UNIVERSIDADE DE SÃO PAULO
ESCOLA POLITÉCNICA

Sobre algoritmos de treinamento de redes neurais

Relatório Final – Iniciação Científica

Bolsa PIBIC-CNPq

Código do Projeto: 2024-3254

Bolsista: Pedro Henrique dos Santos Soares

Orientador: Magno Teófilo Madeira da Silva

Coorientador: Renato Candido

São Paulo
Setembro - 2025

Sumário

Resumo **3**

1 Introdução **4**
1.1 Objetivos 4
1.2 Cronograma de Atividades 4

2 Fundamentos Teóricos **5**
2.1 O Perceptron de Rosenblatt 5
2.2 A rede MLP 7
2.3 Treinamento da MLP 11
2.4 Teorema da Aproximação Universal 13

3 Rede Kolmogorov–Arnold **13**
3.1 Teorema da Representação de Kolmogorov–Arnold 13
3.2 *B-splines* 15
3.3 Construção Intuitiva da KAN 19
3.4 Propagação e Treinamento da KAN 23
3.5 Extração da Expressão Simbólica 25
3.6 Extensão de Nós em Redes KAN 25
3.7 Esparsificação e Poda em Redes KAN 26

4 Aplicação da KAN em Problemas de Classificação e Regressão **27**
4.1 Problema das Meias-Luas 27
4.2 Problemas de Regressão 30
4.3 Problema de Classificação Financeira 33
4.4 Classificação multi-classe com o MNIST 35

5 O algoritmo NoProp **38**
5.1 Processo estocástico de difusão / *Denoising* 39
5.2 Processo reverso de ruído / Variacional posterior 39
5.3 Função de perda 40
5.4 Treinamento e inferência 41
5.5 Arquitetura das camadas 41

6 Aplicação do NoProp em problemas de classificação **43**
6.1 NoProp aplicado ao problema das meias-Luas 43
6.2 NoProp aplicado ao Problema de Classificação Financeira 45
6.3 NoProp aplicado ao MNIST com Dimensões Reduzidas 48

7 Conclusões **49**

A Método dos Mínimos Quadrados Não Linear **50**

Referências **51**

Anexo 1 - Artigo aceito no SBrT 2025 **53**

Resumo

Este relatório final tem como propósito apresentar alternativas à Rede Neural Perceptron Multicamadas, focando principalmente na proposta recente das Redes de Kolmogorov-Arnold (KANs). Inicialmente, são apresentados os objetivos e o cronograma de atividades. Em seguida, são abordadas as partes principais que constituem uma Rede Neural Perceptron Multicamadas, como sua arquitetura, o conceito de função de ativação e os algoritmos utilizados para realizar o aprendizado. De maneira semelhante, é discutida a proposta das Redes de Kolmogorov-Arnold, apresentando suas peculiaridades, objetivos e algumas comparações com o Perceptron Multicamadas. Após explorar os conceitos introdutórios envolvidos, são resolvidos alguns problemas típicos de redes neurais para avaliar como as Redes KANs se comportam em diferentes cenários.

Cabe observar que boa parte deste relatório foi apresentada anteriormente no Relatório Parcial. O objetivo dessa repetição é alcançar um documento completo que possa vir a ser utilizado por futuros pesquisadores do grupo. Nos seis meses finais desse projeto, a KAN foi aplicada a dados reais (Subseções 4.3 e 4.4). Além disso, o algoritmo NoProp foi estudado, implementado e testado em problemas de classificação (Seções 5 e 6). Além de submeter um artigo ao SIICUSP, foi aceito um artigo na categoria de IC para apresentação no Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT) a ser realizado de 29 de setembro a 02 de outubro de 2025 em Natal, RN. Cópia do artigo aceito encontra-se anexada a este relatório.

1 Introdução

Nos últimos anos, a inteligência artificial vem ganhando protagonismo no desenvolvimento tecnológico, influenciando como hardwares e softwares são projetados atualmente e mudando radicalmente a interação humano-máquina [2,5,6]. Nessa guinada, o perceptron foi uma das primeiras propostas para a solução de problemas não lineares a ganhar notoriedade. Nos dias atuais, a rede perceptron multicamada (*Multilayer Perceptron* – MLP) [6] é um padrão nos métodos gerais de aprendizado de máquina. Com o amadurecimento dos modelos de redes neurais para resolver problemas cada vez mais complexos, percebeu-se que as MLPs, embora muito competentes, não permitem analisar como o problema é resolvido. Em outras palavras, os algoritmos conseguem de fato resolver o problema, mas a solução em si é praticamente uma caixa-preta. Nesse contexto, a proposta das redes de Kolmogorov-Arnold (*Kolmogorov-Arnold Network* – KAN) [11] surge como uma tentativa de entender melhor como as redes neurais conseguem resolver problemas.

1.1 Objetivos

Os objetivos principais estabelecidos neste projeto de pesquisa são:

1. Realizar um estudo sobre algoritmos de treinamento de redes neurais, em especial as Redes Kolmogorov-Arnold (KAN);
2. Construir modelos com KAN para verificar seu desempenho em problemas de regressão e classificação binária;
3. Estabelecer comparações entre as redes MLP e as KANs;
4. Verificar alternativas para além de MLP e KAN;
5. Redigir um relatório final que exponha a metodologia, as técnicas estudadas e os resultados das análises comparativas que possa ser útil a futuros pesquisadores do grupo.

1.2 Cronograma de Atividades

Neste projeto de IC, pretende-se seguir as seguintes etapas:

1. Realizar um estudo de redes neurais aplicadas à classificação. Inicialmente, pretende-se considerar um exemplo simples: o problema das meias-luas. Eventualmente, problemas mais complexos serão considerados. Neste item, pretende-se estudar as redes e os ajustes de seus hiperparâmetros;
2. Estudar a KAN proposta em [11];

3. Implementar a KAN e a MLP com o *backpropagation* convencional [6] ou otimizador Adam [5] e aplicar aos problemas de classificação do Item 1 ou outros mais complexos. A comparação deve levar em conta o desempenho em termos de métricas de classificação, como acurácia, por exemplo. Deve-se considerar também o custo computacional das soluções;

4. Buscar outras soluções propostas para melhorar o desempenho do *backpropagation*;

5. Redigir um relatório final que exponha a metodologia, as técnicas estudadas e os resultados das análises comparativas, que possa ser útil a futuros pesquisadores do grupo.

O cronograma de atividades está mostrado na Tabela 1.

Tabela 1: Cronograma de atividades.

Período	Atividades
Setembro de 2024 a Dezembro de 2024	Etapa 1
Novembro de 2024 a Fevereiro de 2025	Etapa 2
Março de 2025 a Maio de 2025	Etapa 3
Junho de 2025 a Agosto de 2025	Etapa 4
Agosto de 2025	Etapa 5

2 Fundamentos Teóricos

Esta seção apresenta a construção das redes neurais artificiais, abordando sua evolução desde os modelos iniciais até os avanços que permitiram seu uso eficiente em aprendizado profundo. Inicialmente, é introduzido o Perceptron de Rosenblatt, modelo pioneiro que estabeleceu os fundamentos do aprendizado supervisionado. Em seguida, discute-se a generalização desse conceito por meio da rede MLP e a necessidade de um algoritmo eficiente para seu treinamento, levando à introdução da retropropagação. Também é abordado o Teorema da Aproximação Universal, que formaliza a capacidade das redes neurais de representar funções arbitrárias. Por fim, é apresentado o otimizador Adam, amplamente utilizado para ajustar os parâmetros do modelo de forma eficiente.

2.1 O Perceptron de Rosenblatt

O perceptron de Rosenblatt, desenvolvido por Frank Rosenblatt em 1958 [14], é um dos primeiros modelos de neurônio e representa um marco significativo no desenvolvimento da inteligência artificial [6]. Este modelo é um algoritmo de aprendizado supervisionado voltado para a classificação binária, inspirado no funcionamento dos neurônios biológicos no

processamento de informações. O perceptron é capaz de aprender a classificar entradas em duas categorias distintas com base em exemplos previamente rotulados, demonstrando uma capacidade notável de aprendizado e generalização [14].

O perceptron é composto pelos seguintes elementos:

- Um vetor de entrada definido como $\mathbf{x} = [x_1\ x_2\ \cdots\ x_n]^T$, em que cada componente do vetor representa uma característica do problema;
- Um viés (*bias*), denotado por b;
- Um vetor de pesos $\boldsymbol{\omega} = [\omega_1\ \omega_2\ \cdots\ \omega_n]^T$, que multiplica as respectivas componentes da entrada;
- Uma função de ativação, que é aplicada à soma ponderada das entradas somada ao viés.

A função de ativação utilizada no modelo original é a função degrau (u), definida como

$$u(z) = \begin{cases} 1, & \text{se } z \geq 0 \\ 0, & \text{se } z < 0. \end{cases} \tag{1}$$

A Figura 1 apresenta uma ilustração da estrutura do perceptron de Rosenblatt.

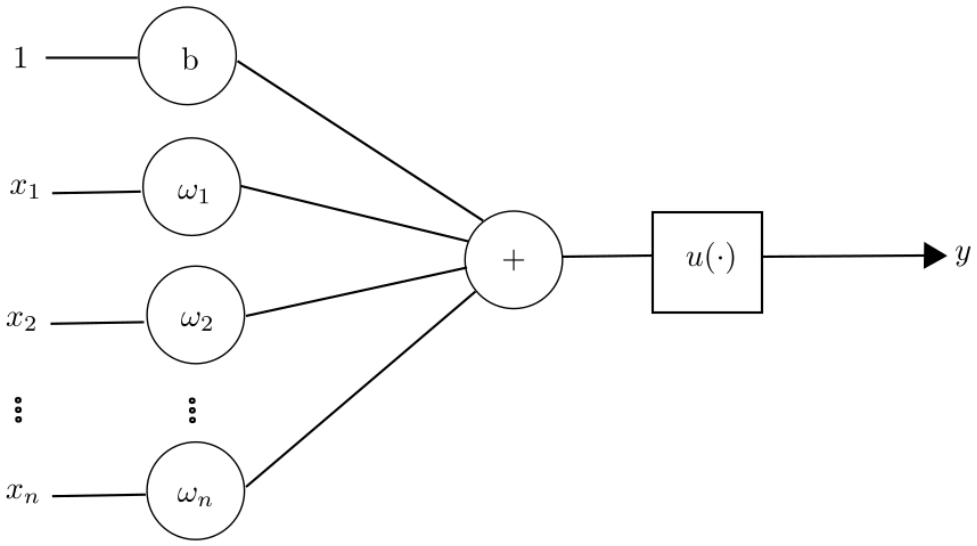

Figura 1: Estrutura do perceptron de Rosenblatt.

A saída do perceptron pode ser expressa matematicamente como

$$y = u(\mathbf{x}^T\boldsymbol{\omega} + b). \tag{2}$$

Apesar de sua simplicidade, o perceptron apresenta uma limitação fundamental: ele só consegue resolver problemas linearmente separáveis, ou seja, aqueles em que os exemplos das duas classes podem ser separados por uma linha (em duas dimensões), um plano (em três dimensões) ou um hiperplano em espaços de maior dimensão [6]. Um exemplo clássico de problema não linearmente separável é o problema da função XOR, que não pode ser resolvido por um perceptron simples [6]. Essa limitação levou ao desenvolvimento de arquiteturas mais complexas, como a organização do perceptron em várias camadas, dando origem à rede MLP.

A relevância do perceptron de Rosenblatt está no fato de que ele estabeleceu as bases para o aprendizado em redes neurais, introduzindo conceitos fundamentais como a atualização de pesos baseada no erro calculado a partir da comparação da saída da rede com um rótulo ou sinal desejado, um princípio que ainda é utilizado em modelos modernos de aprendizado profundo [2, 6].

