UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

DISCIPLINA: REDES NEURAIS - DEEP LEARNING

PROFESSOR: ADRIÃO DUARTE DORIA NETO

3ª LISTA DE EXERCÍCIO – 2025.1

ALUNO: WILSON FRANCELINO DE MORAIS JÚNIOR

Natal, julho/2025.

Questão 1

Apresente um estudo sobre a máquina de aprendizagem transformer considerando as seguintes aplicações. Obs. escolha dois dos itens abaixo.

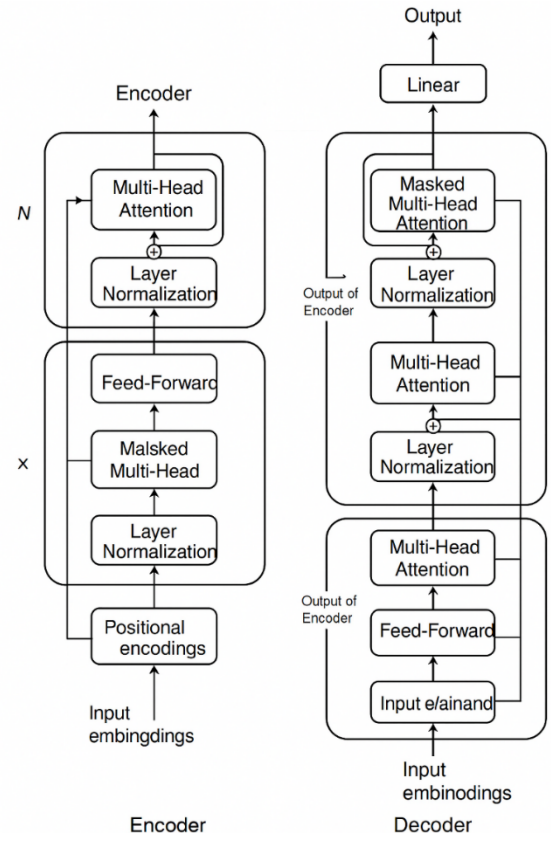
a) Visão Computacional (Vision transformer)
b) Geração de Imagens (Generative image transformers)
c) Processamento de Áudio (Audio data)
d) Conversão Texto - Voz (Text-to-speech)
e) Linguagem Natural (LLM)
f) Visão e Linguagem (Vision and language transformers)
g) Outra aplicação de livre escolha

Resposta

1. INTRODUÇÃO AOS *TRANSFORMERS*

O *Transformer* é uma arquitetura de aprendizado profundo baseada inteiramente em mecanismos de atenção, projetada para tarefas de processamento de linguagem natural. Introduzido no artigo "*Attention Is All You Need*" (2017), o modelo substitui estruturas sequenciais como *RNNs* e *LSTMs* por mecanismos de autoatenção e atenção multi-cabeça, permitindo paralelização total e captura eficiente de dependências contextuais de longo alcance. A entrada textual é representada por *embeddings* vetoriais somados a codificações posicionais, que preservam a ordem sequencial dos tokens. Cada camada do *encoder* aplica atenção própria (*self-attention*), seguida por uma rede *feedforward* posicionada com normalização e conexões residuais, facilitando o fluxo de gradientes e a estabilidade durante o treinamento.

No *decoder*, o *Transformer* utiliza atenção mascarada para garantir a geração autoregressiva, e adiciona uma etapa de atenção cruzada que conecta as representações do *encoder* às saídas geradas. A arquitetura é composta por múltiplas camadas idênticas empilhadas, com o *encoder* responsável por codificar a entrada e o decoder por gerar sequências de saída *token* a *token*. Ao final, uma camada linear seguida de *softmax* produz as distribuições de probabilidade sobre o vocabulário. Essa estrutura altamente modular e paralelizável tornou-se base de modelos avançados como BERT, GPT e T5, estabelecendo um novo paradigma em tarefas de PLN e, mais recentemente, estendendo seu uso a domínios como visão computacional e bioinformática.



1.1. Embedding + Positional Encoding

Word Embedding: Cada *token* de entrada (palavra ou subpalavra) é convertido em um vetor denso $\boldsymbol{x}_i \in \mathbb{R}^d$ usando uma matriz de embedding $E \in \mathbb{R}^{V \times d}$, onde V é o vocabulário e d é a dimensão do *embedding*.

Positional Encoding: Como o *Transformer* não é sequencial, é necessário informar a ordem dos *tokens*. Para tanto, usa-se um vetor de codificação posicional $\mathbf{p}_i \in \mathbb{R}^{V \times d}$, somado ao *embedding*: $\mathbf{z}_i = \mathbf{x}_i + \mathbf{p}_i$.

O método original usa funções senoidais:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \quad ; \quad \mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

1.2. Self-Attention (Atenção Escalada com Produto Escalar)

Para cada *token*, o modelo aprende a "atentar" sobre todos os outros *tokens* com base em três vetores:

- ***Query*:** $\mathbf{q}_i = \mathbf{z}_i W^Q$
- ***Key*:** $\mathbf{k}_i = \mathbf{z}_i W^K$
- ***Value*:** $\mathbf{v}_i = \mathbf{z}_i W^V$

Para toda a sequência *n* de *tokens*:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\boldsymbol{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

Onde:

- $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{V \times d_k}$,
- d_k é a dimensão das *queries/keys* (usado na normalização)
- A *softmax* garante que os pesos atribuídos às palavras somem 1.

1.3. Multi-Head Attention

Em vez de realizar uma única atenção, o *Transformer* executa várias em paralelo:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_2)W^O$$

Cada cabeça é:

$$\text{Head}_i = \text{Attention}(\mathbf{Q}W_i^Q, \mathbf{K}W_i^K, \mathbf{V}W_i^V)$$

Onde:

- h: número de cabeças de atenção
- As projeções $\mathbf{Q}W_i^Q, \mathbf{K}W_i^K, \mathbf{V}W_i^V \in \mathbb{R}^{d \times d_k}, W^O \in \mathbb{R}^{hd_k \times d}$

Isso permite que o modelo aprenda **diferentes relações contextuais** ao mesmo tempo.

1.4. Feedforward Layer (FFN)

Cada posição da sequência é passada por uma MLP (com as mesmas camadas para todas as posições):

$$\text{FFN}(x) = \text{ReLU}(\text{x}W_1 + \text{b}_1)W_2 + b_2$$

Com dimensões típicas:

- $W_1 \in \mathbb{R}^{d \times d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \times d}$
- Usualmente $d_{ff} = 2048, d = 512$

1.5. Residual Connection + Layer Normalization

Cada sub-bloco (atenção ou FFN) é envolvido por um **resíduo + normalização**:

$$\text{LayerNorm}\big(x + \text{SubLayer}(x)\big)$$

Ajuda a estabilizar e acelerar o treinamento, além de facilitar o fluxo de gradientes.

1.6. Empilhamento de Camadas

Um **Transformer Encoder** é formado por *N* blocos idênticos (ex: 6 no *Transformer* original), cada um com:

- Multi-Head Attention
- Feedforward
- Residual + LayerNorm em ambos

1.7. Decoder (somente em tarefas de geração)

- Semelhante ao *encoder*, mas com:
  - **Masked Multi-Head Attention** para impedir que um *token* "veja" *tokens* futuros
  - Uma atenção cruzada: o *decoder* também atenta sobre a saída do *encoder*

1.8. Saída e Geração

- A última camada do *decoder* gera vetores que são passados por uma camada linear e softmax:

$$\hat{y_i} = \text{softmax}(W^T \cdot \mathbf{h}_i + b)$$

- A palavra mais provável é selecionada (ou amostrada) como saída.

2. APLICAÇÕES

A seguir serão apresentados dois estudos que fazem a aplicação dos transformes em reconhecimento de falar e conversão de voz em texto respectivamente.

TRANSFORMER-TRANSDUCER: END-TO-END SPEECH RECOGNITION WITH SELF-ATTENTION

Ching-Feng Yeh, Jay Mahadeokar, Kaustubh Kalgaonkar, Yongqiang Wang, Duc Le, Mahaveer Jain, Kjell Schubert, Christian Fuegen, Michael L. Seltzer

1. Introdução

O artigo *TRANSFORMER-TRANSDUCER: END-TO-END SPEECH RECOGNITION WITH SELF-ATTENTION* apresenta diferentes sistemas de reconhecimento de fala, comparando os modelos tradicionais híbridos (baseados em DNN-HMM) com abordagens end-to-end. É mostrado que, embora os sistemas híbridos tenham obtido bons resultados, eles exigem modelagem separada de componentes e alto custo computacional. Abordagens end-to-end, como RNN-T (*Recurrent Neural Network Transducer*), surgem como alternativas mais simples e eficientes. O trabalho propõe o uso de Transformers no lugar de RNNs dentro do framework *transducer*, visando melhorar paralelização, precisão e eficiência computacional.

