in model.named_children()]

for layer_name in main_layers:
    layer = getattr(model, layer_name)
    layer.register_forward_hook(get_activation(layer_name))

## Test
model.transcribe(sample)
```

```
[NeMo W 2024-11-26 07:19:02 nemo_logging:405] You are using a non-tarred
dataset and requested tokenization during data sampling (pretokenize=True).
This will cause the tokenization to happen in the main (GPU) process,
possibly impacting the training speed if your tokenizer is very large. If
the impact is noticable, set pretokenize=False in dataloader config. (note:
that will disable token-per-second filtering and 2D bucketing features)
Transcribing: 1it [00:01,  1.38s/it]

[" You're the Air to Left Fields, going back up."]

activation

{'bleu': (tensor([[[ 0.6072,  2.4106,  5.3435,  ..., -0.4049, -0.4371,
-0.4595],
          [-0.1531,  2.6809,  5.8575,  ..., -0.4124, -0.4539, -0.5223],
          [-0.5172,  2.7586,  5.6086,  ..., -0.3967, -0.4466, -0.5451],
          ...,
          [-2.5891, -2.4419, -1.6626,  ..., -1.5484, -1.5133, -1.7161],
          [-1.8819, -1.8007, -1.5704,  ..., -1.4911, -1.2636, -1.7081],
          [-1.0207, -0.9640, -0.9132,  ..., -0.8762, -0.9578, -0.9664]]],
         device='cuda:0'),
  tensor([201], device='cuda:0')),
 'preprocessor': (tensor([[[ 0.6072,  2.4106,  5.3435,  ..., -0.4049,
-0.4371, -0.4595],
          [-0.1531,  2.6809,  5.8575,  ..., -0.4124, -0.4539, -0.5223],
          [-0.5172,  2.7586,  5.6086,  ..., -0.3967, -0.4466, -0.5451],
          ...,
          [-2.5891, -2.4419, -1.6626,  ..., -1.5484, -1.5133, -1.7161],
          [-1.8819, -1.8007, -1.5704,  ..., -1.4911, -1.2636, -1.7081],
          [-1.0207, -0.9640, -0.9132,  ..., -0.8762, -0.9578, -0.9664]]],
         device='cuda:0'),
  tensor([201], device='cuda:0')),
 'encoder': (tensor([[[-0.0385, -0.2132, -0.1946,  ..., -0.0151,  0.0932,
0.1486],
          [-0.0432,  0.2971,  0.3973,  ...,  0.0364,  0.2708,  0.3887],
          [-0.0840, -0.0017,  0.2391,  ...,  0.0286,  0.2253,  0.3538],
          ...,
          [ 0.0481,  0.2555,  0.0214,  ..., -0.2425,  0.2096, -0.0992],
          [-0.0261, -0.2649, -0.3325,  ..., -0.0436,  0.1118, -0.1256],
          [-0.0031,  0.0501,  0.3234,  ..., -0.1089,  0.1931,  0.1188]]],
         device='cuda:0'),
  tensor([26], device='cuda:0')),
 'encoder_decoder_proj': tensor([[[-0.0385, -0.0432, -0.0840,  ...,
0.0481, -0.0261, -0.0031],
          [-0.2132,  0.2971, -0.0017,  ...,  0.2555, -0.2649,  0.0501],
          [-0.1946,  0.3973,  0.2391,  ...,  0.0214, -0.3325,  0.3234],
          ...,
```

```
          [-0.0151,  0.0364,  0.0286,  ..., -0.2425, -0.0436, -0.1089],
          [ 0.0932,  0.2708,  0.2253,  ...,  0.2096,  0.1118,  0.1931],
          [ 0.1486,  0.3887,  0.3538,  ..., -0.0992, -0.1256,  0.1188]]],
        device='cuda:0')}
```

Save

```
import os
import pickle

wav_file = sample.split('/')[-1].replace('.wav', '')
main_folder = "my_model_actv"
output_dir = os.path.join(main_folder, model_name.split('/')[-1])
os.makedirs(output_dir, exist_ok=True)

wav_base_name = os.path.splitext(os.path.basename(wav_file))[0]
pkl_file_name = f"{wav_base_name}.pkl"
pkl_file_path = os.path.join(output_dir, pkl_file_name)
with open(pkl_file_path, 'wb') as file:
    pickle.dump(activation, file)