2.2 A rede MLP

A rede MLP organiza os neurônios em múltiplas camadas. As camadas que aparecem entre a de entrada e a de saída são chamadas de camadas ocultas. Isso permite que a rede aprenda representações internas mais ricas e descubra padrões complexos nos dados [6].

A estrutura de uma MLP, ilustrada na Figura 2, consiste em três partes principais [6]:

- **Camada de entrada**: recebe os dados iniciais e os repassa para os neurônios da primeira camada oculta;
- **Camadas ocultas**: realizam transformações sucessivas nos dados, permitindo que a rede aprenda relações não triviais;
- **Camada de saída**: gera a resposta final da rede, que pode representar categorias (em um problema de classificação) ou valores contínuos (em uma tarefa de regressão).

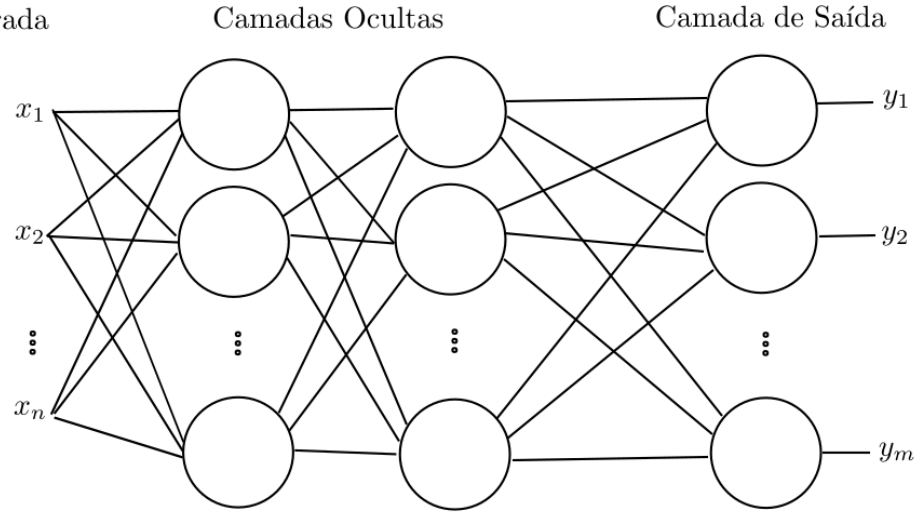

Figura 2: Estrutura da MLP.

Matematicamente, uma MLP com L camadas pode ser descrita como uma sequência de transformações aplicadas sobre a entrada $\mathbf{x}$. Define-se

$$\mathbf{a}^{(0)} = \mathbf{x},$$

em que $\mathbf{a}^{(0)}$ é simplesmente o vetor de entrada, e a saída final da rede é dada por

$$\mathbf{a}^{(L)} = \mathbf{y},$$

em que $\mathbf{y} = [y_1 \ y_2 \ \cdots \ y_m]^T$ representa a predição feita pelo modelo e m é o número de saídas da rede.

Cada camada l contém N_l neurônios e realiza duas operações principais:

1. **Transformação linear**: os valores da camada anterior $\mathbf{a}^{(l-1)}$ são combinados linearmente por meio de uma matriz de pesos $\mathbf{W}^{(l)}$ e um vetor de viés $\mathbf{b}^{(l)}$

$$\mathbf{u}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \tag{3}$$

   em que

   - $\mathbf{W}^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ é a **matriz de pesos**, cujos elementos $w_{ij}^{(l)}$ determinam a conexão entre o neurônio j da camada $l-1$ e o neurônio i da camada l;
   - $\mathbf{b}^{(l)} \in \mathbb{R}^{N_l}$ é o **vetor de vieses**;
   - $\mathbf{u}^{(l)} \in \mathbb{R}^{N_l}$ representa o vetor antes da função não linear.

2. **Aplicação da função de ativação**: o vetor $\mathbf{u}^{(l)}$ passa por uma função g, que introduz não linearidade ao modelo

$$\mathbf{a}^{(l)} = g(\mathbf{u}^{(l)}). \tag{4}$$

Este processo denominado propagação direta é repetido camada após camada até que a saída $\mathbf{a}^{(L)}$ seja obtida. A Figura 3 exemplifica como se dão as relações em uma propagação direta.

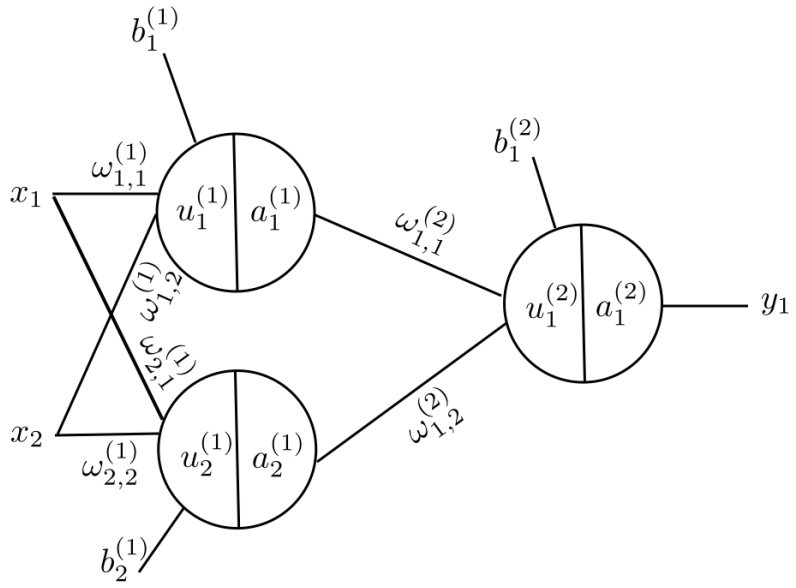

Figura 3: Exemplo de propagação direta para uma rede MLP com uma camada de entrada, uma camada oculta com dois neurônios e uma camada de saída com um neurônio.

A função de ativação g tem um papel fundamental no aprendizado do modelo, pois é responsável por introduzir não linearidade à rede neural [9]. No perceptron original, a ativação era baseada em uma função degrau, onde um neurônio era ativado apenas se a soma ponderada das entradas superasse um certo limiar. No entanto, a derivada da função degrau é igual a zero, exceto na origem, onde não é diferenciável. Para resolver esse problema, a MLP utiliza outras funções de ativação que são diferenciáveis e com derivada diferente de zero. As funções de ativação mais utilizadas são:

$$\textbf{Sigmoide:} \quad g(u) = \frac{1}{1+e^{-u}},$$

$$\textbf{Tangente Hiperbólica:} \quad g(u) = \tanh(u) = \frac{1-e^{-u}}{1+e^{-u}},$$

$$\textbf{ReLU:} \quad g(u) = \max(0, u).$$

A Figura 4 apresenta os respectivos gráficos e derivadas dessas funções. A sigmoide e a tangente hiperbólica são funções suaves e produzem saídas dentro de um intervalo limitado, o que pode ser útil para normalizar os valores ao longo da rede. Já a ReLU (*Rectified Linear Unit*) é amplamente utilizada devido à sua simplicidade computacional e à sua capacidade de manter a ativação de neurônios sem que os valores fiquem restritos a um intervalo fixo.

O fato da ReLu não ter derivada em $u = 0$ não causa problema, bastando considerar o valor da derivada nesse ponto igual a 0 ou 1 de forma arbitrária.

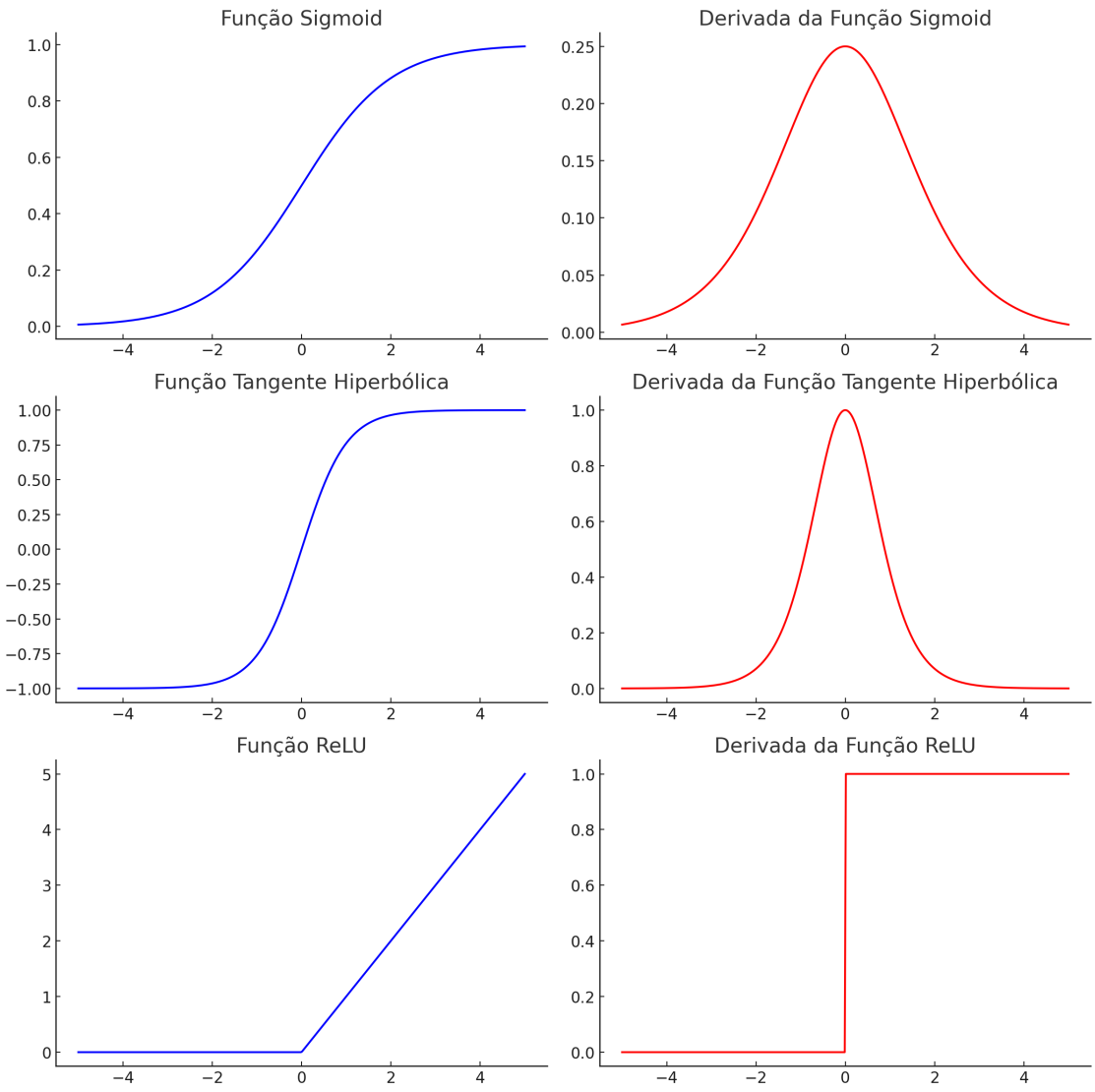

Figura 4: Gráficos das Principais Funções de Ativação.

A rede MLP é amplamente utilizada em aprendizado de máquina por sua capacidade de representar relações complexas entre os dados. Sua estrutura permite que seja aplicada a uma ampla variedade de problemas, desde reconhecimento de padrões até previsão de séries temporais. Entretanto, sua expressividade teórica depende de resultados fundamentais sobre redes neurais, que são abordados na próxima seção. Além disso, sua implementação prática exige um treinamento eficiente, o que será estudado em detalhes ao se discutir o algoritmo de *backpropagation* [15].