2. *Neural Transducer* (RNN-T)

RNN-T é uma arquitetura *end-to-end* que modela a relação entre entrada e saída com comprimentos variáveis, superando as limitações do CTC ao incorporar histórico de saída na predição. A estrutura possui um *encoder* que processa a sequência de entrada e um *decoder* que prevê símbolos com base em saídas anteriores não-vazias, além de um *joiner* que combina os vetores do *encoder* e do *predictor*. RNNs como LSTM são comumente usados, mas neste artigo os autores propõem substituí-los por *Transformers*, permitindo computação paralela e melhor captação de contexto.

3. *Transformer*

Transformers utilizam o mecanismo de *self-attention*, que modela relações contextuais diretamente entre os elementos da sequência sem conexões recorrentes. Isso permite paralelização e captação eficiente de dependências de longo prazo. O artigo descreve a *self-attention* escalada com produto escalar e a extensão para *multi-head attention*, onde múltiplas projeções paralelas capturam diferentes subespaços de representação. O *encoder Transformer* é composto por blocos de atenção, *feedforward* e normalização em camadas (*LayerNorm*), complementados com *dropout*.

4. *Transformer-Transducer*

4.1. *Context Modeling with Causal Convolution*

Transformers não são naturalmente sensíveis à posição, o que é problemático em modelagens sequenciais como a de fala. Para contornar isso, os autores empregam convoluções causais (sem olhar para o futuro) com arquitetura *VGGNet* para incorporar informação posicional e reduzir a taxa de quadros, otimizando a eficiência da inferência. As saídas são repassadas ao *encoder Transformer*, formando o que chamam de "*VGG-Transformer*".

4.2. *Truncated Self-Attention*

Uma *Self-attention* ilimitada impõe alta complexidade $O(T^2)$ e inviabiliza inferência em streaming. A solução proposta é o uso de *self-attention* truncada, onde cada estado htht depende apenas de um intervalo fixo de contexto (passado e futuro limitados). Isso reduz a complexidade para $O(T)$ e permite aplicações com baixa latência, como reconhecimento de fala embarcado.

5. Experimentos

5.1. *Corpus and Setup*

Os experimentos são realizados no corpus *LibriSpeech* (960h de dados), com extração de características acústicas (*Mel-filterbanks*), normalização e uso de data *augmentation* (*SpecAugment*). Um vocabulário de 256 símbolos é treinado via *SentencePiece*. O treinamento é distribuído (32 GPUs) e a inferência usa *beam search* (largura 10).

5.2. *Model Architectures and Details*

Foram comparadas diferentes combinações de *encoders* (LSTM, BLSTM, *Transformer*) e *predictors* (LSTM, *Transformer*). O *encoder Transformer* tem 12 camadas, e o *predictor Transformer* tem 6, ambos com estruturas VGGNet na entrada. A redução de *frames* e *pooling* são aplicados para otimizar o desempenho e eficiência.

5.3. *Results on Transformer/LSTM Combinations*

Resultados mostram que o *encoder Transformer* com *predictor* LSTM é o mais eficiente: alcança **6,08% WER** (*word error rate*) no conjunto *test-clean* e **13,89%** no *test-other*, superando LSTM e BLSTM. No entanto, LSTM ainda é melhor como predictor, pois o uso de Transformer nessa função prejudica a performance e não aproveita a paralelização na decodificação.

5.4. *Results on Truncated Self-Attention*

Avaliações com diferentes janelas de contexto *(L,R)* mostraram que há um bom *trade-off* entre desempenho e latência. Com L=32 e R=4, o sistema alcança **6,37% (*clean*)** e *15,30% (other)* de WER — desempenho comparável ao uso de self-attention completa, mas com vantagens de ser ***streamable*** **e eficiente (complexidade O(T)).**

6. Conclusão

O *Transformer-Transducer* proposto combina *VGG-Transformer* como *encoder* com LSTM como *predictor*, empregando convolução causal e atenção truncada para reduzir latência e complexidade. Os resultados obtidos superam as arquiteturas anteriores de RNNs, mantendo o modelo leve (45,7M parâmetros) e eficiente para uso embarcado, como em dispositivos móveis.

ROBUST AND UNBOUNDED LENGTH GENERALIZATION IN AUTOREGRESSIVE TRANSFORMER-BASED TEXT-TO-SPEECH

Eric Battenberg RJ Skerry-Ryan Daisy Stanton Soroosh Mariooryad Matt Shannon
Julian Salazar David Kao

1. Introdução

Modelos Transformer autoregressivos (AR) são poderosos para tarefas de sequência-para-sequência, mas sofrem com falhas de robustez e generalização de comprimento em tarefas de Text-to-Speech (TTS), frequentemente omitindo ou repetindo palavras em entradas longas. O artigo ROBUST AND UNBOUNDED LENGTH GENERALIZATION IN AUTOREGRESSIVE TRANSFORMER-BASED TEXT-TO-SPEECH apresenta propõe o Very Attentive Tacotron (VAT), um modelo encoder-decoder Transformer discreto que incorpora um mecanismo de alinhamento monotônico aprendido via backpropagation, dispensando alinhamentos forçados. Isso permite que o VAT mantenha a capacidade expressiva do Transformer, com estabilidade em entradas de comprimentos arbitrariamente grandes

2. Trabalhos Relacionados

O artigo revisa abordagens como VALL-E, ELLA-V, MQ-TTS e VALL-T, que tentam corrigir os problemas de robustez com mecanismos baseados em duração, transducers e restrição de atenção. No entanto, esses modelos limitam o poder expressivo dos Transformers ou impõem alto custo computacional. O VAT diferencia-se por introduzir um único ponto de alinhamento monotônico que guia todas as operações de cross-attention multi-head, preservando a capacidade expressiva e generalizando melhor em comprimento

3. *Very Attentive Tacotron (VAT)*

3.1. Arquitetura do Sistema

O VAT baseia-se no T5 Transformer com encoder de atenção e decoder AR. Usa um vocoder GAN e codificação espectral via VQ-VAE com 8 códigos categóricos por quadro. Cada locutor tem um embedding treinável. A discretização com VQ-VAE permite adaptar a taxa de bits sem reconfigurar o codec completo

3.2. *Relative Position Biases (RPBs)*

Transformers tradicionais como o T5 utilizam viés de posição relativa (RPB – Relative Position Bias) para permitir que a atenção distinga posições na sequência. Esses vieses são definidos para distâncias relativas discretas agrupadas em “buckets” que cobrem intervalos específicos (ex.: −1, −2, …, +2, +3, …, até um máximo D). Isso limita a capacidade do modelo de interpolar entre posições não inteiras, o que é crítico para tarefas de síntese onde a duração de cada token pode variar de forma contínua.

No VAT, os autores reconhecem que os RPBs tradicionais são inadequados para tarefas de TTS, especialmente quando o modelo precisa aprender um alinhamento denso e contínuo entre texto e espectrograma, e propõem uma extensão mais flexível para lidar com esse cenário.

3.3. *Interpolated Relative Position Biases (IRPBs)*

Para resolver a limitação dos buckets discretos, o artigo propõe os Interpolated Relative Position Biases (IRPBs). Nesse mecanismo, em vez de mapear uma distância relativa $d=i-j$ diretamente para um bucket, o IRPB computa uma interpolação linear entre os dois buckets vizinhos. Por exemplo, se a posição relativa calculada é $d=4,6$, o modelo interpola entre os buckets 4 e 5 com pesos $(1-0,6)$ e 0,6, respectivamente.

Essa interpolação permite:

- Aprender vieses posicionais contínuos e diferenciáveis, suportando treinamento com gradientes suaves;
- Melhor generalização a sequências mais longas ou com variações sutis de alinhamento;
- Suporte natural a atenção com base em alinhamento não inteiro, o que é essencial para converter texto em espectrogramas discretizados.

A interpolação também é útil para reconstrução precisa da geometria de atenção em casos em que o modelo está extrapolando além dos comprimentos vistos durante o treinamento. Esse mecanismo foi fundamental para o VAT obter alinhamento robusto e fluido ao longo de longas sequências.