print(f"Salvo em: {pkl_file_path}")
```

```
Dicionário salvo em: my_model_actv/canary-1b/stim515_sports_anouncer.pkl
```

[Análise Regressão]

Comparação de Representações de Modelos de Redes Neurais e Respostas Cerebrais

O objetivo principal do artigo é entender em que medida as representações de redes neurais profundas (DNNs) replicam aspectos do processamento auditivo humano, comparando-as com respostas cerebrais medidas por fMRI. Para isso, são utilizadas duas abordagens matemáticas principais: a regressão linear regularizada e a análise de similaridade representacional (RDM). Essas metodologias permitem avaliar como as representações em modelos artificiais se alinham com as respostas cerebrais em regiões do córtex auditivo.

Regressão Linear Regularizada

Esta abordagem busca quantificar o quanto as ativações internas de um modelo de rede neural podem prever a variação nas respostas de voxels cerebrais quando expostos a estímulos sonoros naturais. Os sons são apresentados aos modelos, e as ativações das unidades em diferentes estágios são coletadas e agrupadas para cada estímulo, formando um conjunto de características temporais médias que representam as respostas do modelo.

Para cada voxel do cérebro, é ajustado um mapeamento linear entre as ativações do modelo e as respostas medidas no voxel, utilizando regressão ridge. Essa técnica adiciona uma penalização aos coeficientes do modelo para evitar sobreajustes, especialmente útil devido à alta dimensionalidade das ativações do modelo em relação aos dados de fMRI. O conjunto de estímulos é dividido em grupos de treinamento e teste, garantindo que as previsões sejam generalizáveis. A eficácia do modelo é medida pela variância explicada nas respostas dos voxels, indicando o quanto a combinação linear das ativações do modelo captura os padrões de variação nas respostas cerebrais.

Análise de Similaridade Representacional (RDM)

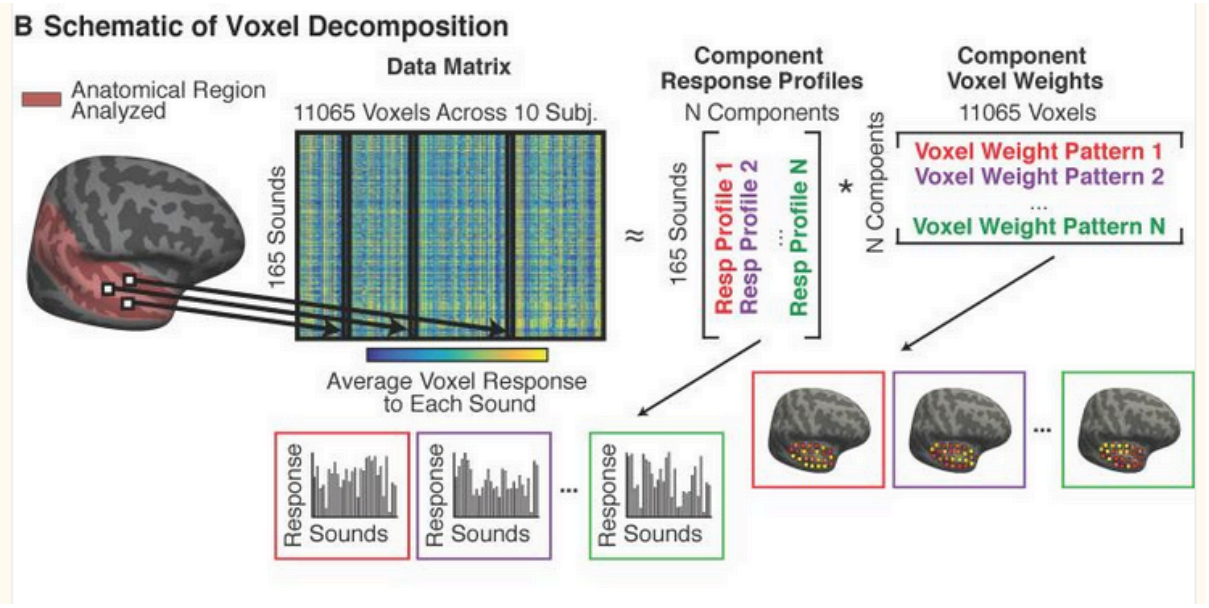
Enquanto a regressão linear mede a capacidade preditiva direta, a RDM avalia se as estruturas de resposta entre estímulos no modelo e no cérebro são similares, sem depender de mapeamentos lineares explícitos. Para cada conjunto de respostas (cerebrais ou do modelo), é calculada uma matriz de dissimilaridade, onde cada elemento representa a diferença entre as respostas para um par de estímulos, calculada como 1 menos a correlação de Pearson entre as respostas aos dois estímulos. A matriz do cérebro é comparada à matriz do modelo usando a correlação de Spearman, avaliando se a estrutura das representações é consistente entre o modelo e o cérebro. Para contextualizar os

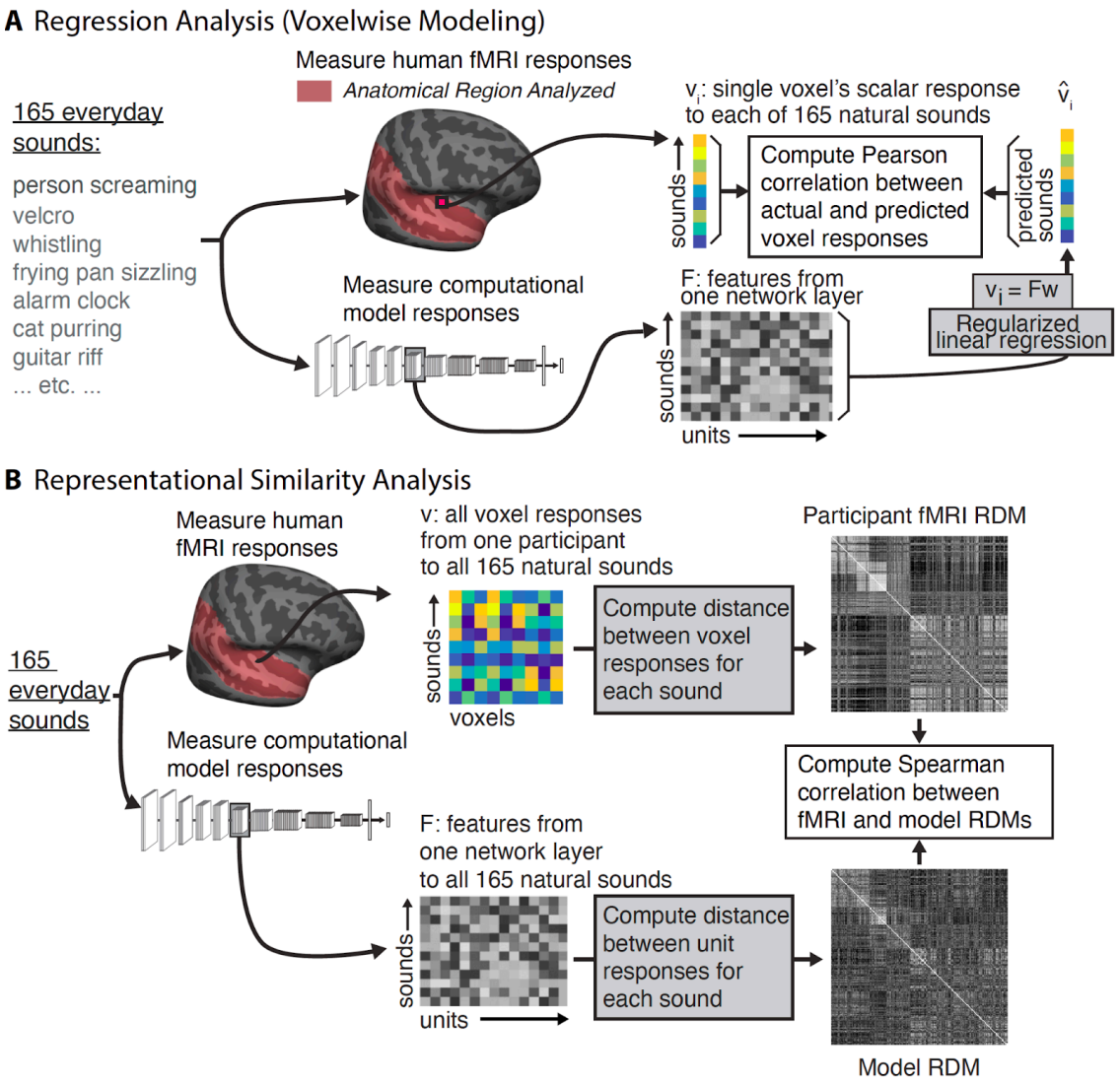
resultados, é estimado um teto de ruído que representa o máximo de similaridade esperado, dado o nível de variação entre diferentes participantes no experimento de fMRI.

Convergência das Abordagens

Embora distintas, ambas as abordagens fornecem insights complementares. A regressão linear mostra a capacidade preditiva direta das ativações de um modelo em capturar variações específicas nas respostas de voxels. Resultados robustos indicam que certas camadas de modelos, especialmente redes profundas treinadas para tarefas auditivas complexas, predizem com maior precisão as respostas de regiões não primárias do córtex auditivo. A RDM avalia a similaridade estrutural das representações sem depender de um mapeamento explícito, indicando que modelos bem ajustados replicam melhor a organização representacional observada no cérebro.

Resultados Principais





Os resultados demonstram que modelos treinados em tarefas auditivas ecologicamente relevantes, como reconhecimento de fala em ambientes ruidosos, não apenas predizem melhor as respostas cerebrais na regressão, mas também exibem maior similaridade estrutural com as representações cerebrais na RDM. Além disso, camadas intermediárias das redes tendem a predizer melhor o córtex auditivo primário, enquanto camadas mais profundas mostram maior correspondência com regiões auditivas não primárias. Esses achados corroboram a ideia de que as representações no córtex auditivo humano possuem uma organização hierárquica, com regiões distintas correspondendo a estágios diferentes de processamento em redes neurais profundas.

A relação entre os métodos de comparação e o conjunto de dados utilizado envolve um fluxo específico de processamento que vincula os dados cerebrais aos estímulos e às ativações

geradas pelas redes neurais. O conjunto de dados contém 165 sons naturais de 2 segundos, abrangendo diversas categorias como fala, música e sons ambientais. As respostas cerebrais são capturadas via fMRI na área do córtex auditivo de participantes durante a audição desses estímulos, e as ativações dos modelos são extraídas ao processar os mesmos sons, gerando representações em diferentes camadas que podem ser interpretadas como estágios hierárquicos do processamento auditivo.

Os sons são processados para produzir entradas apropriadas para os modelos, e as respostas cerebrais são resumidas em valores escalares representando a resposta média ao som, alinhando-se à temporalidade do fMRI. O conjunto de sons é dividido em subconjuntos de treinamento e teste para ajustar os parâmetros na regressão e para avaliar a qualidade das previsões ou similaridade na RDM.

A abordagem integrativa do conjunto de dados mostrou que a correspondência entre modelo e cérebro depende significativamente do tipo de tarefa para a qual o modelo foi treinado e da presença de ruído nos estímulos. Camadas intermediárias dos modelos predizem melhor o córtex auditivo primário, enquanto camadas mais profundas refletem melhor regiões não primárias. Esses resultados destacam a importância do conjunto de dados em garantir uma avaliação robusta da correspondência entre modelos de redes neurais e o processamento auditivo humano.

[regression_analysis.ipynb]

Setup