2.3 Treinamento da MLP

O treinamento da MLP consiste no ajuste dos pesos $\mathbf{W}^{(l)}$ e dos vieses $\mathbf{b}^{(l)}$ de modo que a rede consiga aprender a tarefa desejada. Esse ajuste é realizado minimizando uma função de custo por meio do algoritmo de retropropagação (*backpropagation*) [6, 15].

A função custo, ou função de perda, mede o erro da rede ao comparar suas previsões com os valores reais esperados [6]. Durante o treinamento, busca-se minimizar essa função para melhorar o desempenho do modelo. As funções custo mais comuns são:

- **Erro Quadrático Médio (MSE - *Mean Squared Error*)**, utilizado principalmente em problemas de regressão, é definido como

$$J = \frac{1}{2K} \sum_{i=1}^{K} \|\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\|^2. \tag{5}$$

  em que

  - K representa o número de amostras do conjunto de treinamento;
  - $\mathbf{y}^{(i)}$ é o vetor de saída esperado para a i-ésima amostra, ou seja, o valor real (desejado) associado à i-ésima entrada fornecida à rede.
  - $\hat{\mathbf{y}}^{(i)}$ é o vetor de saída previsto pela rede para a mesma amostra i, ou seja, o valor retornado pelo modelo após a propagação direta.

  Essa função mede a diferença entre as previsões do modelo e os valores reais, penalizando erros quadráticos. O fator $\frac{1}{2}$ é introduzido por conveniência matemática, facilitando a derivação no cálculo do gradiente.

- **Entropia Cruzada**, amplamente empregada em problemas de classificação, é definida como

$$J = -\frac{1}{K} \sum_{i=1}^{K} \sum_{j=1}^{c} y_j^{(i)} \log \hat{y}_j^{(i)}. \tag{6}$$

  onde:

  - c representa o número total de classes possíveis no problema de classificação;
  - $y_j^{(i)}$ é um valor binário que assume 1 se a i-ésima amostra pertence à classe j e 0 caso contrário (codificação *one-hot*);
  - $\hat{y}_j^{(i)}$ é a probabilidade prevista pelo modelo de que a amostra i pertença à classe j, obtida a partir da camada de saída da rede.

  Essa função mede a similaridade entre as distribuições de $\mathbf{y}^{(i)}$ e $\hat{\mathbf{y}}^{(i)}$, incentivando o modelo a atribuir probabilidades altas às classes corretas e penalizando previsões erradas.

Para minimizar J, aplica-se o gradiente descendente, um método iterativo que ajusta os pesos e vieses na direção oposta ao gradiente da função de custo [6]. A atualização ocorre da seguinte forma

$$W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \eta \frac{\partial J}{\partial W_{ij}^{(l)}}, \quad b_i^{(l)} \leftarrow b_i^{(l)} - \eta \frac{\partial J}{\partial b_i^{(l)}}, \tag{7}$$

em que η representa a taxa de aprendizado. O cálculo das derivadas é feito usando a regra da cadeia. Uma vez calculadas as derivadas, ocorre a propagação da camada de saída até a camada de entrada. Mais detalhes sobre o algoritmos podem ser encontrados em [2, 6].

No contexto do treinamento de redes neurais, define-se uma **época** como um ciclo completo no qual a rede processa todas as amostras do conjunto de dados uma vez. Durante uma época, cada exemplo do conjunto de treinamento é passado pela rede, os erros são calculados e os pesos são atualizados. O treinamento geralmente ocorre por múltiplas épocas até que a função custo seja minimizada ou um critério de parada seja atendido.

O algoritmo de treinamento pode ser descrito da seguinte maneira:

1. Inicializar os pesos $\mathbf{W}^{(l)}$ e vieses $\mathbf{b}^{(l)}$ com valores aleatórios pequenos.

2. Para cada **época**:

   Para cada amostra do conjunto de dados no modo de treinamento estocástico:

   i. Executar a propagação direta para calcular as ativações $\mathbf{a}^{(l)}$;

   ii. Utilizar a função de custo para medir o erro da rede;

   iii. Calcular os gradientes $\frac{\partial J}{\partial \mathbf{W}^{(l)}}$ e $\frac{\partial J}{\partial \mathbf{b}^{(l)}}$;

   iv. Atualizar os pesos e vieses usando a Equação (7).

O algoritmo de retropropagação tem a tendência de ficar parado em mínimos locais, o que dificultou o uso de redes profundas com muitas camadas ocultas até a década de 2010. Para contornar esse problema foram propstas modificações no algoritmo, levando ao otimizador Adam (*Adaptive Moment Estimation*) [5, 7].

O otimizador Adam é um dos algoritmos de otimização mais utilizados atualamente no treinamento de redes MLP. Em vez do gradiente convencional, ele utiliza uma média dos gradientes passados utilizando uma janela exponencial com fator de esquecimento β_1. Além disso, ele calcula uma média quadrática dos gradientes passados, utilizando outra janela exponencial com fator de esquecimento β_2. Essa média quadrática é utilizada para normalizar o gradiente médio. Dessa forma, o Adam ajusta automaticamente a taxa de aprendizado para cada parâmetro individualmente, permitindo atualizações mais eficientes e estáveis em redes neurais profundas [5, 7].

2.4 Teorema da Aproximação Universal

O Teorema da Aproximação Universal estabelece que uma rede neural com uma única camada oculta e um número suficiente de neurônios pode aproximar qualquer função contínua definida em um conjunto compacto.

Formalmente, temos [4, 6]:

Seja $\varphi(\cdot)$ uma função contínua, não constante, limitada e monotônica crescente. Denotamos por I_{N_0} o hipercubo unitário $[0,1]^{N_0}$ de dimensão N_0, e por $C(I_{N_0})$ o espaço das funções contínuas sobre esse domínio. Então, para qualquer $f \in C(I_{N_0})$ e $\varepsilon > 0$, existe um inteiro N_1 e coeficientes reais α_i, b_i e w_{ij}, com $i = 1, \ldots, N_1$ e $j = 1, \ldots, N_0$, tais que a função

$$F(x_1, \ldots, x_{N_0}) = \sum_{i=1}^{N_1} \alpha_i g \left(\sum_{j=1}^{N_0} w_{ij} x_j + b_i \right)$$

satisfaz a desigualdade

$$\sup_{x \in I_{N_0}} |F(x) - f(x)| < \varepsilon.$$

Isso significa que redes neurais com uma única camada oculta são aproximadores universais, podendo representar qualquer função contínua sobre um domínio compacto. No entanto, o teorema não especifica o número mínimo de neurônios necessário para uma aproximação eficiente, nem como determinar os pesos e bias ótimos. Na prática, redes mais profundas costumam ser preferidas, pois podem alcançar a mesma precisão com menos parâmetros e maior eficiência computacional.

3 Rede Kolmogorov–Arnold

Nesta seção, é apresentada a *Kolmogorov–Arnold Network* (KAN), uma abordagem baseada no teorema de Kolmogorov–Arnold, que garante a possibilidade de representar qualquer função contínua como uma composição de funções univariadas. Inicialmente, são expostos o teorema fundamental e sua relevância para a modelagem de funções, seguidos de uma introdução ao conceito de B-splines, que desempenham um papel essencial na construção da KAN ao permitir aproximações suaves e flexíveis de funções. Em seguida, é desenvolvida uma construção intuitiva da KAN, explicando sua estrutura e funcionamento de forma acessível, com destaque para as diferenças em relação às redes neurais tradicionais. Por fim, é apresentada a formalização matemática da KAN, com a definição das expressões analíticas envolvidas.

3.1 Teorema da Representação de Kolmogorov–Arnold

O Teorema da Representação de Kolmogorov–Arnold, estabelecido por Andrey Kolmogorov e posteriormente refinado por Vladimir Arnold, afirma que qualquer função multivariada

contínua pode ser expressa como uma composição finita de funções contínuas de uma única variável e a operação de adição [8, 11].

Em termos formais, se $f : [0,1]^n \to \mathbb{R}$ é uma função contínua definida em um hiperparalelogramo $[0,1]^n$, então existe um conjunto de funções contínuas univariadas $\{\phi_{q,p}\}$ e $\{\Phi_q\}$ tais que f pode ser representada como [8, 11]

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right). \tag{8}$$

Esse resultado é notável porque demonstra que qualquer função multivariada pode ser reduzida a combinações de funções de uma única variável, o que tem implicações profundas em diversas áreas, destacando-se o aprendizado de máquina. Uma motivação central para o teorema é a redução da complexidade no cálculo de funções multivariadas. Em muitas aplicações, trabalhar diretamente com $f(x_1, x_2, \ldots, x_n)$ pode ser computacionalmente custoso, enquanto sua representação em termos de funções de variável única permite que se usem algoritmos mais eficientes.

Para ilustrar as vantagens dessa representação, consideremos a seguinte função multivariada $f(x_1, x_2) = 2x_1x_2$. Sem a representação de Kolmogorov–Arnold, seria necessário calcular explicitamente a interação entre x_1 e x_2. No entanto, aplicando a decomposição do teorema, podemos reescrevê-la como

$$\begin{aligned} f(x_1, x_2) &= 2x_1x_2 = (x_1 + x_2)^2 - x_1^2 - x_2^2 \\ &= \Phi_0(\phi_{0,1}(x_1) + \phi_{0,2}(x_2)) + \Phi_1(\phi_{1,1}(x_1)) + \Phi_2(\phi_{2,2}(x_2)), \end{aligned} \tag{9}$$

em que definimos

$$\begin{aligned} &\phi_{0,1}(x) = x, && \phi_{0,2}(x) = x, && \phi_{1,1}(x) = x, \\ &\phi_{1,2}(x) = 0, && \phi_{2,1}(x) = 0, && \phi_{2,2}(x) = x, \\ &\Phi_0(x) = x^2, && \Phi_1(x) = -x^2, && \Phi_2(x) = -x^2. \end{aligned}$$

Essa representação mostra que a interação entre as variáveis x_1 e x_2 foi convertida em uma soma seguida de uma transformação univariada. A Figura 5 ilustra um diagrama de blocos levando-se em conta a representação de (9). É possível observar a capacidade dessa representação já que ela permite estabelecer uma estrutura simples de unidades funcionais. Essa característica tem um ótimo potencial para o uso em redes neurais [11].