3.4. Alinhamento Monotônico

O modelo incorpora uma camada de alinhamento LSTM (256 unidades) para prever posições contínuas de alinhamento entre texto e áudio. Essa camada usa atenção baseada apenas em posição, sem conteúdo, o que confere estabilidade e monotonicidade à geração.

3.5. Atenção Cruzada com IRPB

A operação de cross-attention no decoder é informada pelo alinhamento aprendido. A atenção considera posições relativas contínuas e usa IRPBs para reforçar relações estruturais entre texto e espectrograma, especialmente úteis para entrada longa e repetitiva.

3.6. Inicialização Gaussiana

Para facilitar a aprendizagem do alinhamento desde as primeiras etapas, os IRPBs são inicializados com distribuições gaussianas centradas nas janelas esperadas de atenção.

3.7. Penalidade de Distância Máxima (MDP)

Para controlar a influência de relações muito distantes, é aplicada uma penalidade de distância nas IRPBs quando $|d|$ excede D, garantindo que o modelo se concentre em

regiões relevantes e evite comportamentos erráticos em testes com sequências maiores do que as vistas no treino.

4. Experimentos

4.1. Configuração

O encoder possui convoluções residuais (2 camadas) seguidas de self-attention não causal. O decoder usa 6 blocos Transformer com 16 heads e largura 1024. A camada de alinhamento usa um LSTM de 256 unidades e atenção local com 4 cabeças. O VQ-VAE opera com taxa de 40 Hz e bitrate de 2.56 kbps.

4.2. Dados

Foram usados dois conjuntos: (1) um dataset interno multisspeaker (670h) e (2) o LibriTTS clean-460 (213h). O baseline Tacotron-GMMA foi treinado com voz única (Lessac), enquanto os modelos T5 e VAT foram treinados com múltiplos locutores.

4.3. Treinamento

Modelos foram treinados por 650k etapas com otimizador Adam e ajuste de taxa de aprendizagem. A versão LibriTTS teve redução de escala (3/8 do tamanho original) para evitar overfitting. O alinhamento do VAT foi inicializado para refletir a taxa média esperada de avanço temporal entre texto e espectrograma.

4.4. Avaliação

As métricas incluem:

- MOS (Mean Opinion Score);
- SxS (Side-by-Side comparison);
- CER via ASR (robustez e generalização de comprimento);

Teste de palavras repetidas (modelos tendem a errar contagens ou repetir trechos inteiros).

5. Resultados

5.1. Naturalidade

VAT obteve MOS ≈ 3,68 (Lessac) e ≈ 3,16 (LibriTTS), levemente abaixo do T5. Mas superou o Tacotron em SxS e expressividade subjetiva.

5.2. Robustez via ASR

O VAT apresentou menor CER que o T5, especialmente em entradas longas (CER do T5 atinge 10,7 no LibriTTS; VAT = 4.6). O modelo NAT (duração fixa) foi ainda mais robusto, mas perdeu expressividade.

5.3. Generalização de Comprimento

T5 falha rapidamente após os 9.6s vistos no treino. Já VAT e NAT mantêm desempenho estável até 90 segundos (1500 caracteres). O VAT demonstra boa retenção de palavras e estruturas sintáticas complexas.

5.4. Palavras Repetidas

Em frases com repetições, VAT teve 0 erros em 27 frases. O T5 falhou em 52% das vezes, às vezes gerando dezenas de repetições inesperadas.

6. Discussão

O VAT combina a expressividade dos *Transformers* com a estabilidade de alinhamentos monotônicos. Suporta comprimentos arbitrários sem perda de coerência ou naturalidade. A arquitetura é extensível a outras tarefas com alinhamento monotônico (como ASR) e oferece um novo caminho para TTS robusto e escalável. Os principais desafios residem na complexidade de implementação e tempo de treinamento ligeiramente superior (≈12–20% a mais que o T5).

Questão 2

Desenvolva um trabalho sobre Aprendizagem por Reforço Profundo (*Deep Reinforce-ment Learnig*) considerando aplicações da técnica.

Resposta

APRENDIZAGEM POR REFORÇO PROFUNDO

A aprendizagem por reforço (Reinfo*rcement Learning – RL*) é uma das ferramentas fundamentais dentro do campo da inteligência artificial (IA) e do aprendizado de máquina (*machine learning*), caracterizando-se pela capacidade de um agente aprender a tomar decisões sequenciais em ambientes dinâmicos e incertos por meio da interação direta com o ambiente. Diferentemente das abordagens supervisionadas, nas quais há um conjunto explícito de exemplos com rótulos, na aprendizagem por reforço o agente aprende com base em um sistema de recompensas e penalidades, obtidas como resultado das ações realizadas ao longo do tempo.

O objetivo principal do agente no *RL* é maximizar a recompensa acumulada, ou seja, aprender uma política de ação ótima que determine qual ação tomar em cada estado do ambiente, de forma a obter o maior retorno possível a longo prazo. Esse processo de aprendizagem ocorre por tentativa e erro, sendo inicialmente conduzido por ações aleatórias e, progressivamente, refinado à medida que o agente descobre estratégias mais eficientes. Um exemplo clássico dessa abordagem consiste em treinar um agente para alcançar uma recompensa evitando obstáculos em um ambiente simulado, onde ações corretas são reforçadas positivamente e ações equivocadas recebem penalidades.

A aprendizagem por reforço é modelada por meio de Processos de Decisão de Markov (*Markov Decision Processes – MDPs*), que descrevem o ambiente em termos de estados, ações, funções de transição de estados e funções de recompensa. A política aprendida é representada como uma função de decisão que associa, a cada estado, uma probabilidade de escolher determinada ação. O valor de uma política é medido pelo retorno esperado, ou seja, a soma das recompensas futuras descontadas ao longo do tempo.

Um *MDP* é definido pelo quíntuplo (S,A,P,R,γ), onde:

- $\boldsymbol{S}$ é o **conjunto de estados** do ambiente;
- $\boldsymbol{A}$ é o **conjunto de ações** possíveis para o agente;
- $\boldsymbol{P}(\boldsymbol{s'}|\boldsymbol{s},\boldsymbol{a})$ é a **função de transição**, que fornece a probabilidade de transitar ao estado $\boldsymbol{s'}$ após executar a ação $\boldsymbol{a}$ no estado $\boldsymbol{s}$;

- ***R*(*s*,*a*)** é a **função de recompensa** imediata, que define o ganho esperado ao executar ***a*** em ***s***;
- $\gamma \in [0,1]]$ é o **fator de desconto**, que determina a importância de recompensas futuras.

A propriedade de Markov afirma que a evolução do ambiente depende apenas do estado atual e da ação tomada, não do histórico anterior. A meta do agente é aprender uma política ótima $\pi^* : S \rightarrow A$ que maximize o valor esperado do **retorno** (soma de recompensas futuras descontadas):

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Duas funções são fundamentais nesse contexto:

- Função de valor de estado: $V^{\pi} = \mathbb{E}[G_t | S_t = s, \pi]$
- Função de valor de ação: $Q^{\pi} = \mathbb{E}[G_t | S_t = s, A_t = a, \pi]$

A política ótima π^* é tal que:

$$\pi^* = \arg\max_a Q^*(s, a)$$

A aprendizagem por reforço também se integra a outras áreas da IA. Ela é um subconjunto do aprendizado de máquina e frequentemente se beneficia do uso de técnicas de *deep learning*, sobretudo na chamada **Aprendizagem por Reforço Profundo (*Deep Reinforcement Learning – Deep RL*), que utiliza redes neurais profundas para aproximar funções de valor e políticas em ambientes com espaços de estados ou ações de alta dimensionalidade.**

A aprendizagem por reforço profundo resolve a limitação da escalabilidade dos métodos clássicos utilizando redes neurais profundas como aproximadores de funções para $Q(s,a)$, $V(s)$ ou diretamente para a política $\pi(a|s)$. Dentre as técnicas de utilizadas na aprendizagem por reforço profundo podemos destacar o ***Deep Q-Network* (*DQN*)** e os **Métodos Baseados em Gradientes de Política**.

O algoritmo ***Deep Q-Network* (*DQN*)**, proposto por Mnih em 2015, foi pioneiro na integração de redes neurais convolucionais com *RL*. Nesse modelo, uma rede neural estima a função $Q(s,a;\theta)$, onde θ são os pesos da rede. Dois mecanismos fundamentais foram introduzidos para estabilizar o aprendizado:

- **Experience Replay**: armazenamento de experiências (s, a, r, s') em um *buffer* e treinamento por amostragem aleatória;
- **Rede-alvo**: uso de uma segunda rede neural congelada temporariamente, usada para calcular os alvos de *Q-learning*.