```
# !pip install seaborn==0.11.0
# !pip install matplotlib==3.3.3

from auditory_brain_dnn.aud_dnn.utils import *
from tqdm import tqdm
DATADIR = (Path(os.getcwd()) / '..' / 'data').resolve()
RESULTDIR = (Path(os.getcwd()) / '..' / 'results').resolve()
CACHEDIR = (Path(os.getcwd()) / '../..' /
'my_model_actv').resolve().as_posix()

from resources import source_layer_map

# Set random seed
import random
np.random.seed(0)
random.seed(0)

date = datetime.datetime.now().strftime("%m%d%Y-%T")

source_model = 'canary-1b'  # Nome do modelo DNN de origem
source_layer = 'preprocessor'  # Camada de origem do DNN
target = 'NH2015'  # Dados neurais ou de componente como alvo
alphalimit = 50  # Limite superior para alphas na regressão ridge
randnetw = 'False'  # Usar modelo permutado ('True') ou original ('False')
save_plot = 'False'  # Salvar gráficos diagnósticos ('True' ou 'False')
save_preds = 'True'  # Salvar predições por som em cada fold de validação
cruzada
verbose = False  # Imprimir progresso no console ('True' ou 'False')

identifier = (
    f'AUD-MAPPING-Ridge_'
    f'TARGET-{target}_'
    f'SOURCE-{source_model}-{source_layer}_'
    f'RANDNETW-{randnetw}_'
    f'ALPHALIMIT-{alphalimit}'
)

RESULTFOLDER = (Path(RESULTDIR / source_model / identifier))
(Path(RESULTFOLDER)).mkdir(exist_ok=True, parents=True)

PLOTFOLDER = False
if save_plot == 'True': # generate folder for plots
    PLOTFOLDER = (Path(RESULTFOLDER / 'diagnostic_plots'))
```

```
    (Path(PLOTFOLDER)).mkdir()

# Logging
if not verbose:
    sys.stdout = open(os.path.join(RESULTFOLDER,
                                   f'out-{date}.log'), 'a+')
```

Data

Stimuli

```
##### Stimuli (sounds) #####
sound_meta = np.load(os.path.join(DATADIR,
                                  f'neural/NH2015/neural_stim_meta.npy')) #
(original indexing, neural data is extracted in this order)

# Extract wav names in order (this is how the neural data is ordered -->
order all activations in the same way)
stimuli_IDs = []
for i in sound_meta:
    stimuli_IDs.append(i[0][:-4].decode("utf-8")) # remove .wav
```

Voxel