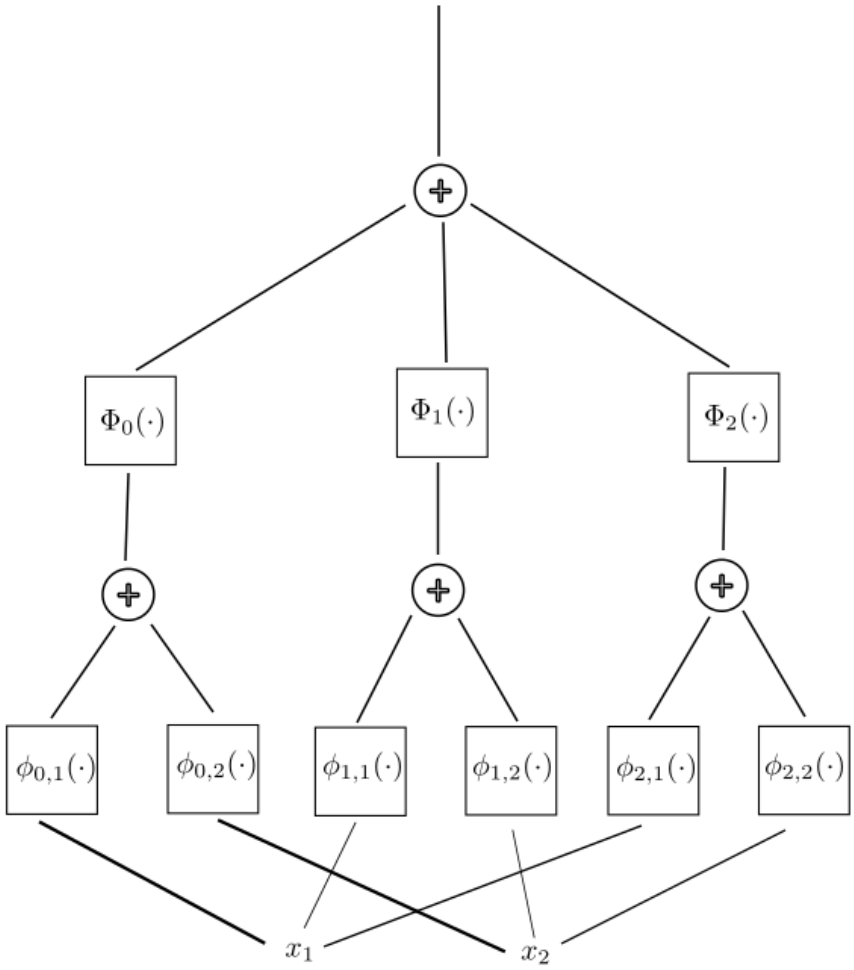

Figura 5: Ilustração esquemática de uma função $f(x_1,x_2) = 2x_1x_2$ representada pelo teorema de Kolmogorov–Arnold.

No entanto, embora o teorema garanta que existam funções univariadas que permitem a representação, ele não dá indícios de como obtê-las e também não fornece o número de funções necessárias. Além disso, mesmo que tais funções existam, não há garantia nenhuma de que elas tenham uma expressão analítica. Portanto, sua viabilidade em aplicações depende de métodos adicionais para contornar essas limitações [11].

3.2 *B-splines*

B-splines são funções de base polinomiais definidas em intervalos específicos, denominados intervalos de nó e são usadas para formar curvas contínuas de maneira suave. Elas são construídas a partir de polinômios de baixo grau e, por meio de combinações lineares, podem representar curvas complexas [16].

Uma curva *B-splines* de grau k é gerada pela combinação linear das funções de base $B_{i,k}$, ou seja,

$$C(x) = \sum_{i=0}^{n} c_i \cdot B_{i,k}(x), \tag{10}$$

em que c_i são coeficientes constantes, também chamados pontos de controle. As funções de base $B_{i,k}(x)$ são definidas com a fórmula de Cox-deBoor, dada por [11, 16]

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - t_i} \cdot B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} \cdot B_{i+1,k-1}(x), \tag{11}$$

em que t_i é o i-ésimo nó (*knot*).

Devido ao caráter recursivo da Equação (11) em que a função base $B_{i,k}(x)$ de grau k utiliza a função base $B_{i,k-1}(x)$ de grau $k-1$, é necessário definir a função base de grau 0, ou seja,

$$B_{i,0} = \begin{cases} 1, & \text{se } t_i \leq x \leq t_{i+1} \\ 0, & \text{caso contrário.} \end{cases} \tag{12}$$

Para fins de ilustração, considere a função $f(x) = \sin(x)$ no intervalo $[0, 2\pi]$. Ela pode ser representada por uma curva *B-spline* com grau $k = 3$ e 5 nós. A representação do domínio $[0, 2\pi]$ pode ser feita definindo os nós como pontos uniformemente espaçados nesse intervalo.

Uma vez que os nós são selecionados, as funções base são então definidas por (11). Nesse exemplo, as funções resultantes são apresentadas na Figura 6. Para simplificar a notação, como k está definido, utiliza-se B_i para se referir à função base $B_{i,k}$. É importante destacar que a forma das curvas independe do intervalo escolhido, isto é, para um grau $k = 3$, quaisquer cinco pontos uniformemente espaçados, tomados como nós, resultarão em funções com estas mesmas geometrias.

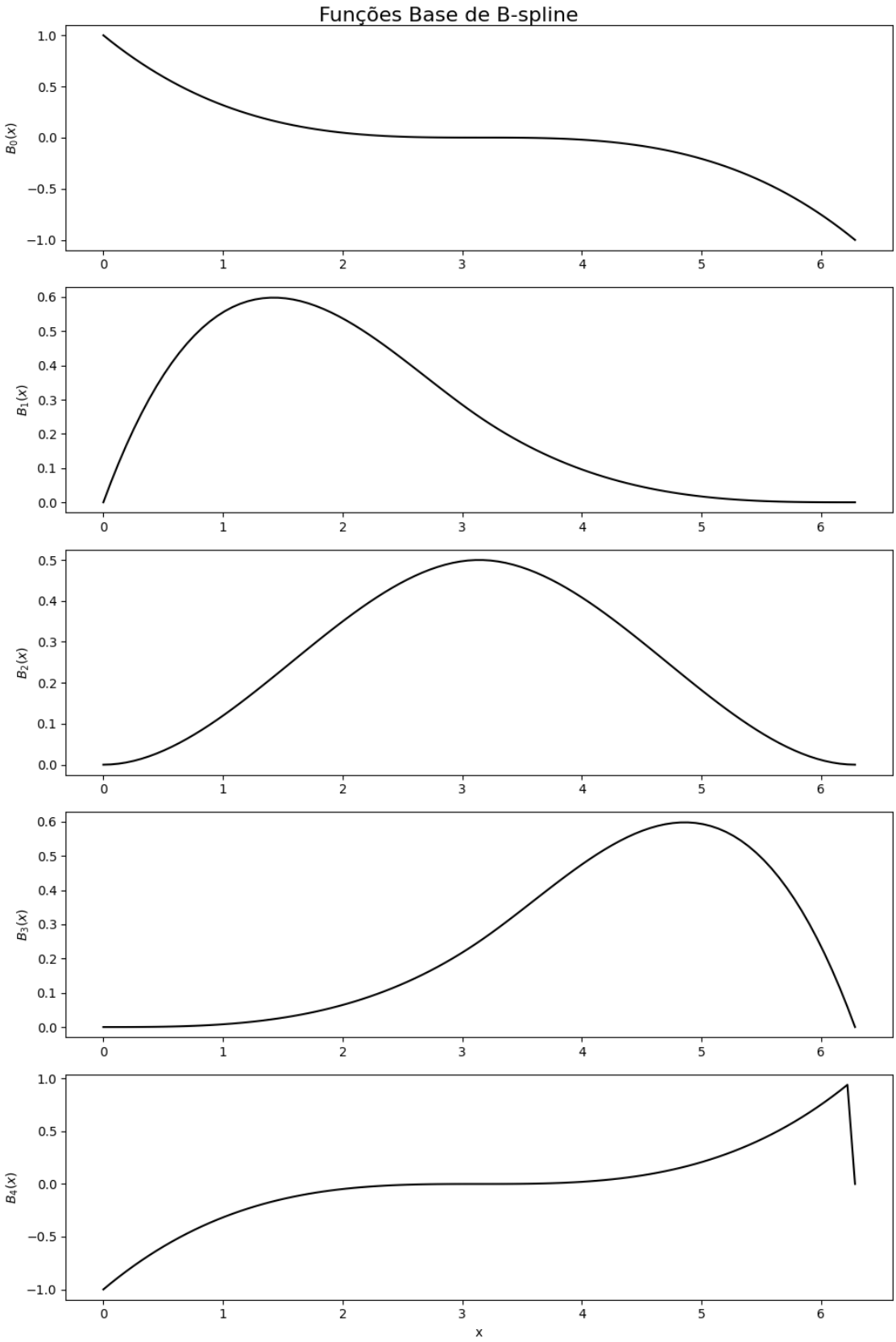

Figura 6: Funções Base de B-spline.

Uma vez determinadas as funções base, deve-se encontrar os coeficientes para que a *B-spline* aproxime a função desejada conforme a Equação (10), ou seja,

$$c_0 \cdot B_0(x) + c_1 \cdot B_1(x) + c_2 \cdot B_2(x) + c_3 \cdot B_3(x) + c_4 \cdot B_4(x) \approx \sin(x). \tag{13}$$

Utilizando o método dos mínimos quadrados, chega-se aos coeficientes

$$c_0 = 0, \;\; c_1 = 1{,}778, \;\; c_2 = 0, \;\; c_3 = -1{,}778 \quad \text{e} \quad c_4 = 0.$$

Na Figura 7, são mostradas as curvas das funções base e da aproximação (13). Na Figura 8 é mostrada a curva aproximada pela *B-spline* e a curva da função original. Verifica-se que a aproximação da *B-spline* coincide com os valores da função $\sin(x)$ nos nós e aproxima os valores do $\sin(x)$ em outros pontos do intervalo.

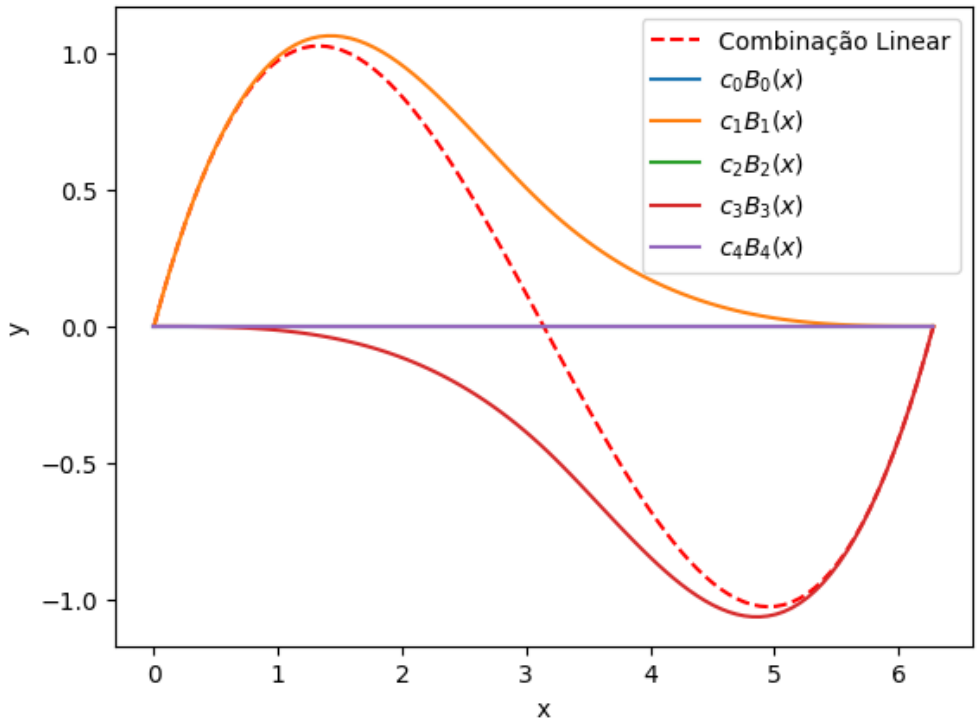

Figura 7: Resultado da Combinação Linear das Funções Base de Spline.