O DQN obteve desempenho sobre-humano em diversos jogos do Atari 2600, aprendendo diretamente a partir de pixels brutos.

Já nos **Métodos Baseados em Gradientes de Política**, em vez de estimar valores de estado ou ação, estima-se diretamente a política:

$$\nabla \boldsymbol{J}(\theta) = \mathbb{E}_{\pi\theta}[\nabla_\theta \log \pi_\theta(a|s) \boldsymbol{Q}^{\pi_\theta}(s,a)]$$

Isso permite tratar espaços contínuos e políticas estocásticas. Exemplos:

A aprendizagem por reforço profundo tem sido amplamente utilizada em aplicações de destaque, como o treinamento de agentes para jogos (Go, Atari, Xadrez), controle de robôs, sistemas de recomendação e planejamento autônomo. Um marco importante foi o desenvolvimento do AlphaGo pela DeepMind, que demonstrou o potencial da aprendizagem por reforço combinada a técnicas de aprendizado profundo e busca heurística.

Entre os principais desafios da aprendizagem por reforço destacam-se:

- A criação de ambientes simulados fidedignos, especialmente em contextos de aplicação do mundo real como veículos autônomos ou robótica industrial;
- A escalabilidade dos modelos, dada a dificuldade de projetar redes neurais capazes de generalizar a partir de sinais de recompensa esparsos;
- A risco de convergência a ótimos locais, em que o agente aprende estratégias subótimas ou até mesmo formas de "burlar" o sistema de recompensas sem, de fato, resolver o problema proposto.

Para exemplificar a aplicação da aprendizagem por reforço profundo é apresentado a seguir uma análise do artigo ***"Aprendizagem por Reforço Profundo com Redes Convolucionais Aplicada à Navegação Autônoma de Robôs Reais Utilizando Treinamento em Cenário Virtual"***.

APRENDIZAGEM POR REFORÇO PROFUNDO COM REDES CONVOLUCIONAIS APLICADA À NAVEGAÇÃO AUTÔNOMA DE ROBÔS REAIS UTILIZANDO TREINAMENTO EM CENÁRIO VIRTUAL

Carlos Daniel de Sousa Bezerra; Flávio Henrique Teles Vieira - Universidade Federal de Goi´as (UFG)

I. Introdução

O trabalho analisado contextualiza o surgimento da Indústria 4.0, que demanda soluções de automação baseadas em tecnologias inteligentes como IoT, IA, e robótica móvel. Destaca-se o uso crescente de robôs móveis autônomos (AMRs) como solução adaptável para tarefas repetitivas e de monitoramento em ambientes industriais. A Inteligência Artificial, especialmente o Aprendizado por Reforço Profundo, é apresentada como um meio de permitir que agentes robóticos aprendam a se adaptar a ambientes variáveis por meio de interação e feedback. O problema central abordado é a dificuldade de **aplicar esse aprendizado em robôs reais sem danificá-los no processo de aprendizagem**, sendo proposta uma abordagem *Sim-to-Real* para resolver essa limitação.

II. Trabalhos Relacionados

É apresentado uma revisão da literatura sobre o uso de robôs móveis e aprendizado por reforço em aplicações reais e simuladas. Destacam-se protótipos de baixo custo e controladores robustos em ambientes incertos. A seção enfatiza a dificuldade na transferência de aprendizado do simulador para o robô real, devido a limitações como a variabilidade física e sensorial entre plataformas. Estudos recentes discutem a relevância de novas técnicas de simulação e transferência de políticas para mitigar essas diferenças, apontando o trabalho como parte dessa fronteira de pesquisa.

III. Aprendizado por Reforço Profundo com CNN

O algoritmo *Deep Q-Network (DQN)*, conforme diagrama de bloco da figura 1, substitui as tradicionais tabelas Q por redes neurais profundas, tornando possível o aprendizado em ambientes com espaços de estado complexos (como imagens RGB captadas por câmeras). O robô é treinado para permanecer em uma pista circular. A função de recompensa é desenhada para incentivar a redução da distância ao objetivo, penalizar colisões e bonificar sucessos.

A função $Q(s,a)$ é uma variação do algoritmo de aprendizado temporal proposto por Bellman e é dada por:

$$Q^{\pi}(s,a) = r + \gamma \max Q^{\pi}(s',a')$$

Onde r é a recompensa imediata obtida e γ é um fator de desconto para recompensas futuras, s, s', a e a' são os estados e ações presentes e futuros, respectivamente.

A função de recompensa, que guia o processo de aprendizado por reforço, dado por:

$$R = \begin{cases} dist_{k-1} - dist_k & \text{se ativo} \\ -1, & \text{se colidir} \\ +1, & \text{se atingir o alvo} \end{cases}$$

Onde $dist_k$ e $dist_{k-1}$ são, respectivamente, as distâncias do robô até o alvo no instante k.

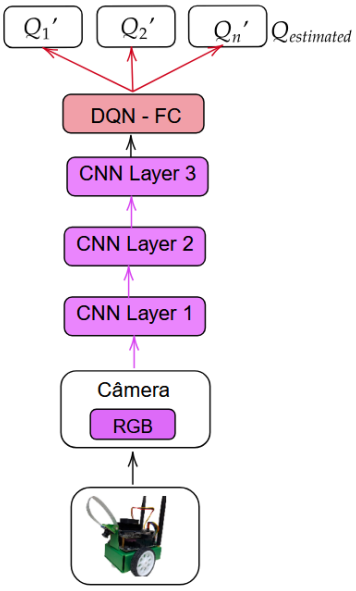


Fig. 1. Diagrama de Blocos da Rede Neural DQN Utilizada.

A CNN extrai características visuais das imagens de entrada, enquanto as camadas densas mapeiam essas características em três ações: virar à direita, à esquerda ou seguir em frente. A arquitetura da CNN utilizada é detalhada em termos de filtros, tamanhos de *kernel* e *stride*.

TABLE I
ESTRUTURA DA REDE CNN.

CNN Layer	Filters	Kernel	Stride
1st Layer	16	(5, 5)	5
2nd Layer	32	(3, 3)	2
3rd Layer	32	(2, 2)	2

Rede DQN-CNN com Transferência de Aprendizado

Aqui é proposto o modelo de treinamento *sim-to-Real*, na qual o agente treinado no simulador Coppelia V-REP é transferido para um robô real. O processo envolve:

1) configuração do simulador;

2) escolha do robô (modelo diferencial);

3) execução do algoritmo de navegação; e

4) programação e adaptação dos atuadores no robô físico (Jetbot).

Por serem plataformas distintas, ajustes como velocidades e tempos de amostragem foram necessários. O objetivo era verificar se o comportamento aprendido em ambiente virtual permanecia eficaz no ambiente real.

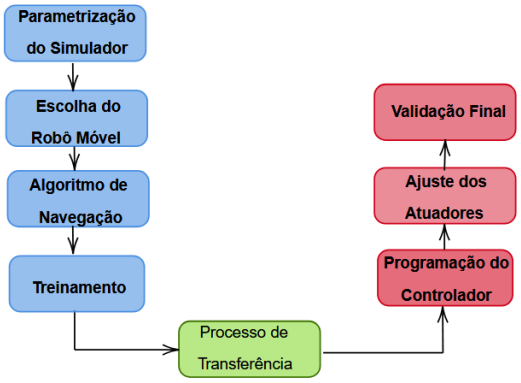


Fig. 2. Proposta de Transferência de Aprendizado.

IV. Ambientes de Simulação e Teste

Apresenta-se o simulador Coppelia V-REP como ambiente de treinamento. O robô é treinado para completar uma volta em pista virtual usando o algoritmo DQN. A vantagem é que erros durante o aprendizado não causam danos ao sistema físico.

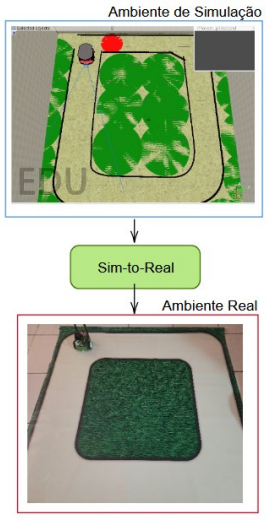


Fig. 3. Proposta de Transferência de Aprendizado.