```
##### Load target (neural data or components) #####
voxel_data, voxel_id = get_target(target=target,
                          stimuli_IDs=stimuli_IDs,
                          DATADIR=DATADIR)
n_stim = voxel_data.shape[0]
n_vox = voxel_data.shape[1]

voxel_data.shape
```

```
(165, 7694, 3)
```

DNN Activations

```
source_features = get_source_features(source_model=source_model,
                                      source_layer=source_layer,
                                      source_layer_map=source_layer_map,
                                      stimuli_IDs=stimuli_IDs,
                                      randnetw=randnetw,
                                      CACHEDIR=CACHEDIR)

source_features.shape
```

```
(165, 25729)
```

CrossVal

```
##### Cross-validation #####

## Setup splits ##
n_CV_splits = 10
possible_alphas = [10 ** x for x in range(-alphalimit, alphalimit)]
possible_alphas = possible_alphas[::-1]
n_for_train = 83
n_for_test = 82
```

Regression

```
# store r-values for test and train (and alpha values)
r_voxels_test_prior_zero_manipulation = np.zeros([n_vox, n_CV_splits]) #
save r value without setting it to zero if it is negative or std = 0
r_voxels_test = np.zeros([n_vox, n_CV_splits])
r2_voxels_test = np.zeros([n_vox, n_CV_splits])
r2_voxels_test_corrected = np.zeros([n_vox, n_CV_splits])
r2_voxels_train = np.zeros([n_vox, n_CV_splits])
alphas = np.zeros([n_vox, n_CV_splits])

# store std of predicted and actual responses
y_pred_test_std_mean = np.zeros([n_vox, n_CV_splits]) # predicted
y_pred_train_std_mean = np.zeros([n_vox, n_CV_splits]) # predicted
y_test_std_mean = np.zeros([n_vox, n_CV_splits]) # neural
y_train_std_mean = np.zeros([n_vox, n_CV_splits]) # neural

# store warnings
warning_constant_mean = np.zeros([n_vox, n_CV_splits]) # whether there is a
warning for a constant prediction when fitting on the mean of the three
repetitions
warning_constant_splits = np.zeros([n_vox, n_CV_splits]) # whether there is
a warning for a constant prediction when fitting on just two repetitions
(for reliability estimation)
warning_alphas = np.zeros([n_vox, n_CV_splits]) # whether there is a
warning for hitting either upper or lower bound of alpha range

# store predictions
if save_preds == 'True':
    # store the predictions for each sound in the test set
    y_preds_test = np.zeros([n_stim, n_CV_splits, n_vox])
    y_preds_test[:] = np.nan # fill with nan, because we wont have every
value for every sound (given that some will be in the train set)

    y_preds_test_rescaled = np.zeros([n_stim, n_CV_splits, n_vox]) # if the
mean was subtracted during regression, this is a version of the predictions
with the mean added back in
```

```
    y_preds_test_rescaled[:] = np.nan


all_train_idxs = np.zeros([n_stim, n_CV_splits]) # same split across all
voxels
all_test_idxs = np.zeros([n_stim, n_CV_splits])

n_CV_splits

10

split_idx = 0

print(f'Running split {split_idx}\n')

# Randomly pick train/test indices
train_data_idxs = np.random.choice(n_stim, size=n_for_train, replace=False)
set_of_possible_test_idxs = set(np.arange(n_stim)) - set(train_data_idxs)
test_data_idxs = np.random.choice(list(set_of_possible_test_idxs),
size=n_for_test, replace=False)
is_train_data, is_test_data = np.zeros((n_stim), dtype=bool),
np.zeros((n_stim), dtype=bool)
is_train_data[train_data_idxs], is_test_data[test_data_idxs] = True, True

all_train_idxs[:, split_idx] = is_train_data.copy() # columns: splits -->
denotes whether stim was used as train data
all_test_idxs[:, split_idx] = is_test_data.copy()

track_vox_idx, voxel_idx = 0,0
r_voxels_test_prior_zero_manipulation[track_vox_idx, split_idx],\
r_voxels_test[track_vox_idx, split_idx], \
r2_voxels_test[track_vox_idx, split_idx],\
r2_voxels_test_corrected[track_vox_idx, split_idx],\
r2_voxels_train[track_vox_idx, split_idx],\
y_pred_test, y_pred_test_rescaled, \
alphas[track_vox_idx, split_idx],\
y_pred_test_std_mean[track_vox_idx, split_idx], \
y_pred_train_std_mean[track_vox_idx, split_idx], \
y_test_std_mean[track_vox_idx, split_idx],\
y_train_std_mean[track_vox_idx, split_idx], \
warning_constant_mean[track_vox_idx, split_idx], \
warning_constant_splits[track_vox_idx, split_idx], \
warning_alphas[track_vox_idx, split_idx] =
ridgeCV_correctedR2(source_features=source_features,

voxel_data=voxel_data,
```

```
voxel_idx=voxel_idx,

split_idx=split_idx,

is_train_data=is_train_data,

is_test_data=is_test_data,

possible_alphas=possible_alphas,

save_plot=PLOTFOLDER)
```

```
/opt/conda/lib/python3.11/site-packages/sklearn/linear_model/_ridge.py:2341
: FutureWarning: 'store_cv_values' is deprecated in version 1.5 and will be
removed in 1.7. Use 'store_cv_results' instead.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/linear_model/_ridge.py:2341
: FutureWarning: 'store_cv_values' is deprecated in version 1.5 and will be
removed in 1.7. Use 'store_cv_results' instead.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/linear_model/_ridge.py:2341
: FutureWarning: 'store_cv_values' is deprecated in version 1.5 and will be
removed in 1.7. Use 'store_cv_results' instead.
  warnings.warn(
/opt/conda/lib/python3.11/site-packages/sklearn/linear_model/_ridge.py:2341
: FutureWarning: 'store_cv_values' is deprecated in version 1.5 and will be
removed in 1.7. Use 'store_cv_results' instead.
  warnings.warn(
```

```
# Store predictions
if save_preds == 'True':
    # Append y_pred_test in the test indices, for the correct voxel, for
the correct split
    y_preds_test[is_test_data, split_idx, track_vox_idx] =
y_pred_test.ravel()
    y_preds_test_rescaled[is_test_data, split_idx, track_vox_idx] =
y_pred_test_rescaled.ravel()


sys.stdout.flush()

for split_idx in tqdm(range(n_CV_splits), desc="Processing CV splits"):
    print(f'Running split {split_idx}\n')