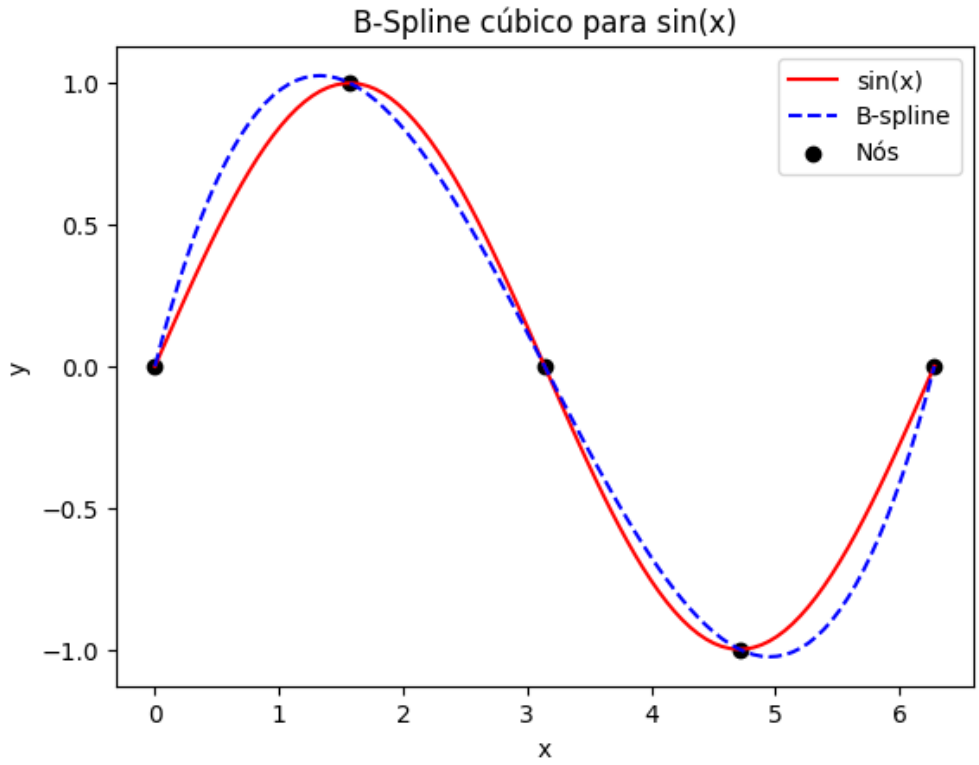

Figura 8: Comparação entre a Curva B-Spline resultante e a senoide original.

Essa capacidade de aproximação por meio de uma combinação linear de funções base univariadas torna as funções *B-spline* excelentes candidatas no Teorema da Representação de Kolmogorov–Arnold para aproximar funções.

3.3 Construção Intuitiva da KAN

A KAN foi proposta como uma alternativa ao perceptron multicamada, utilizando como inspiração o teorema da Representação de Kolmogorov–Arnold em vez do teorema da aproximação universal [11].

Para entender a KAN, imagine que a função $f(x_1, x_2)$ possa ser decomposta pelo Teorema da Representação de Kolmogorov–Arnold em funções univariadas da seguinte forma

$$f(x_1, x_2) = \Phi_0(\phi_{0,1}(x_1) + \phi_{0,2}(x_2)). \tag{14}$$

Repare que o teorema não fornece ferramentas para encontrar as funções $\Phi_0, \phi_{0,1}$ e $\phi_{0,2}$, sendo então o objetivo da KAN descobrir como determiná-las.

A Figura 9 ilustra a representação gráfica de um modelo de KAN para a Equação (14). A soma de sinais é definida como neurônio e as funções $\Phi_0, \phi_{0,1}$ e $\phi_{0,2}$ são todas designadas como funções de ativação.

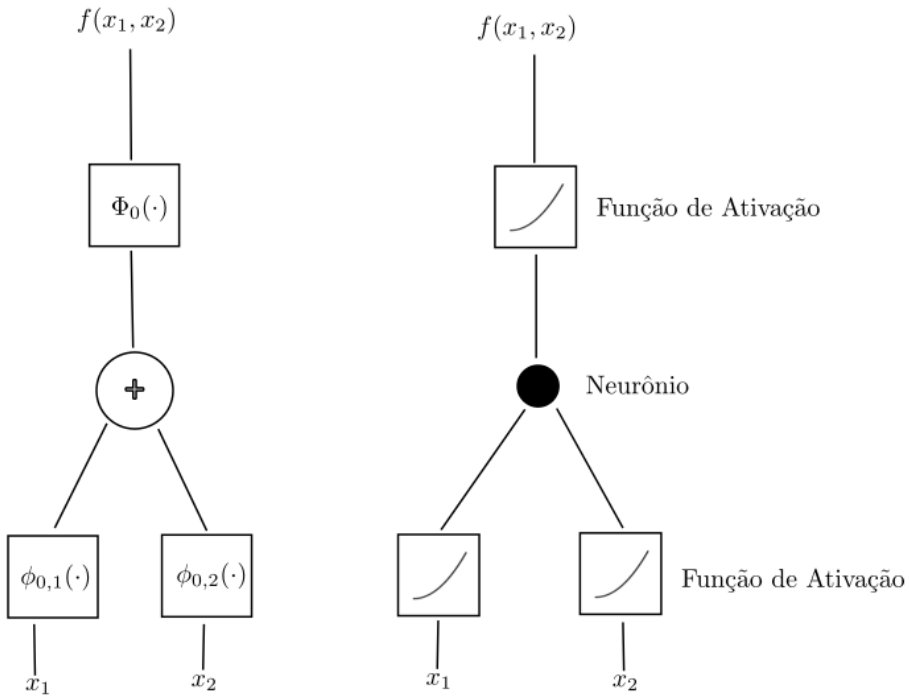

Figura 9: Representação de uma KAN.

Para lidar com o problema da representação das funções, a KAN propõe que cada função de ativação seja aproximada por um *B-spline*. Portanto, cada função de ativação vai ser representada por um conjunto de coeficientes. É importante garantir que as funções base sejam iguais para todas as funções de ativação. Para isso, é então necessário definir o número de nós e o grau das funções base como hiperparâmetros [11]. Para exemplificar, a Figura 10 ilustra como cada função é composta pela combinação linear das funções base fixas.

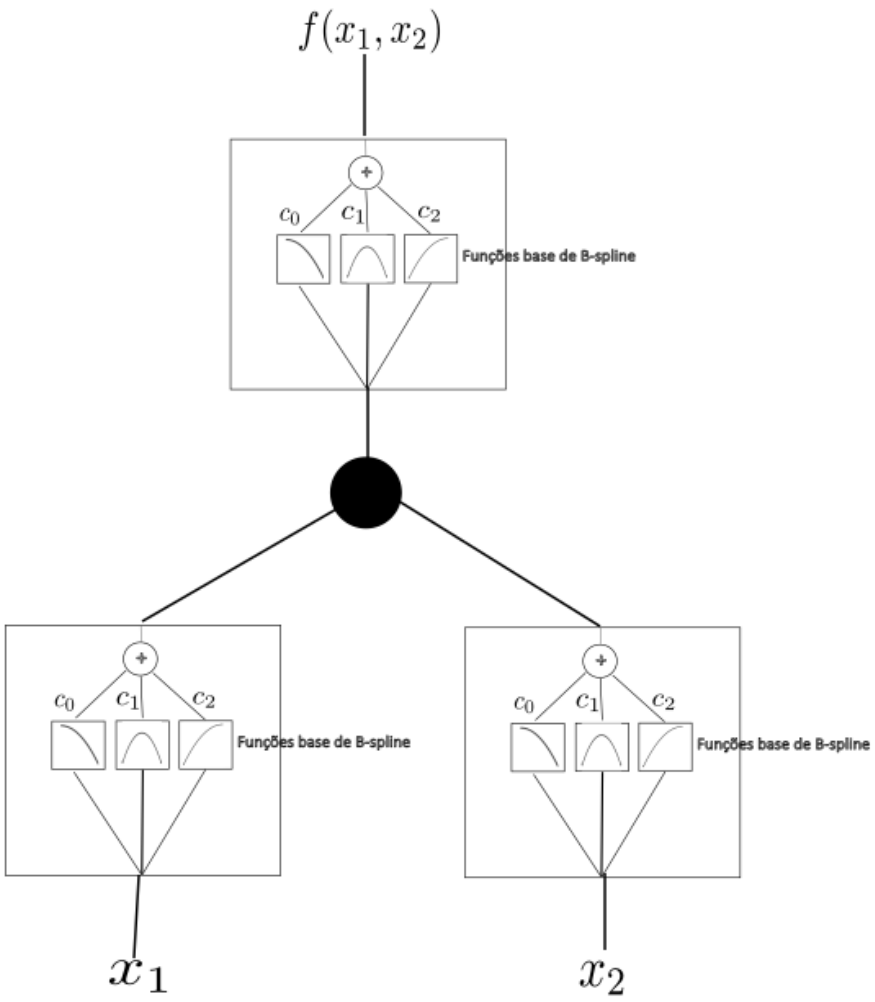

Figura 10: Estrutura das Funções de Ativação.

A utilização de *B-splines* para aproximar funções de ativação transforma o problema em uma questão de determinação dos coeficientes que compõem a combinação linear das funções base. Para estimar esses coeficientes, empregam-se algoritmos de treinamento similares aos utilizados em redes MLP.

Seguindo o mesmo raciocínio apresentado até aqui e ampliando a abordagem com algumas generalizações, é possível obter expressões mais complexas que representam redes profundas, conforme exemplificado na Figura 11.

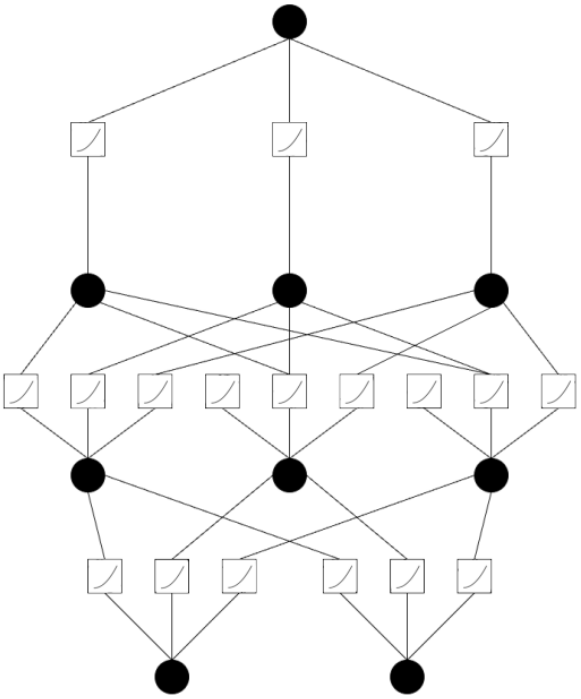

Figura 11: Exemplo de Rede KAN profunda.

A KAN se assemelha à MLP, pois ambas possuem uma estrutura de camadas densamente conectadas, onde todos os neurônios de uma camada estão ligados aos neurônios das camadas subsequentes. A principal diferença está na forma como a ativação de cada neurônio é calculada: na MLP, a entrada é ponderada por pesos ajustáveis e a saída passa por uma função de ativação fixa, enquanto na KAN, cada componente da entrada é processado por uma função de ativação ajustável baseada em *B-splines*, e a saída é obtida por uma soma simples dessas transformações, conforme ilustrado na Figura 12.