O robô utilizado no mundo real é o Jetbot, com Jetson Nano, que oferece boa capacidade de processamento e integração com Python. A câmera RGB embutida fornece dados visuais para o algoritmo. A arquitetura completa do sistema é apresentada e detalha-se a compatibilidade entre o ambiente de simulação e o robô físico.

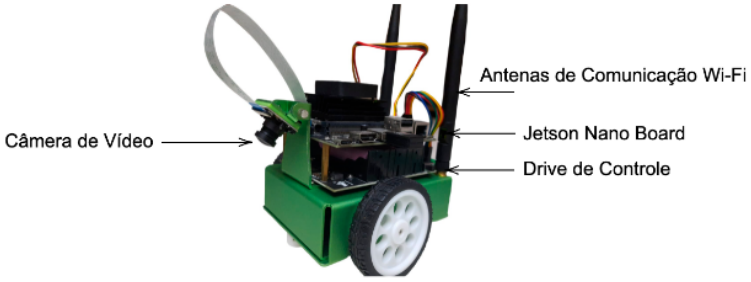


Fig. 4. Robô móvel com Placa de Desenvolvimento Jetson Nano.

V. Resultados e Discussões

Os resultados de simulação demonstram uma curva de aprendizado ascendente em 500 episódios, com média final de recompensa de 17,51 (de um máximo de ~20) e taxa de sucesso de 60%.

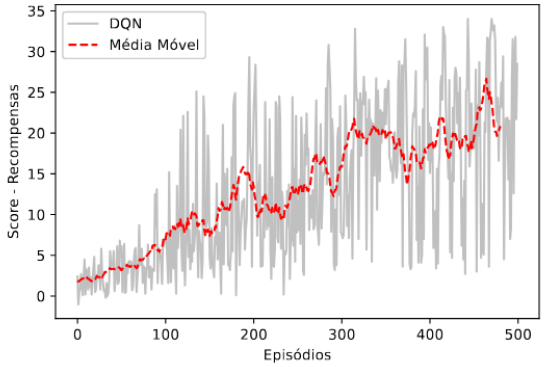


Fig. 5. Curva de Aprendizado do Algoritmo DQN - 500 Episódios.

A métrica de taxa de sucesso indica a proporção de episódios bem-sucedidos em relação ao total de episódios simulados. A métrica de recompensa média representa a média das recompensas obtidas ao longo dos episódios simulados.

TABLE II
RESULTADOS DOS TESTES DA REDE DQN-CNN

Rede DQN	Média de Recompensas Final	Taxa de Sucesso Final
Teste 1	15,06	0,4
Teste 2	17,00	0,4
Teste 3	17,40	0,8
Teste 4	19,26	0,8
Teste 5	16,82	0,6
Média Final	17,51	0,6

O treinamento foi eficaz em fazer o robô completar a tarefa. A transferência para o robô real mostrou que, mesmo com diferenças visuais e físicas, o robô foi capaz de executar a mesma missão com sucesso.

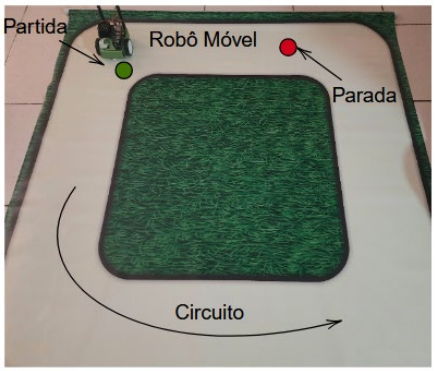


Fig. 6. Frames capturados de movimentação do Robô.

A movimentação foi precisa, e a navegação dentro dos limites da pista foi alcançada. As imagens do robô em movimento demonstram a eficácia do processo *Sim-to-Real*.

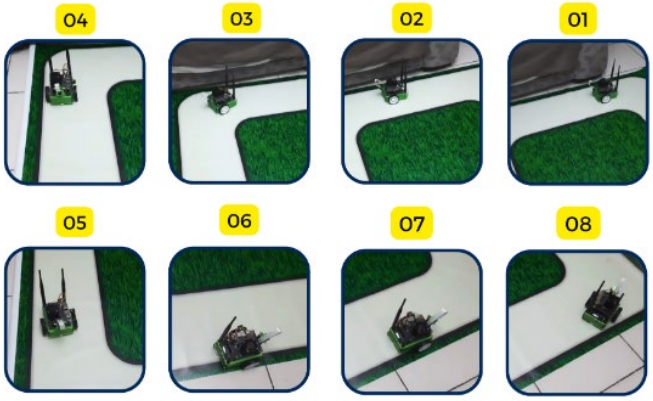


Fig. 7. Frames capturados de movimentação do Robô.

VI. Conclusões

A proposta de aplicação do DQN com CNN para navegação autônoma demonstrou-se eficaz tanto em ambiente virtual quanto real. A metodologia permitiu transferência segura e funcional do aprendizado do simulador para o robô Jetbot. A curva de aprendizado confirma a eficiência da abordagem, e a execução bem-sucedida da tarefa no mundo real valida a robustez da solução.

Questão 3

Considere os dados apresentados na tabela abaixo. Determine os centroides dos aglomerados *"clusters"* presentes nos dados, fazendo uso do algoritmo da rede competitiva que corresponde ao algoritmo *K-means*. Para tanto considere os itens (a) - (c) referentes ao processo de inicialização.

Amostra	**x1**	**x2**	**x3**
1	7,82	-4,58	-3,97
2	-6,68	3,16	2,71
3	4,36	2,19	2,09
4	6,72	0,88	2,80
5	-8,64	-3,06	3,50
6	-6,87	0,57	-5,45
7	4,47	-2,62	5,76
8	6,73	-2,01	4,18
9	-7,71	2,34	-6,33
10	-6,91	-0,49	-5,68
11	6,18	-2,81	5,82
12	6,72	-0,93	-4,04
13	-6,25	-0,26	0,56
14	-6,94	-1,22	1,13
15	8,09	0,20	2,25
16	6,81	0,17	-4,15
17	-5,19	4,24	4,04
18	-6,38	-1,74	1,43
19	4,08	1,30	5,33
20	6,27	0,93	-2,78

a) Considere que existam três clusters e a inicialização dos centros seja aleatória.
b) Considere que existam três clusters e a inicialização dos centros seja dada por $\mathbf{m_1} = (0, 0, 0)^t$, $\mathbf{m_2} = (0,1,1)^t$, $\mathbf{m_3} = (-1, 1, 2)^t$.
c) Repita o item a considerando que os centros iniciais sejam $\mathbf{m_1} = (-0.1, 0.0, 0.1)^t$, $\mathbf{m_2} = (0.0, -0.1, 0.1)^t$, $\mathbf{m_3} = (-0.1, -0.1, 0.1)^t$. Compare o resultado obtido com o item (a) e explique a razão da diferenças, incluindo o número de interações para alcançar a convergência.

Resposta

O algoritmo *K-means* é um método de aprendizado de máquina não supervisionado usado para dividir um conjunto de dados em K grupos ou *clusters* distintos. O objetivo é minimizar a soma das distâncias quadradas entre cada ponto de dados e o centroide (ponto médio) do *cluster* ao qual ele é atribuído.

O algoritmo começa escolhendo aleatoriamente K pontos de dados para serem os centroides iniciais dos *clusters*. Cada ponto de dados é atribuído ao *cluster* cujo centroide está mais próximo, utilizando uma medida de distância (geralmente a distância euclidiana). Após atribuir todos os pontos, os centroides são recalculados como a média dos pontos de dados pertencentes a cada *cluster*. As etapas são repetidas iterativamente até que os centroides não mudem mais ou um número máximo de iterações seja atingido.

a) Na figura 1 são mostrados os três *clusters* produzido pelo algoritmo *K-means* quando a inicialização dos centros foi aleatória e o deslocamento dos centroides de cada um desses *clusters* ao longo das três iterações até a convergência.