    # Randomly pick train/test indices
    train_data_idxs = np.random.choice(n_stim, size=n_for_train,
```

```
replace=False)
    set_of_possible_test_idxs = set(np.arange(n_stim)) -
set(train_data_idxs)
    test_data_idxs = np.random.choice(list(set_of_possible_test_idxs),
size=n_for_test, replace=False)
    is_train_data, is_test_data = np.zeros((n_stim), dtype=bool),
np.zeros((n_stim), dtype=bool)
    is_train_data[train_data_idxs], is_test_data[test_data_idxs] = True,
True

    all_train_idxs[:, split_idx] = is_train_data.copy() # columns: splits
--> denotes whether stim was used as train data
    all_test_idxs[:, split_idx] = is_test_data.copy()

    for track_vox_idx, voxel_idx in enumerate(range(n_vox)):
        if voxel_idx % 100 == 1:
            print(f'Running voxel number: {voxel_idx}')

        # Run components: not possible to obtain corrected value
        if target == 'NH2015comp':
            r_voxels_test_prior_zero_manipulation[track_vox_idx,
split_idx],\
            r_voxels_test[track_vox_idx, split_idx], \
            r2_voxels_test[track_vox_idx, split_idx],\
            r2_voxels_train[track_vox_idx, split_idx],\
            y_pred_test, y_pred_test_rescaled, \
            alphas[track_vox_idx, split_idx],\
            y_pred_test_std_mean[track_vox_idx, split_idx], \
            y_pred_train_std_mean[track_vox_idx, split_idx], \
            y_test_std_mean[track_vox_idx, split_idx],\
            y_train_std_mean[track_vox_idx, split_idx], \
            warning_constant_mean[track_vox_idx, split_idx], \
            warning_alphas[track_vox_idx, split_idx] =
ridgeRegressSplits(source_features=source_features,

y=voxel_data.to_numpy()[:, voxel_idx][:, None],

is_train_data=is_train_data,

is_test_data=is_test_data,

possible_alphas=possible_alphas,

voxel_idx=voxel_idx,

split_idx=split_idx,)
```

```
        # Run neural data with correction
        else:
            r_voxels_test_prior_zero_manipulation[track_vox_idx,
split_idx],\
            r_voxels_test[track_vox_idx, split_idx], \
            r2_voxels_test[track_vox_idx, split_idx],\
            r2_voxels_test_corrected[track_vox_idx, split_idx],\
            r2_voxels_train[track_vox_idx, split_idx],\
            y_pred_test, y_pred_test_rescaled, \
            alphas[track_vox_idx, split_idx],\
            y_pred_test_std_mean[track_vox_idx, split_idx], \
            y_pred_train_std_mean[track_vox_idx, split_idx], \
            y_test_std_mean[track_vox_idx, split_idx],\
            y_train_std_mean[track_vox_idx, split_idx], \
            warning_constant_mean[track_vox_idx, split_idx], \
            warning_constant_splits[track_vox_idx, split_idx], \
            warning_alphas[track_vox_idx, split_idx] =
ridgeCV_correctedR2(source_features=source_features,

voxel_data=voxel_data,

voxel_idx=voxel_idx,

split_idx=split_idx,

is_train_data=is_train_data,

is_test_data=is_test_data,

possible_alphas=possible_alphas,

save_plot=PLOTFOLDER)

        # Store predictions
        if save_preds == 'True':
            # Append y_pred_test in the test indices, for the correct
voxel, for the correct split
            y_preds_test[is_test_data, split_idx, track_vox_idx] =
y_pred_test.ravel()
            y_preds_test_rescaled[is_test_data, split_idx, track_vox_idx] =
y_pred_test_rescaled.ravel()


        sys.stdout.flush()
```

```
            # if track_vox_idx == 5:
            #     break

    ## LOAD METADATA ##
    df_roi_meta = pd.read_pickle(os.path.join(DATADIR,
f'neural/{target}/df_roi_meta.pkl'))

    ## CV SPLITS ##
    dict_splits = {'all_train_idxs': all_train_idxs,
                   'all_test_idxs': all_test_idxs}

    ## SAVE RESULTS ACROSS CV SPLITS ##
    vox_idx_coord = np.arange(n_vox)

    ds = xr.Dataset(
        {"r_prior_zero": (("vox_idx", "split_idx"),
r_voxels_test_prior_zero_manipulation),
         "r_test": (("vox_idx", "split_idx"), r_voxels_test),
         "r2_test": (("vox_idx", "split_idx"), r2_voxels_test),
         "r2_test_c": (("vox_idx", "split_idx"), r2_voxels_test_corrected),
         "r2_train": (("vox_idx", "split_idx"), r2_voxels_train),
         "alphas": (("vox_idx", "split_idx"), alphas),
         "y_pred_test_std_mean": (("vox_idx", "split_idx"),
y_pred_test_std_mean),
         "y_pred_train_std_mean": (("vox_idx", "split_idx"),
y_pred_train_std_mean),
         "y_test_std_mean": (("vox_idx", "split_idx"), y_test_std_mean),
         "y_train_std_mean": (("vox_idx", "split_idx"), y_train_std_mean),
         "warning_constant_mean": (("vox_idx", "split_idx"),
warning_constant_mean),
         "warning_constant_splits": (("vox_idx", "split_idx"),
warning_constant_splits),
         "warning_alphas": (("vox_idx", "split_idx"), warning_alphas),
         },
        coords={
            "vox_idx_coord": vox_idx_coord,
            "vox_idx": vox_idx_coord,
            "voxel_id": voxel_id,
            "split_idx_coord": np.arange(10),
            "split_idx": np.arange(10),
            "source_model": ("vox_idx_coord", [str(source_model)]*n_vox),
            "source_layer": ("vox_idx_coord", [str(source_layer)]*n_vox),
            "randnetw": ("vox_idx_coord", [str(randnetw)]*n_vox)},
        attrs=dict_splits)
```

```
    ## SAVE MEANED RESULTS ##
    # Assert that no nans exist in r_voxels_test, r2_voxels_test,
r2_voxels_test_corrected
    assert np.isnan(r_voxels_test).sum() == 0
    assert np.isnan(r2_voxels_test).sum() == 0
    assert np.isnan(r2_voxels_test_corrected).sum() == 0