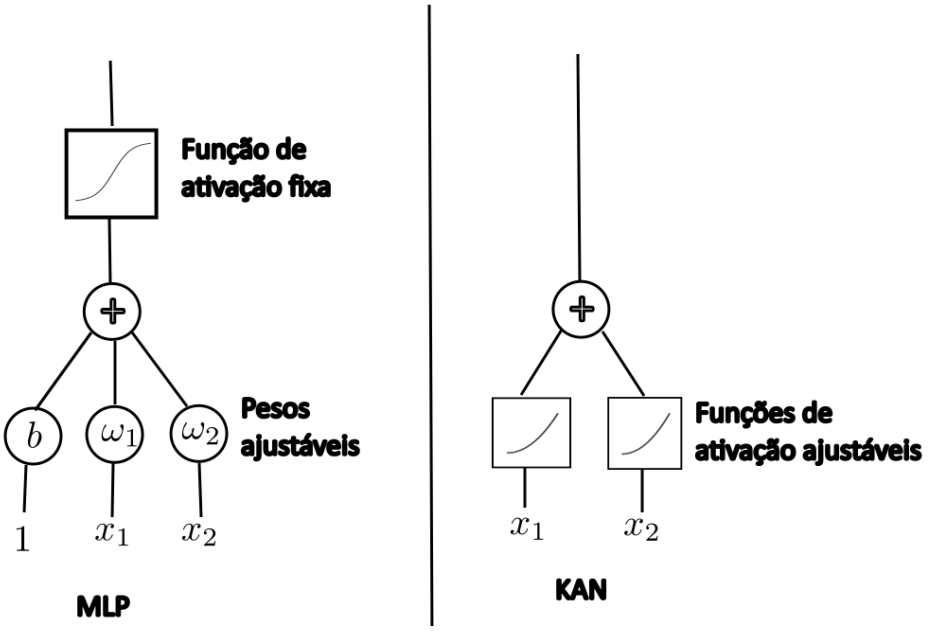

Figura 12: Comparação entre as ativações de uma MLP e de uma KAN.

3.4 Propagação e Treinamento da KAN

O processo de treinamento da Kolmogorov-Arnold Network (KAN) baseia-se na propagação direta da entrada até a saída e na retropropagação do erro para ajuste dos parâmetros. Os procedimentos são equivalentes aos vistos para uma MLP tradicional, mas adaptado para fazer o ajuste dos coeficientes de B-spline.

A propagação direta na KAN ocorre conforme os seguintes passos:

1. A rede recebe como entrada um vetor $\boldsymbol{x} = [x_1 \;\; x_2 \;\; \ldots \;\; x_n]^T$;

2. Antes de calcular as ativações da camada seguinte, os domínios das *B-splines* associadas às funções de ativação devem ser ajustados. Isso é feito modificando o primeiro ou o último nó da *B-spline*, caso a entrada esteja fora do intervalo original, garantindo que a forma da curva seja preservada;

3. As ativações dos neurônios das camadas ocultas e da camada de saída são computadas por

$$x_{\ell+1,j} = \sum_{i=1}^{n_\ell} \phi_{\ell,i,j}(x_{\ell,i}), \tag{15}$$

em que $x_{\ell+1,j}$ representa a ativação do j-ésimo neurônio da camada $\ell+1$, $x_{\ell,i}$ é a saída do i-ésimo neurônio da camada ℓ, e $\phi_{\ell,i,j}$ é a função de ativação conectando o neurônio i da camada ℓ ao neurônio j da camada $\ell+1$. Para a camada de entrada, temos que $x_{0,i} = x_i$;

4. As funções de ativação da KAN são compostas por uma combinação linear entre a função $b(x)$ e uma *B-spline*, ou seja,

$$\phi_{\ell,i,j}(x) = \omega_{\ell,i,j} \cdot \left[b(x) + \text{spline}_{\ell,i,j}(x)\right], \tag{16}$$

em que usualmente $b(x)$ é a função SiLU (*sigmoid linear unit*) definida como

$$b(x) = \frac{x}{1 + e^{-x}}, \tag{17}$$

e $\text{spline}_{\ell,i,j}(x)$ é uma *B-spline* de ordem $k = 3$. É importante notar que para $k = 1$, as funções base da *B-spline* são compostas por segmentos de retas, o que resulta em pontos não diferenciáveis. Portanto, para utilizar o algoritmo de *backpropagation* é necessário que $k > 1$.

Embora, na construção teórica da KAN, as funções de ativação sejam representadas exclusivamente por *B-splines*, na implementação prática, há termos adicionais, como evidenciado em (16). Essa modificação se faz necessária pois, na prática, o treinamento é mais eficiente quando os coeficientes das *B-splines* são inicializados segundo uma distribuição de média nula. No entanto, essa escolha de inicialização faz com que, nas primeiras iterações do algoritmo de treinamento, as *B-splines* resultem em valores nulos independentemente da entrada.

Para mitigar esse efeito e garantir que as ativações não sejam completamente nulas no início do treinamento, adicionam-se termos auxiliares, como o peso ω e a função $b(x)$, permitindo uma evolução mais estável do aprendizado.

O treinamento da KAN envolve a minimização de uma função de custo J utilizando o algoritmo de retropropagação do erro. O procedimento segue os seguintes passos:

1. Para cada amostra do conjunto de treinamento, a saída é calculada por meio da propagação direta;

2. O erro é calculado na camada de saída e propagado para as camadas anteriores por meio do cálculo dos gradientes;

3. Os pesos e coeficientes das *B-splines* são atualizados utilizando o gradiente descendente, ou seja,

$$\omega_{\ell,i,j} \leftarrow \omega_{\ell,i,j} - \eta \cdot \frac{\partial J}{\partial \omega_{\ell,i,j}}, \tag{18}$$

$$c^{(k)}_{\ell,i,j} \leftarrow c^{(k)}_{\ell,i,j} - \eta \cdot \frac{\partial J}{\partial c^{(k)}_{\ell,i,j}}, \tag{19}$$

em que $c^{(k)}_{\ell,i,j}$ representa o k-ésimo coeficiente da *B-spline* associado a $\phi_{\ell,i,j}$. O fator η é a taxa de aprendizado do otimizador.

O processo de treinamento iterativo permite ajustar os parâmetros da rede para minimizar a função de custo, garantindo que a KAN aprenda a mapear corretamente os padrões dos dados de treinamento.

3.5 Extração da Expressão Simbólica

Após o treinamento da KAN, com as funções de ativação ajustadas para resolver o problema desejado, torna-se relevante extrair uma expressão simbólica que represente adequadamente a função aprendida. Para isso, pode-se empregar o método dos mínimos quadrados não linear, apresentado no Apêndice A. Assim, as funções de ativação podem ser descritas por meio de funções do tipo

$$\phi_{\ell,i,j}(x) \approx g_{\ell,i,j}(x) = a + b \cdot f(c \cdot x + d), \tag{20}$$

em que a, b, c e d são parâmetros a serem ajustados pelo método dos mínimos quadrados e f é uma função conhecida, como seno (sin), cosseno (cos), tangente hiperbólica (tanh), logaritmo (log), polinômios, entre outras. Na KAN, pode-se tanto definir à priori a função f quanto utilizar diferentes funções e selecioná-la baseado em uma função custo.

Por fim, a saída da rede é expressa em termos da composição das funções g, permitindo a tradução da solução encontrada pela KAN em uma expressão simbólica interpretável.

3.6 Extensão de Nós em Redes KAN

Uma das principais vantagens das redes KAN é a possibilidade de ajustar a resolução das funções de ativação. Como essas funções são modeladas por splines, é possível torná-las mais precisas aumentando o número de nós utilizados. Esse processo é chamado de *extensão de nós*.

Em termos simples, a ideia é começar com poucas subdivisões nos domínios das funções — ou seja, com um número pequeno de nós — e, conforme o treinamento avança, adicionar mais nós para capturar melhor os detalhes da função que está sendo aprendida. A cada extensão, a spline se torna mais flexível, permitindo que a rede represente funções mais complexas com maior fidelidade.

A principal vantagem dessa abordagem é que não é necessário reiniciar o treinamento com uma rede maior. Em vez disso, utiliza-se o que já foi aprendido com poucos nós e expande-se a capacidade expressiva das funções de ativação. Os novos coeficientes da spline refinada podem ser inicializados para se aproximarem da função anterior, normalmente por meio de um ajuste de mínimos quadrados.

Esse processo pode ser repetido ao longo do treinamento, permitindo que a rede evolua gradualmente de uma forma simples e eficiente para uma representação mais precisa. Dessa forma, a extensão de nós oferece um caminho prático para melhorar a acurácia da rede sem comprometer sua estabilidade ou interpretabilidade.

3.7 Esparsificação e Poda em Redes KAN

Para tornar as redes KAN mais interpretáveis e eficientes, são introduzidas técnicas de simplificação baseadas em *esparsificação* e *poda*. Diferente das MLPs, que calculam uma combinação linear, somam um *bias* e utilizam uma função de ativação em cada neurônio, as KANs substituem esse cálculo por funções de ativação univariadas parametrizadas por splines. Isso exige uma adaptação das técnicas tradicionais de regularização.

A esparsificação visa induzir esparsidade nas conexões da rede, favorecendo representações mais simples. Define-se a norma ℓ_1 de uma função de ativação $\phi_{\ell,i,j}$ como a média da magnitude das ativações sobre um conjunto de entrada $x_1, x_2, \cdots, x_{N_p}$, ou seja

$$|\phi_{\ell,i,j}|_1 \equiv \frac{1}{N_p} \sum_{s=1}^{N_p} |\phi_{\ell,i,j}(x_s)|. \tag{21}$$

Para uma camada ℓ, representada como Φ_ℓ, com n_{in} entradas e n_{out} saídas, a norma ℓ_1 total é dada pela soma das normas das funções de ativação dessa camada, ou seja

$$|\Phi_\ell|_1 \equiv \sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} |\phi_{\ell,i,j}|_1. \tag{22}$$

Além disso, introduz-se uma regularização por entropia que penaliza distribuições uniformes entre as ativações, incentivando que apenas algumas conexões sejam relevantes, definida como

$$S(\Phi_\ell) \equiv -\sum_{i=1}^{n_{\text{in}}} \sum_{j=1}^{n_{\text{out}}} \frac{|\phi_{\ell,i,j}|_1}{|\Phi_\ell|_1} \log\left(\frac{|\phi_{\ell,i,j}|_1}{|\Phi_\ell|_1}\right). \tag{23}$$

A função de custo total utilizada no treinamento da rede é composta pela soma do custo preditivo com os termos de regularização por esparsidade e entropia

$$J_{\text{total}} = J_{\text{pred}} + \lambda \left(\mu_1 \sum_{\ell=0}^{L-1} |\Phi_\ell|_1 + \mu_2 \sum_{\ell=0}^{L-1} S(\Phi_\ell) \right), \tag{24}$$

em que J_{pred} representa o custo preditivo (por exemplo, calculado com o MSE ou entropia cruzada), λ controla a intensidade da regularização, e os pesos μ_1 e μ_2 ajustam a importância relativa dos termos de esparsificação e entropia, respectivamente. Em [11], considerou-se $\mu_1 = \mu_2 = 1$, já que esses pesos parecem não influenciar no desempenho. No entanto, o valor do parâmetro λ é essencial para controlar o efeito da regularização. Se λ for escolhido com um valor muito pequeno, a regularização não tem efeito. Se for escolhido com um valor muito grande, a saída da rede não converge adequadamente. Em geral, considera-se λ no intervalo $[0{,}1, 0{,}9]$.

Após o treinamento com esparsificação, é possível realizar a poda de neurônios considerados irrelevantes. Para isso, definem-se pontuações de entrada e de saída para cada neurônio

i na camada ℓ, respectivamente, como

$$I_{\ell,i} = \max_k |\phi_{\ell-1,k,i}|_1 \quad \text{e} \quad O_{\ell,i} = \max_k |\phi_{\ell+1,i,k}|_1. \tag{25}$$

Um neurônio é mantido na rede somente se ambas as pontuações superarem um limiar θ, geralmente definido como 10^{-2}. Caso contrário, o neurônio é removido, resultando em uma arquitetura mais compacta e com uma interpretabilidade mais simples.

Essas técnicas permitem a descoberta automática de arquiteturas KAN otimizadas para o problema em questão, oferecendo maior clareza e controle sobre o comportamento do modelo.

4 Aplicação da KAN em Problemas de Classificação e Regressão

Nesta seção, a KAN é avaliada em diferentes contextos, sendo seu desempenho comparado ao de uma MLP. Inicialmente, considera-se o problema clássico de classificação binária das meias-luas [6], a fim de verificar se a rede é capaz não apenas de classificar corretamente os dados, mas também de deduzir uma expressão analítica equivalente à sua estrutura interna.