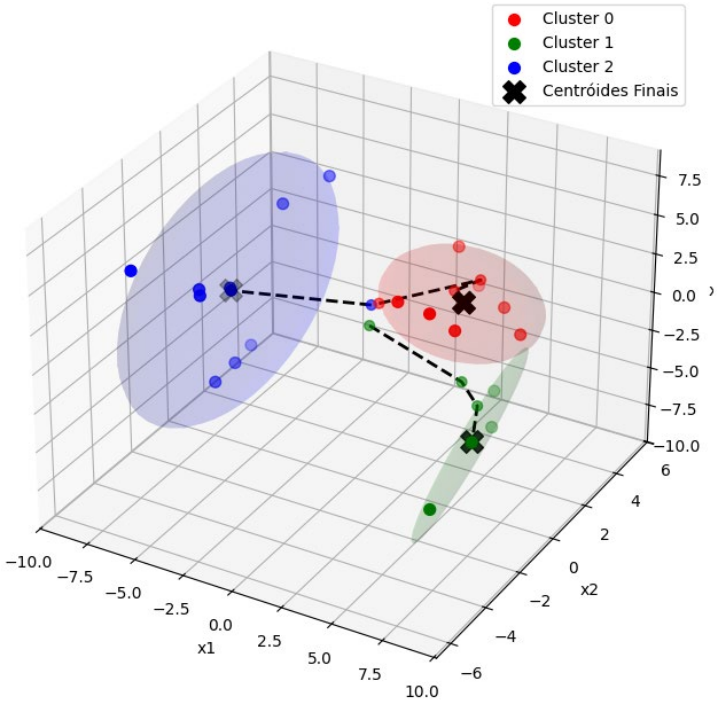


Figura 1 - Clusters produzido pelo *K-means* com inicialização aleatória.

b) Na figura 2 são mostrados os três *clusters* produzido pelo algoritmo *K-means* quando a inicialização dos centros foi dada pelos pontos $\mathbf{m_1} = (0, 0, 0)^t$, $\mathbf{m_2} = (0,1,1)^t$, $\mathbf{m_3} = (-1, 1, 2)^t$ e o deslocamento dos centroides de cada um desses *clusters* ao longo das três iterações até a convergência.

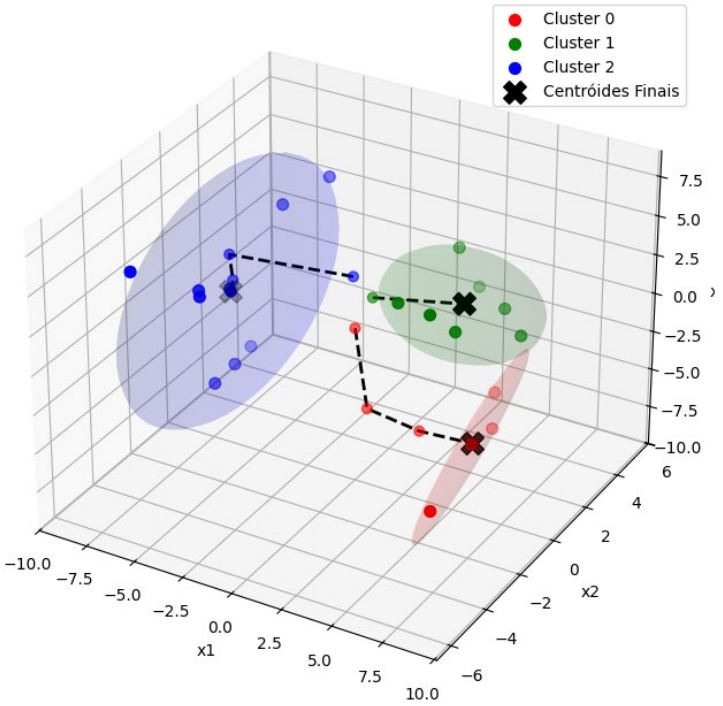


Figura 2 - Clusters produzido pelo *K-means* com inicialização em $\mathbf{m_1} = (0, 0, 0)^t$, $\mathbf{m_2} = (0,1,1)^t$, $\mathbf{m_3} = (-1, 1, 2)^t$.

c) Na figura 3 são mostrados os três *clusters* produzido pelo algoritmo *K-means* quando a inicialização dos centros foi dada pelos pontos $\mathbf{m_1} = (-0.1, 0.0, 0.1)^t$, $\mathbf{m_2} = (0.0, -0.1, 0.1)^t$, $\mathbf{m_3} = (-0.1, -0.1, 0.1)^t$ e o deslocamento dos centroides de cada um desses *clusters* ao longo das três iterações até a convergência.

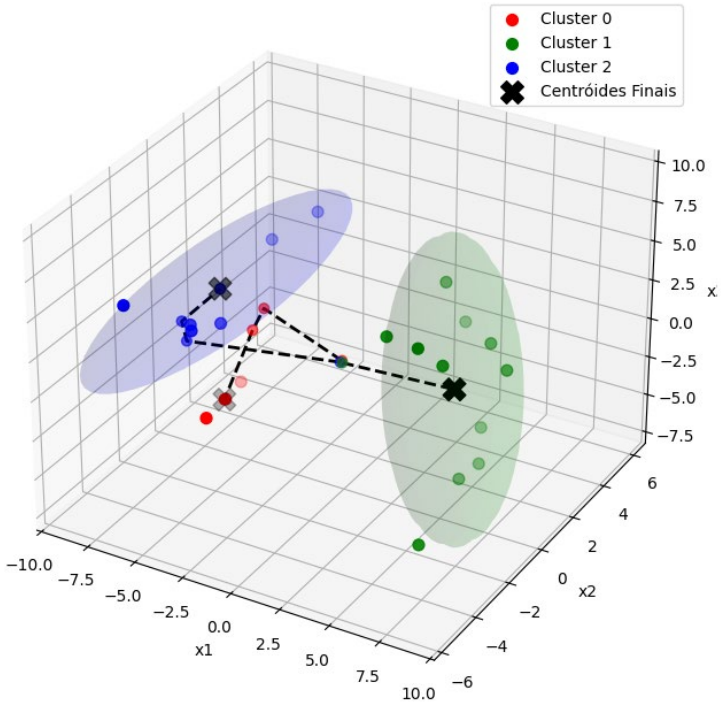


Figura 3 - Clusters produzido pelo *K-means* com inicialização em $\mathbf{m_1} = (-0.1, 0.0, 0.1)^t$, $\mathbf{m_2} = (0.0, -0.1, 0.1)^t$, $\mathbf{m_3} = (-0.1, -0.1, 0.1)^t$.

Da análise dos resultados obtidos pode-se verificar que dependendo do ponto de inicialização dos centroides o algoritmo *K-means* produz *clusters* diferentes. Os centroides de todos os *clusters* convergiram com o mesmo número de iterações, mas poderia ter sido diferente dependendo dos pontos gerados aleatoriamente.

O código das simulações está disponível no seguinte link:

https://github.com/wilsonfmjr/PPGEEC2321---REDES-NEURAIS-E-DEEP-LEARNING/blob/main/LISTA_3/Questao_3.ipynb

Questão 4

A propriedade de ordenação topológica do algoritmo SOM pode ser usada para formar uma representação bidimensional abstrata para fins de visualização de um espaço de entrada de alta dimensionalidade. O objetivo é visualizar os dados de dimensão 8 em um espaço de dimensão 2, constituído pela grade de neurônios. Para investigar esta forma de representação, considere uma grade bidimensional de neurônios que é treinada tendo como entrada os dados oriundos de quatro distribuições gaussianas, C_1, C_2, C_3, e C_4, em um espaço de entrada de dimensionalidade igual a oito, isto é $\mathbf{x} = (x_1, x_2, \ldots, x_8)^t$. Todas as nuvens têm variâncias unitária, mas centros ou vetores média diferentes dados por $\mathbf{m_1} = (0,0,0,0,0,0,0,0)^t$, $\mathbf{m_2} = (4,0,0,0,0,0,0,0)^t$, $\mathbf{m_3} = (0,0,0,4,0,0,0,0)^t$, $\mathbf{m_4} = (0,0,0,0,0,0,0,4)^t$. Calcule o mapa produzido pelo algoritmo SOM, e verifique como as distribuições dos dados estão representadas.

Resposta

O *Self-Organizing Map* (SOM), proposto por Teuvo Kohonen, é um algoritmo de aprendizado não supervisionado baseado em redes neurais artificiais, voltado principalmente para a visualização, agrupamento e redução de dimensionalidade de dados multidimensionais. O princípio fundamental do SOM é a projeção de dados de alta dimensão em um espaço de menor dimensão, geralmente bidimensional, de forma que a topologia dos dados originais seja preservada. Isso significa que dados similares no espaço original serão mapeados para regiões próximas no mapa SOM.

O algoritmo funciona por meio de uma grade regular de neurônios, dispostos geralmente em uma malha 2D, onde cada neurônio está associado a um vetor de pesos de mesma dimensão que os dados de entrada. Inicialmente, esses vetores de pesos são atribuídos com valores aleatórios ou amostrados dos dados.