    # Create a df with num voxels as rows, and columns with metrics of
interest
    df_results = pd.DataFrame({'mean_r_prior_zero_test':
np.nanmean(r_voxels_test_prior_zero_manipulation, 1),
                               'median_r_prior_zero_test':
np.nanmedian(r_voxels_test_prior_zero_manipulation, 1),
                               'mean_r_test': np.mean(r_voxels_test, 1),
                               'median_r_test': np.median(r_voxels_test,
1),
                               'mean_r2_test': np.mean(r2_voxels_test, 1),
                               'median_r2_test': np.median(r2_voxels_test,
1),
                               'std_r2_test': np.std(r2_voxels_test, 1),
                               'mean_r2_test_c':
np.mean(r2_voxels_test_corrected, 1),
                               'median_r2_test_c':
np.median(r2_voxels_test_corrected, 1),
                               'std_r2_test_c':
np.std(r2_voxels_test_corrected, 1),
                               'mean_r2_train': np.mean(r2_voxels_train,
1),
                               'median_r2_train':
np.median(r2_voxels_train, 1),
                               'std_r2_train': np.std(r2_voxels_train, 1),
                               'median_alpha': np.median(alphas, 1),
                               'mean_alpha': np.mean(alphas, 1),
                               })

    # Use the voxel_id as index
    df_results['voxel_id'] = voxel_id
    df_results = df_results.set_index('voxel_id', drop=False,
inplace=False)
    df_results['vox_idx_coord'] = vox_idx_coord

    # concatenate with metadata
    assert(voxel_id == df_roi_meta.voxel_id.values).all() # Assert that the
meta and the results are in the same order (voxel_id)
    df_roi_meta = df_roi_meta.set_index('voxel_id', drop=True,
inplace=False) # otherwise we end up with two identical voxel_id columns
```

```
    assert(df_results.index == df_roi_meta.index).all() # Once more
assertion that the meta and the results are in the same order (voxel_id)

    df_output = pd.concat([df_results, df_roi_meta], axis=1)

    ## Log ##
    df_output['source_model'] = source_model
    df_output['source_layer'] = source_layer
    df_output['randnetw'] = randnetw
    df_output['target'] = target

    ## Save ##
    df_output.to_pickle(os.path.join(RESULTFOLDER, 'df_output.pkl'))
    pickle.dump(ds, open(os.path.join(RESULTFOLDER, 'ds.pkl'), 'wb'))
    if save_preds == 'True':
        pickle.dump(y_preds_test, open(os.path.join(RESULTFOLDER,
'y_preds_test.pkl'), 'wb'))
        pickle.dump(y_preds_test_rescaled, open(os.path.join(RESULTFOLDER,
'y_preds_test_rescaled.pkl'), 'wb'))
        # Indexed according to stimuli_IDs, and vox order is the same as in
the df_roi_metadata