Em seguida, são conduzidos experimentos em problemas de regressão simples, com o objetivo de analisar a relação entre as expressões resultantes da KAN e as funções reais que elas procuram aproximar, investigando potenciais vantagens quanto à interpretabilidade e à fidelidade da representação.

Por fim, são explorados dois problemas de aplicação prática: um de classificação binária no setor financeiro e outro de classificação multiclasse, utilizando a base de dados MNIST.

4.1 Problema das Meias-Luas

Um problema clássico abordado em aprendizado de máquina é o *problema de classificação das meias-luas* [6]. Esse problema consiste em um conjunto de dados, gerados a partir da seleção de alguns parâmetros geométricos que definem a forma das meias-luas, como ilustrado na Figura 13. O objetivo é gerar uma curva de separação no plano xy, de modo que cada metade da lua seja definida como pertencente a uma determinada região.

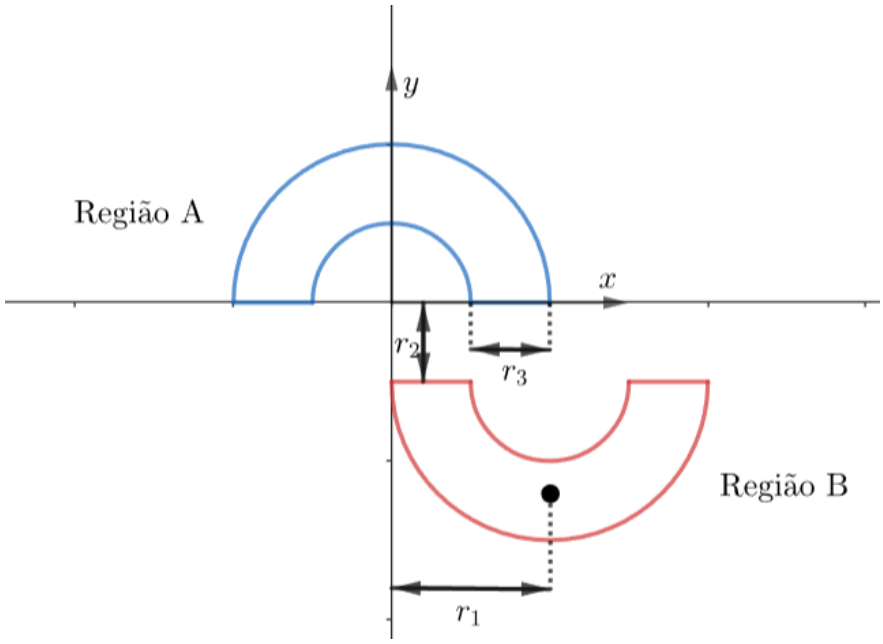

Figura 13: Apresentação do problema das meias-luas.

No problema, são definidos três parâmetros: r_1, r_2 e r_3. A partir deles, consideram-se duas variáveis aleatórias: θ, distribuída no intervalo $[0, \pi]$, e ρ, distribuída no intervalo $[r_1 - r_3/2, r_1 + r_3/2]$. Essas variáveis determinam a distribuição dos pontos gerados no plano xy. Os pontos da Região A possuem sinal desejado $d = 1$, enquanto os pontos da Região B possuem sinal desejado $d = -1$.

Para verificar a acurácia da KAN no problema das meias-luas com os parâmetros $r_1 = 2$, $r_2 = -0.8$ e $r_3 = 3$, considerou-se uma rede na configuração [2, 1, 1], isto é, 2 entradas, 1 neurônio na camada oculta e 1 saída, com *B-splines* de ordem $k = 3$ e 6 pontos de nó. Foram usados 1000 pontos no treinamento e 100 no teste. Além disso, considerou-se o otimizador Adam com parâmetros $\beta_1 = 0{,}9$ e $\beta_2 = 0{,}99$ [5].

A Figura 14 ilustra como a KAN foi capaz de delimitar bem uma fronteira de decisão que permite obter uma acurácia de 100% para o problema proposto. A KAN resultante do treinamento para essa fronteira de decisão é apresentada na Figura 15, a partir da qual foi possível extrair a seguinte expressão simbólica equivalente para o problema:

$$\begin{aligned} f(x,y) = 1{,}01 \tanh \Big(& 0{,}7207 \sin(0{,}9983x + 4{,}954) \\ & + 0{,}7207 \sin(0{,}5226y - 0{,}2717) + 1{,}646 \Big) \\ & + 0{,}000263. \end{aligned} \tag{26}$$

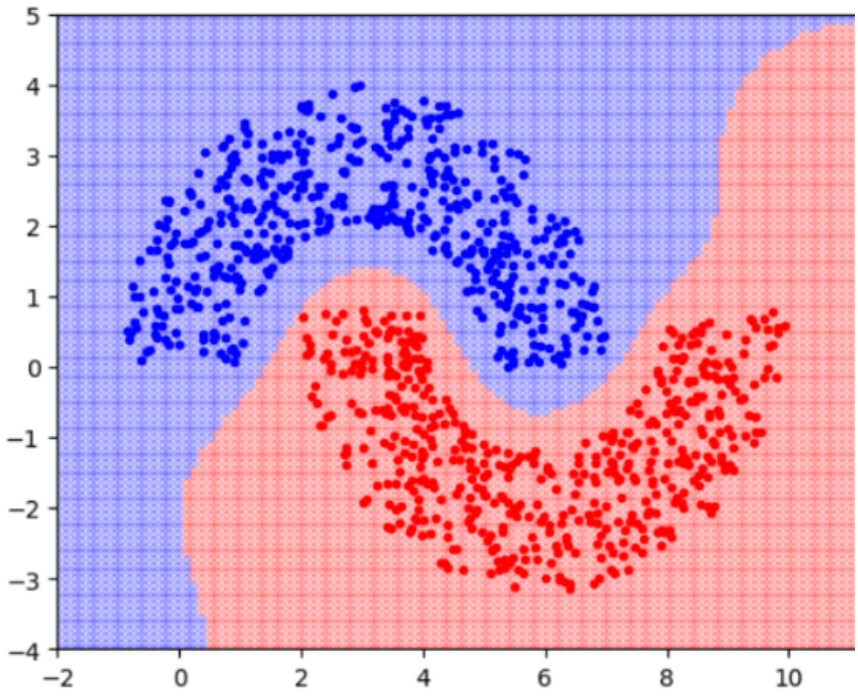

Figura 14: Fronteira definida pela rede proposta.

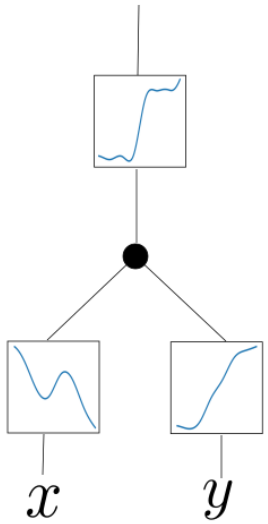

Figura 15: KAN treinada para o problema das meia-luas.

Para comparar a rede MLP e a KAN, procuramos as mais simples configurações de rede de modo que resolvessem o problema das meias-luas com 100% de acurácia média em uma sequência de 100 testes.

As redes selecionadas e treinadas para o comparativo foram: uma MLP com configuração [5, 1] e uma KAN com configuração [2, 1, 1] com *B-splines* de ordem $k = 3$ e 6 pontos de nó. Para ambas as redes, o otimizador Adam foi utilizado. A Tabela 2 apresenta o tempo levado para o treinamento e a quantidade de parâmetros treinados.

	KAN	MLP
Tempo (s)	506,6	280,7
Parâmetros treinados	21	21

Tabela 2: Tabela comparativa entre MLP e KAN.

É importante notar que, embora a KAN tenha utilizado o mesmo número de parâmetros durante o treinamento, a sua expressão extraída precisa de menos parâmetros para ser representada, como visto em (26).

4.2 Problemas de Regressão

Para avaliar a interpretabilidade da rede, foi selecionada e treinada uma KAN com configuração [2, 1, 1], utilizando *B-splines* de ordem $k = 3$ com 7 pontos de nó. Uma MLP com quatro neurônios na camada única e uma saída foi treinada também para fazer uma comparação de desempenho. O treinamento foi realizado com o otimizador Adam em ambas, configurado com $\beta_1 = 0.9$ e $\beta_2 = 0.99$. Em seguida, foram selecionadas três funções de duas variáveis cuja representação de Kolmogorov-Arnold pode ser expressa na forma

$$f(x,y) = \Phi_0(\phi_{0,1}(x) + \phi_{0,2}(y)).$$

A primeira função analisada é

$$f_1(x, y) = x + y, \quad (x,y) \in [0,1] \times [0,1].$$

Essa função pode ser representada na forma proposta com

$$\Phi_0(x) = x, \qquad \phi_{0,1}(x) = x, \qquad \phi_{0,2}(x) = x.$$

A Figura 16 ilustra a comparação entre a representação exata de Kolmogorov-Arnold e a KAN treinada, na qual é possível constatar que os gráficos das funções de ativação ajustadas são muito próximas às da expressão exata. O modelo resultante, ajustado pelo método dos mínimos quadrados, é expresso como

$$g_1(x,y) = 0.998x + 0.998y + 0.002.$$

A aproximação obtida pela rede é bastante precisa, apresentando coeficientes muito próximos dos valores esperados. O erro é pequeno, indicando que a KAN conseguiu capturar corretamente a estrutura linear da função proposta.

Os valores de desempenho obtidos foram:

- KAN: RMSE $= 0.012$, $R^2 = 0.985$
- MLP (4 neurônios): RMSE $= 0.005$, $R^2 = 0.996$

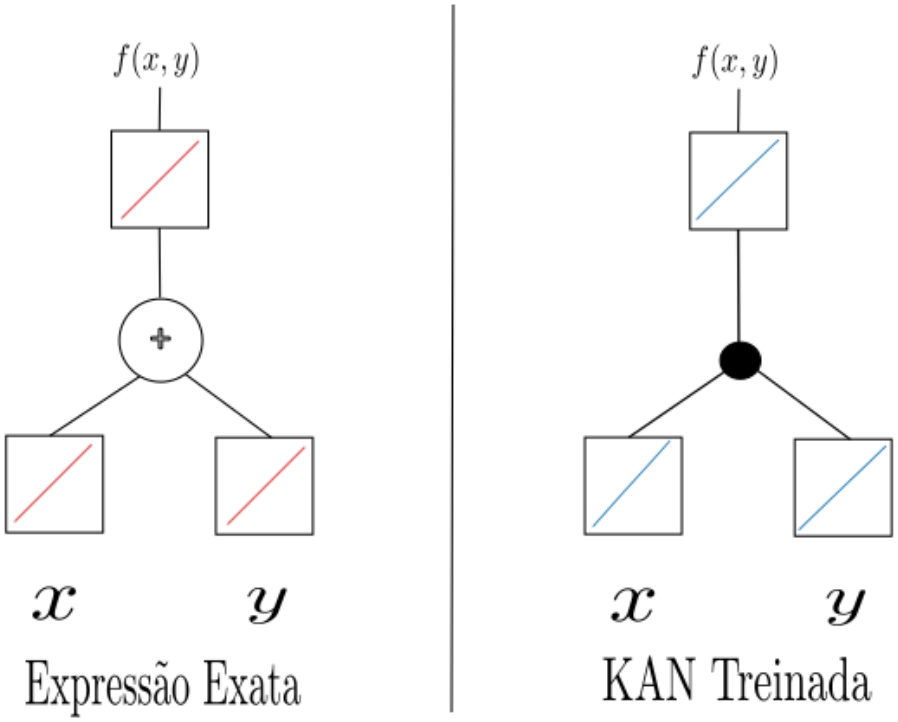

Figura 16: Comparação entre a representação exata da função f_1 e a rede treinada.