Durante o treinamento, cada vetor de entrada é apresentado à rede, e calcula-se a distância (normalmente Euclidiana) entre esse vetor e todos os vetores de pesos dos neurônios. O neurônio cujo vetor de pesos for o mais próximo do vetor de entrada é denominado *Best Matching Unit* (BMU). Em seguida, o vetor de pesos do BMU, bem como os de seus vizinhos dentro de um raio definido, são ajustados para se aproximarem do vetor de entrada. Essa atualização é controlada por uma taxa de aprendizagem decrescente e por uma função de vizinhança (como uma Gaussiana), que também decresce ao longo do tempo, tanto em amplitude quanto em raio.

A fórmula de atualização dos pesos é dada por:

$$w_i(t+1) = w_i(t) + \alpha(t) \cdot h_{bi}(t) \cdot \big(x(t) - w_i(t)\big)$$

Onde $w_i(t)$ representa o vetor de pesos do neurônio i no tempo t, $x(t)$ é o vetor de entrada, $\alpha(t)$ é a taxa de aprendizado, e $h_{bi}(t)$ é a função de vizinhança centrada no BMU b, que determina o quanto cada neurônio vizinho será influenciado pela entrada.

O processo de treinamento é iterativo e pode ser dividido em duas fases: a fase de ordenação, onde ocorre uma rápida convergência inicial com uma alta taxa de aprendizado e raio de vizinhança amplo, e a fase de ajuste fino, onde esses parâmetros são reduzidos para estabilizar o mapa. Ao final do treinamento, os neurônios do SOM passam a representar uma discretização do espaço de entrada, sendo que padrões similares de entrada ativam neurônios próximos no mapa.

O SOM é amplamente utilizado para tarefas como agrupamento, segmentação de dados e visualização de relações topológicas em dados complexos, com aplicações em áreas como bioinformática, reconhecimento de padrões, análise de imagens, economia e geopolítica. Uma de suas principais vantagens está na capacidade de interpretar graficamente grandes volumes de dados e identificar clusters de forma intuitiva, mesmo em situações onde métodos supervisionados não são aplicáveis.

Nas simulações realizadas foram utilizadas uma malha de neurônios em duas dimensões 10x10. A taxa de aprendizado era igual a 0,5 e função de vizinhança Gaussiana com variância igual a 1.

A figura 1 apresenta o mapa SOM produzido e a figura 2 a dispersão dos dados.

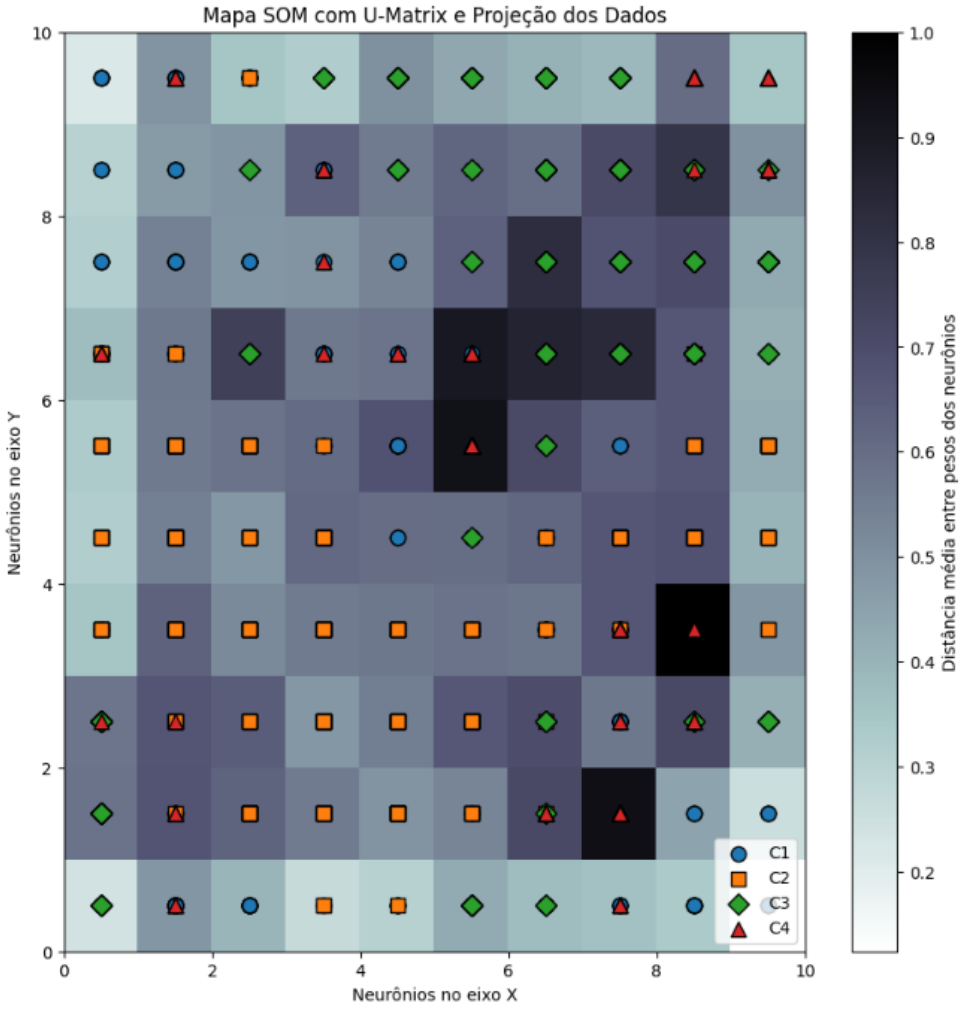


Figura 1 - Mapa SOM com U-Matrix e Projeção dos Dados.

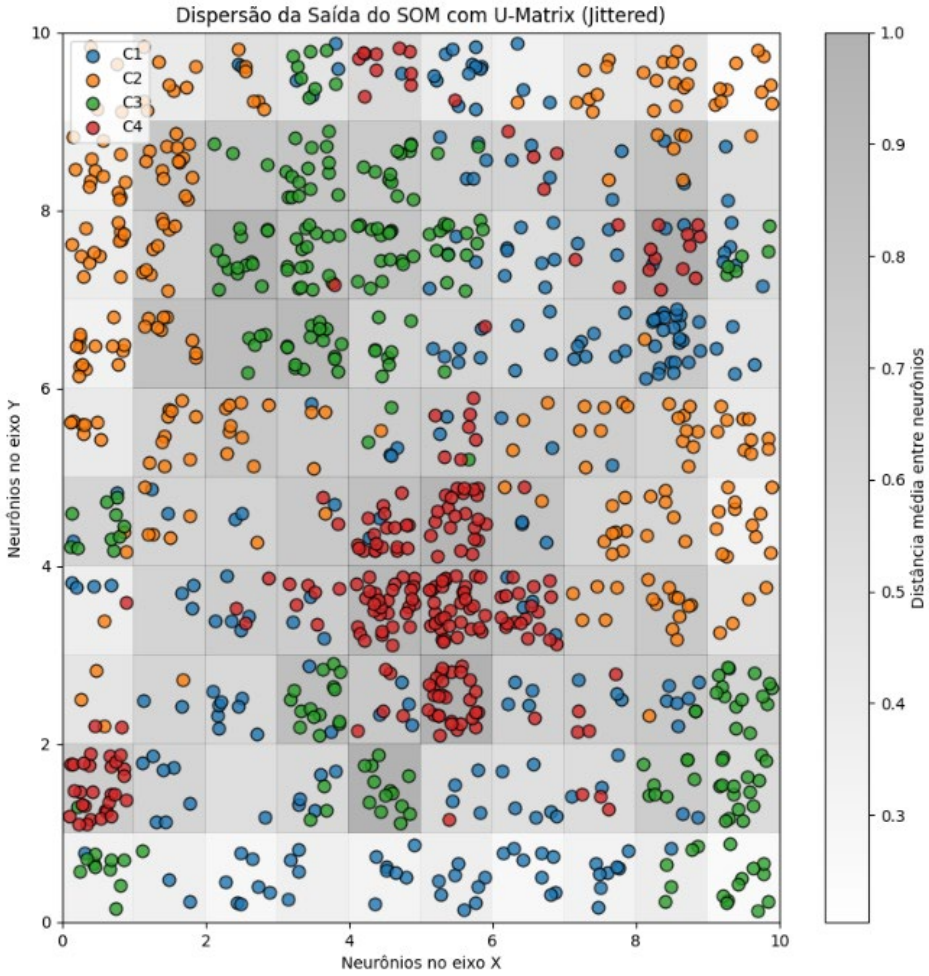


Figura 2 - Dispersão da Saída do SOM com a Matriz-U.