    print(f'Saved results to {RESULTFOLDER}')
```

```
Processing CV splits:   0%|
| 0/10 [00:00<?,
```

APÊNDICE 6

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 4 de dez. de 2024

Participantes da Entrega [matriculados em Residência em IA]:

MARCELO HENRIQUE LOPES FERREIRA

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

A entrega final contempla os resultados obtidos nos testes realizados durante a Residência.

Foram gerados gráficos que demonstram a similaridade entre os modelos testados em comparação com a baseline:
https://drive.google.com/drive/folders/1C7lnPo_rVnwiRDdId9WvKuVh6JRZ5yPb?usp=drive_link

Todo o material de estudos desenvolvidos ao longo da Residência foram digitalizados:
https://drive.google.com/drive/folders/1bdsR7SR-xe5ZBEchbN5IqgOa0Cro7mqd?usp=drive_link

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

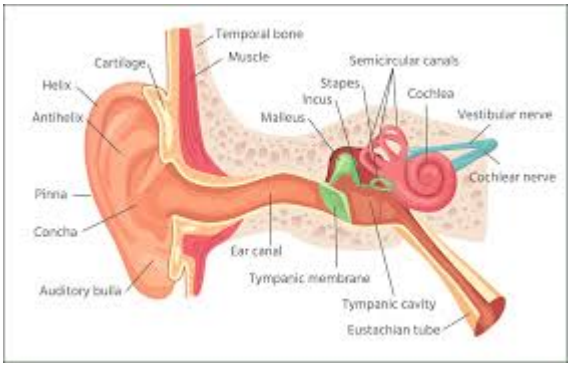
Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

CEDRIC LUIZ DE CARVALHO: Go!

[Anotações estudos sobre audição]

Percepção Auditiva

→ Recepção do Som no Ouvido



1 Estrutura do Ouvido

O ouvido humano é dividido em três partes principais:

- **Ouvido Externo**: Composto pelo pavilhão auricular (orelha) e o canal auditivo externo, sua função é captar e direcionar as ondas sonoras para o tímpano.
- **Ouvido Médio**: Contém o tímpano e os três ossículos auditivos (martelo, bigorna e estribo), que amplificam e transmitem as vibrações sonoras para o ouvido interno.
- **Ouvido Interno**: Abriga a cóclea, responsável pela transdução das vibrações mecânicas em impulsos nervosos, e o sistema vestibular, relacionado ao equilíbrio.

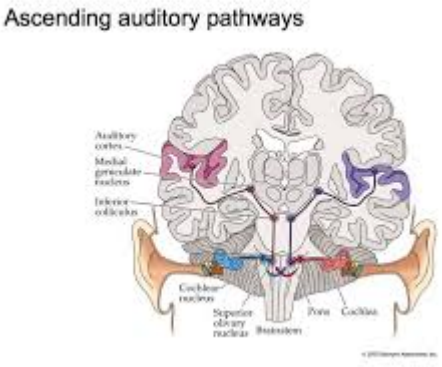
2 Transdução na Cóclea

A cóclea é uma estrutura em forma de espiral preenchida com fluido e revestida pela **membrana basilar**. Sobre essa membrana está o **Órgão de Corti**, que contém células sensoriais chamadas **células ciliadas**.

- **Células Ciliadas Internas**: São as principais responsáveis pela transdução auditiva. Quando o fluido coclear vibra, elas são deslocadas, abrindo canais iônicos que geram potenciais de ação nos neurônios auditivos.
- **Células Ciliadas Externas**: Amplificam e ajustam as vibrações, melhorando a sensibilidade e a seletividade auditiva.

A posição na cóclea onde as vibrações são mais intensas depende da frequência do som, um fenômeno conhecido como **tonotopia**.

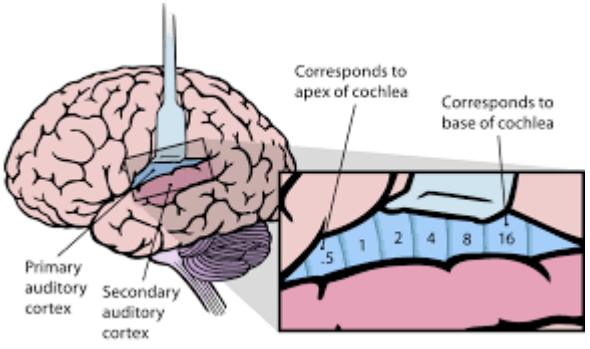
→ Transmissão para o Cérebro



Os potenciais de ação gerados nas células ciliadas são transmitidos pelo **nervo auditivo** até o cérebro. O caminho auditivo inclui várias sinapses em diferentes núcleos:

- **Núcleo Coclear**: Primeiro ponto de processamento no tronco encefálico.
- **Complexo Olivar Superior**: Importante para a localização do som, processando diferenças de tempo e intensidade entre os ouvidos.
- **Colículo Inferior**: Integra informações auditivas e participa de reflexos auditivos.
- **Corpo Geniculado Medial**: Parte do tálamo que retransmite sinais para o córtex auditivo.

→ Processamento no Cérebro



1 Córtex Auditivo Primário (A1)

Localizado no lobo temporal, o A1 é organizado de forma tonotópica, refletindo a organização frequencial da cóclea. Neurônios em diferentes áreas respondem a diferentes frequências sonoras.

2 Vias Auditivas

- **Via Ventral ("O quê")**: Envolve o reconhecimento de padrões sonoros complexos, como a fala e a música.
- **Via Dorsal ("Onde")**: Relacionada à localização espacial dos sons e ao processamento temporal.

3 Processamento de Características Sonoras