O mapa mostra quatro clusters principais (C1 a C4), representados por cores distintas (laranja, verde, azul, vermelho). A maioria dos grupos se distribui em regiões distintas da grade, indicando que os padrões de dados possuem diferenças significativas entre si.

O cluster vermelho (C4) ocupa uma região relativamente ampla, mas densa, indicando um grupo dominante ou com alta frequência nos dados. O cluster verde (C3) e o laranja (C2) estão mais dispersos e parecem ocupar regiões de transição, podendo refletir maior variabilidade interna. O cluster azul (C1) está mais presente na região inferior esquerda e partes do centro, mas em regiões separadas, o que pode indicar subgrupos internos ou ambiguidade de fronteira.

A ausência de grande sobreposição de pontos de cores diferentes dentro da mesma célula do mapa indica que o SOM conseguiu separar bem os padrões. O uso de *jittering* (leve ruído visual nas posições dos pontos) também ajudou a identificar sobreposições que poderiam passar despercebidas.

O código das simulações está disponível no seguinte link:

https://github.com/wilsonfmjr/PPGEEC2321---REDES-NEURAIS-E-DEEP-LEARNING/blob/main/LISTA_3/Questao_4.ipynb

Questão 5

Considere a tabela de índices de desenvolvimento de países (Fonte ONU- 2002, Livro – Análise de dados através de métodos de estatística multivariada – Sueli A. Mingoti) abaixo. Gere o mapa SOM e com isto identifique os clusters existentes, i.e., os países com características mais similares.

Países	Expectativa de Vida	Educação	PIB	Estabilidade Política
UK	0,88	0,99	0,91	1,1
Austrália	0,9	0,99	0,93	1,26
Canadá	0,9	0,98	0,94	1,24
EUA	0,87	0,98	0,97	1,18
Japão	0,93	0,93	0,93	1,2
França	0,89	0,97	0,92	1,04
Cingapura	0,88	0,87	0,91	1,41
Argentina	0,81	0,92	0,8	0,55
Uruguai	0,82	0,92	0,75	1,05
Cuba	0,85	0,9	0,64	0,07
Colômbia	0,77	0,85	0,69	-1,36
Brasil	0,71	0,73	0,72	0,67
Paraguai	0,75	0,83	0,63	-0,87
Egito	0,7	0,62	0,6	0,21
Nigéria	0,44	0,58	0,37	-1,36
Senegal	0,47	0,37	0,45	-0,68
Serra Leoa	0,23	0,33	0,27	-1,26
Angola	0,34	0,36	0,51	-1,98
Etiópia	0,31	0,35	0,32	-0,55
Moçambique	0,24	0,37	0,36	0,2
China	0,76	0,8	0,95	1,09

Resposta

Diferente da questão 3, onde o objetivo era redução de dimensionalidade de dados multidimensionais, nesta o objetivo é utilizar O *Self-Organizing Map* (SOM), agrupamento de dados multidimensionais. Diferente de algoritmos como k-means, o SOM não impõe uma estrutura rígida de partição, mas permite que os agrupamentos emerjam de forma autônoma e visualmente interpretável.

Nas simulações realizadas foram utilizadas uma malha de neurônios em duas dimensões 5x5. A taxa de aprendizado era igual a 0,05 e função de vizinhança Gaussiana com variância igual a 5,0.

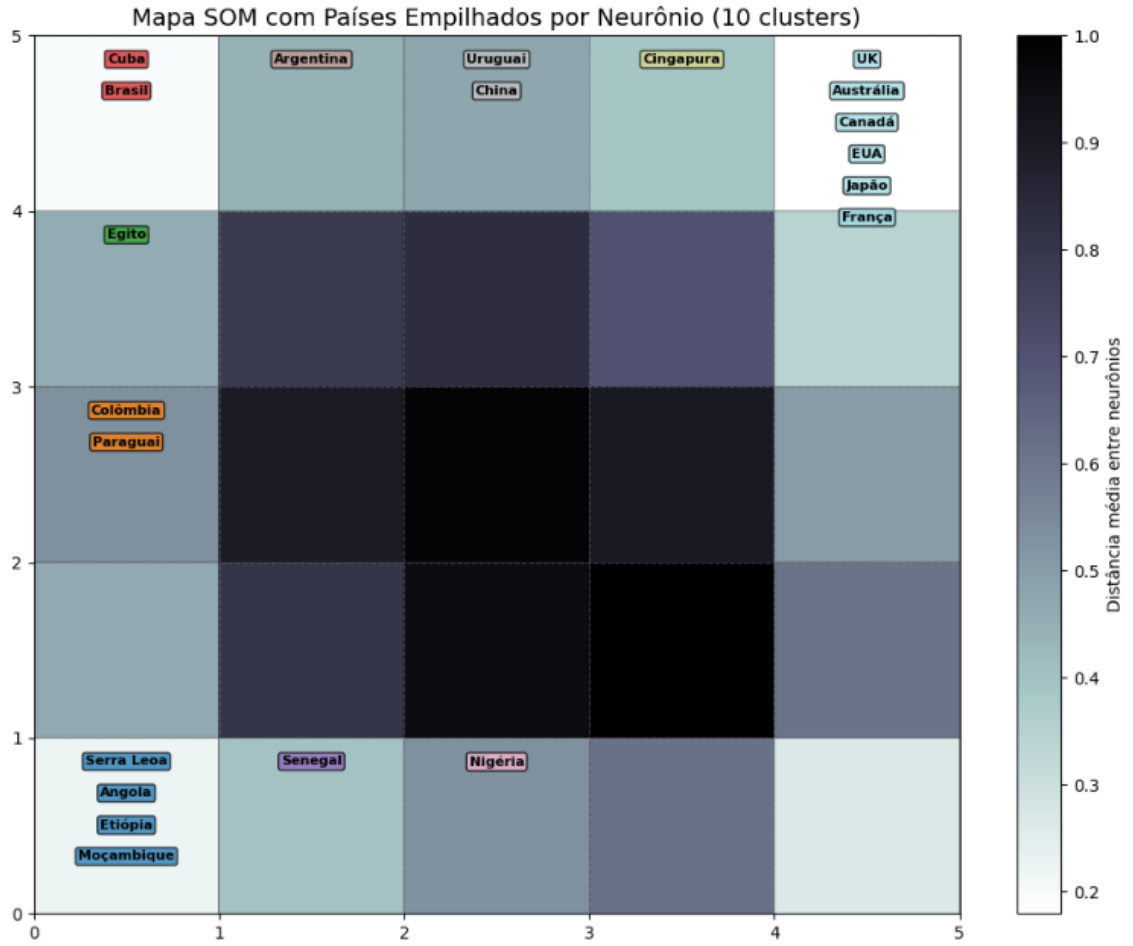


Figura 1 - Mapa SOM com Países Empilhados por Neurônio.

O SOM agrupou países com índices de desenvolvimento semelhantes no mesmo neurônio ou em neurônios vizinhos. Por exemplo, Cuba e Brasil estão no mesmo neurônio, sugerindo perfil de desenvolvimento comparável. Nigéria e Senegal estão em neurônios próximos, indicando similaridade em indicadores de desenvolvimento.

O neurônio com UK, Austrália, Canadá, EUA, Japão e França agrupa nações desenvolvidas, refletindo um perfil comum de altos índices de desenvolvimento.

Países da África Subsaariana (como Angola, Etiópia, Moçambique, Serra Leoa) foram agrupados em uma mesma região do mapa (inferior esquerda), reforçando a semelhança estrutural em seus dados.

Países sul-americanos como Colômbia, Paraguai, Argentina, Uruguai e Brasil estão distribuídos em neurônios próximos, mostrando padrões regionais de desenvolvimento intermediário.

Egito está isolado, o que pode indicar um perfil intermediário entre África e países do Oriente Médio ou emergentes.

O fundo em escala de cinza representa a U-Matrix, ou seja, a distância média entre os neurônios vizinhos:

O número de clusters (10) foi adequado para permitir agrupamentos coerentes sem fragmentação excessiva, permitindo uma separação clara entre países desenvolvidos, em desenvolvimento e menos desenvolvidos, conforme seus indicadores.

O código das simulações está disponível no seguinte link:

https://github.com/wilsonfmjr/PPGEEC2321---REDES-NEURAIS-E-DEEP-LEARNING/blob/main/LISTA_3/Questao_5.ipynb