O cérebro interpreta diversos aspectos do som:

- **Frequência (Tom)**: Determina o quão agudo ou grave é um som.
- **Intensidade (Volume)**: Codificada pela amplitude das vibrações e a taxa de disparo dos neurônios.
- **Localização Sonora**: Baseada em diferenças de tempo e intensidade com que o som chega a cada ouvido.

→ Experimentos Notáveis

Georg von Békésy e a Teoria da Onda Viajante

Georg von Békésy, laureado com o Prêmio Nobel em 1961, investigou como a cóclea responde a diferentes frequências sonoras.

- **Descobertas Principais**:
  - Demonstrou que as ondas sonoras criam uma **onda viajante** na membrana basilar da cóclea.
  - A posição de máxima vibração depende da frequência:
    - **Frequências Altas**: Máxima vibração próxima à base da cóclea.
    - **Frequências Baixas**: Máxima vibração próxima ao ápice.

Lloyd Jeffress e a Localização Sonora

Lloyd Jeffress propôs um modelo para explicar como o cérebro determina a direção de um som.

- **Modelo de Coincidência Temporal**:
  - Sugere que neurônios específicos disparam quando recebem sinais simultâneos de ambos os ouvidos.
  - As diferenças de tempo de chegada do som a cada ouvido são usadas para calcular a localização.

Wilder Penfield e o Mapeamento Cortical

Neurocirurgião que, durante cirurgias em pacientes conscientes, estimulou diferentes áreas do cérebro.

- **Contribuições**:
  - Mapeou áreas do córtex auditivo e suas funções.
  - Pacientes relataram sensações auditivas específicas quando certas áreas eram estimuladas.

→ Plasticidade Auditiva e Períodos Críticos

Desenvolvimento Auditivo

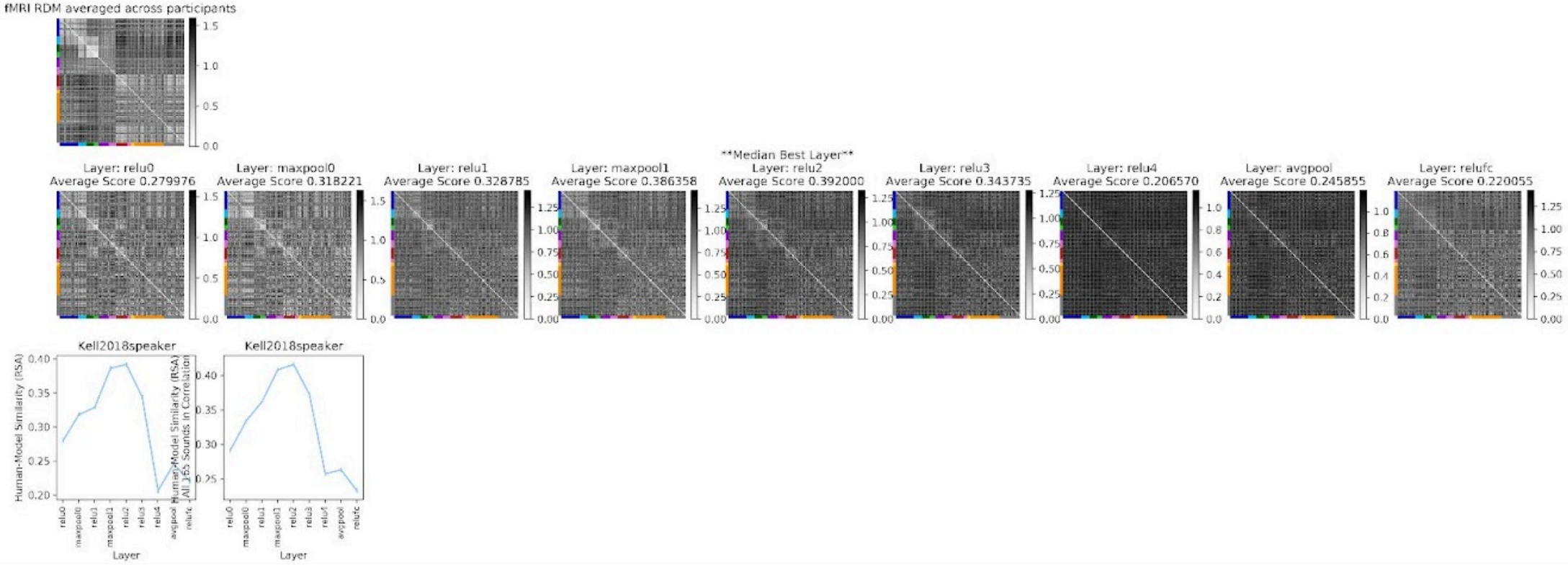
- **Períodos Críticos**: Fases no desenvolvimento em que o sistema auditivo é especialmente sensível a estímulos.
- **Privação Auditiva**: Falta de estímulos sonoros nesses períodos pode levar a déficits permanentes na percepção auditiva.

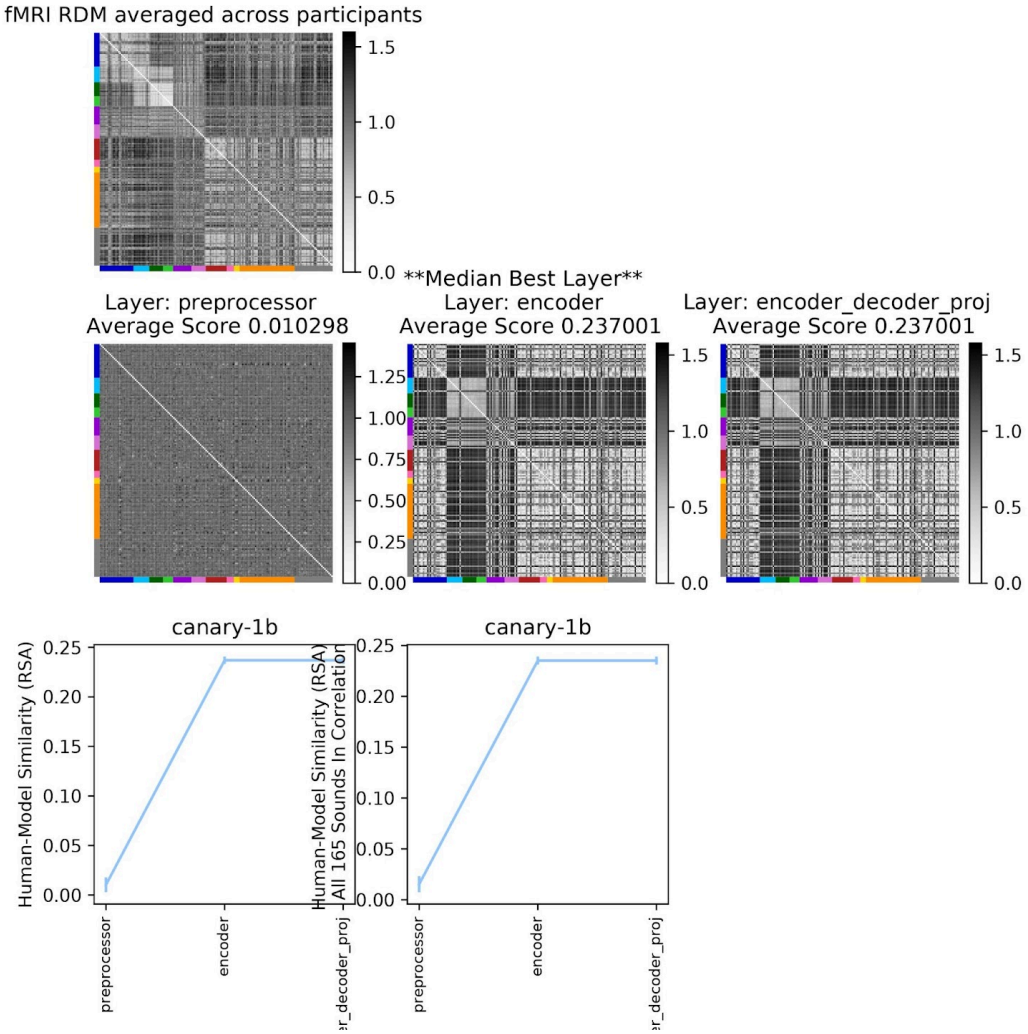
Plasticidade Neural

- O sistema auditivo pode reorganizar-se em resposta a danos ou alterações ambientais.
- **Exemplo**: Indivíduos com perda auditiva podem apresentar maior sensibilidade em outras modalidades sensoriais.

A percepção auditiva é essencial para a comunicação e interação com o ambiente. Desde a física das ondas sonoras até o processamento neural, cada etapa é crucial para a interpretação dos sons. Os experimentos de pesquisadores como Georg von Békésy, Lloyd Jeffress e Wilder Penfield ampliaram significativamente nosso entendimento desse sistema complexo, permitindo avanços na medicina e tecnologia que melhoram a qualidade de vida de muitas pessoas.

[Resultados]


fMRI RDM averaged across participants
Layer: relu0
Average Score 0.279976
Layer: maxpool0
Average Score 0.318221
Layer: relu1
Average Score 0.328785
Layer: maxpool1
Average Score 0.386358
Median Best Layer
Layer: relu2
Average Score 0.392000
Layer: relu3
Average Score 0.343735
Layer: relu4
Average Score 0.206570
Layer: avgpool
Average Score 0.245855
Layer: relufc
Average Score 0.220055
Kell2018speaker
Human-Model Similarity (RSA)
Human-Model Similarity (RSA) All_165 Sounds in Correlation
Layer
Kell2018speaker
Layer


fMRI RDM averaged across participants
1.5
1.0
0.5
0.0
Median Best Layer
Layer: preprocessor
Average Score 0.010298
Layer: encoder
Average Score 0.237001
Layer: encoder_decoder_proj
Average Score 0.237001
1.25
1.00
0.75
0.50
0.25
0.00
canary-1b
canary-1b
Human-Model Similarity (RSA)
Human-Model Similarity (RSA)
All 165 Sounds In Correlation
0.25
0.20
0.15
0.10
0.05
0.00
preprocessor
encoder