A segunda função considerada é

$$f_2(x,y) = xy, \quad (x,y) \in (0,1] \times (0,1].$$

Essa função admite diversas representações na forma proposta. Uma possível escolha é dada por

$$\Phi_0(x) = \exp(x), \qquad \phi_{0,1}(x) = \ln(x), \qquad \phi_{0,2}(x) = \ln(x).$$

A Figura 17 ilustra a comparação entre uma possível representação de Kolmogorov-Arnold e a KAN treinada. O modelo ajustado via mínimos quadrados não linear resultou na seguinte expressão

$$g_2(x,y) = \exp(0.765\ln(0.638x + 0.007) + 0.765\ln(0.902y + 0.011) + 3.501).$$

A aproximação para essa função é razoavelmente boa, mas apresenta um leve desvio nos coeficientes multiplicativos e aditivos. No entanto, dentro do intervalo considerado, a diferença é pequena o suficiente para que a função da rede ainda capture adequadamente o comportamento do produto entre as variáveis.

Os valores de desempenho foram:

- KAN: RMSE $= 0.018$, $R^2 = 0.972$
- MLP: RMSE $= 0.009$, $R^2 = 0.989$

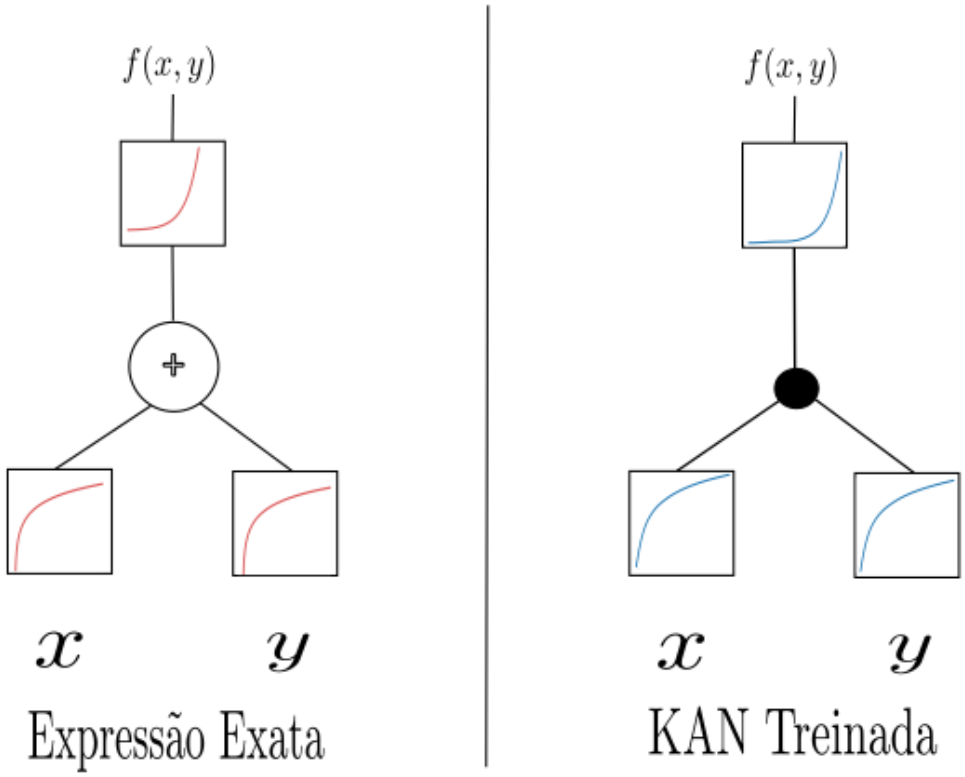

Figura 17: Comparação entre uma representação da função f_2 e a rede treinada.

A terceira função analisada é

$$f_3(x,y) = \sin(x) + \cos(2y), \quad (x,y) \in [-\pi,\pi] \times [-\pi,\pi].$$

A representação de Kolmogorov-Arnold para essa função pode ser descrita como

$$\Phi_0(x) = x, \qquad \phi_{0,1}(x) = \sin(x), \qquad \phi_{0,2}(x) = \cos(2x).$$

A Figura 18 ilustra as funções de ativação da KAN treinada para essa função. Utilizando mínimos quadrados não linear, obteve-se a seguinte expressão aproximada:

$$g_3(x,y) = 1.01\sin(1.000x - 0.004) + 1.00\sin(2.001y + 1.571) + 2.000.$$

A aproximação é muito boa, pois os coeficientes estão extremamente próximos dos valores exatos esperados. Há apenas desvios mínimos nos coeficientes e deslocamentos, e o comportamento oscilatório característico da função original foi capturado de forma bastante precisa.

Os resultados obtidos para essa função indicam a superioridade da KAN:

- KAN: RMSE = 0.024, $R^2 = 0.918$
- MLP: RMSE = 0.062, $R^2 = 0.895$

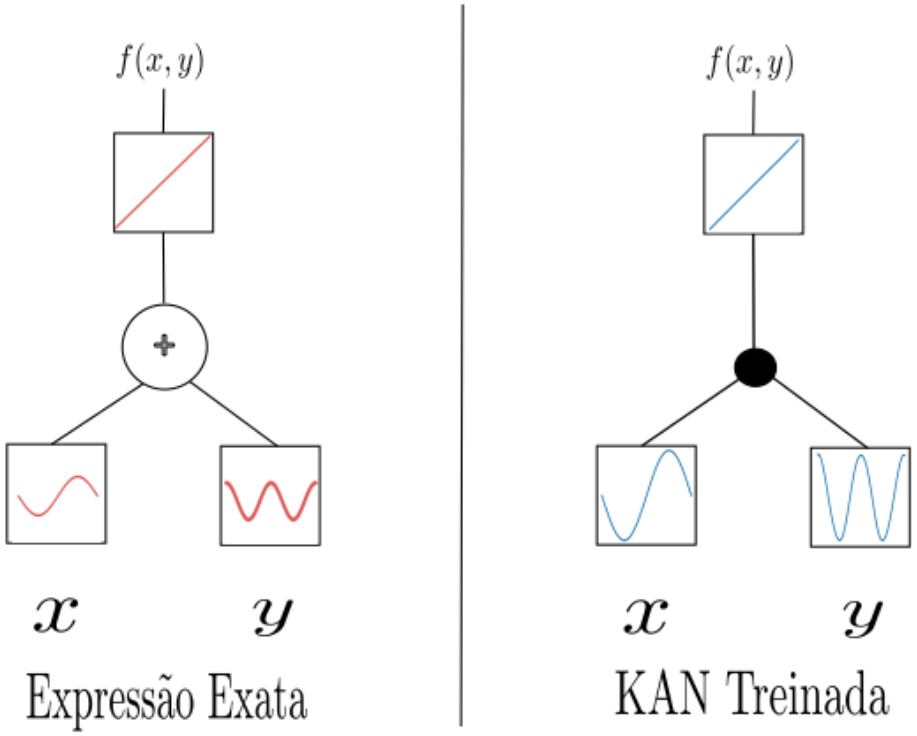

Figura 18: Comparação entre a representação da função f_3 e a rede treinada.

De modo geral, a rede treinada conseguiu capturar bem as três funções, com pequenas distorções nos coeficientes que podem ser explicadas pela aproximação numérica e pela estrutura da KAN. Essas diferenças são pequenas o suficiente para garantir que, nos intervalos analisados, a rede tenha aprendido as estruturas fundamentais das funções originais. Além disso, observou-se que, embora a MLP tenha apresentado desempenho superior nas funções mais simples, a KAN demonstrou vantagem ao lidar com estruturas mais complexas e não lineares, evidenciando seu potencial interpretativo.

4.3 Problema de Classificação Financeira

O *Statlog (German Credit Data)* é uma base da dados que contém registros de 1000 clientes de uma instituição bancária alemã, descritos por 20 atributos de natureza categórica e numérica, que abrangem desde informações pessoais até histórico de crédito e detalhes de operações financeiras. O conjunto de dados faz a distinção de clientes entre bons ($d = 1$) e maus pagadores ($d = -1$).

Para garantir equivalência no custo computacional, a KAN e a MLP foram configuradas com arquiteturas distintas, porém comparáveis. A KAN foi estruturada com duas camadas contendo 2 e 1 neurônios, enquanto a MLP utilizou três camadas compostas por 20, 13 e 1 neurônio, respectivamente. Na KAN, adotaram-se os parâmetros $k = 3$ e $grid = 5$. Já na MLP, empregou-se a função de ativação ReLU nas camadas ocultas e a tangente hiperbólica na camada de saída. Ambas as redes foram treinadas com o otimizador Adam ($\beta_1 = 0{,}9$ e $\beta_2 = 0{,}999$) ao longo de 1000 épocas.

A KAN atingiu uma acurácia de 68%, enquanto a MLP alcançou 77%. A MLP convergiu mais rapidamente que a KAN, como pode ser observado na Figura 19. Em termos de custo computacional, a MLP treinou 707 parâmetros por 17,3 s, ao passo que a KAN treinou 714 parâmetros por 33,2 s. Apesar do desempenho inferior ao da MLP, simplificando a KAN com a técnica de poda ($\theta = 0{,}1$) e utilizando o método dos mínimos quadrados não linear, foi possível extrair uma expressão de saída da KAN.

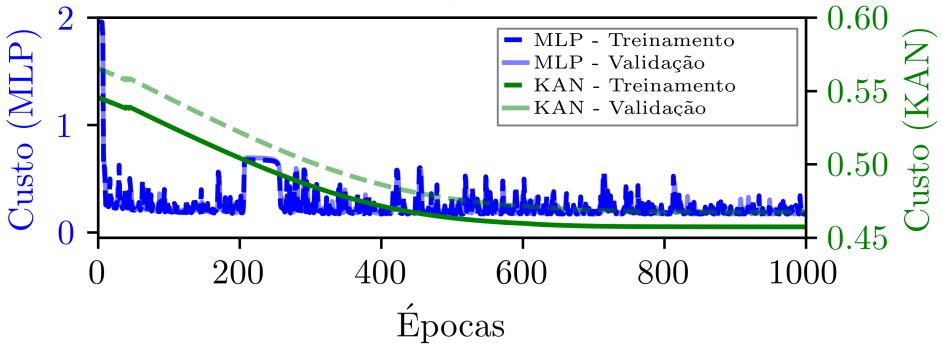

Figura 19: Evolução do custo durante o treinamento dos modelos KAN e MLP.

As funções de ativação treinadas da KAN foram ajustadas considerando três candidatos: afim, cosseno e tangente hiperbólica. Para selecionar o melhor ajuste de cada função de ativação, adotou-se o maior valor do coeficiente de determinação. Ao final do processo, a composição das funções resultou em:

$$\begin{aligned}\hat{d} = \; & -0{,}025\cos(-0{,}032x_2 + 1{,}842\cdot 10^{-4}x_5 + 6{,}374) \\ & + 0{,}210\tanh(2{,}151x_3 + 1{,}235x_2 + 2{,}198x_4) \\ & - 0{,}009x_5 + 45{,}244 - 0{,}116.\end{aligned}$$

A expressão obtida apresenta forma relativamente compacta e atinge a mesma acurácia da KAN nos dados de teste. Além disso, ela permite avaliar a relevância de cada componente da entrada (x_k, $k = 1, \ldots, 20$) no processo de decisão de crédito, característica que não é facilmente alcançada com a MLP. Dessa forma, a KAN mostra-se novamente ser um modelo promissor em cenários nos quais a interpretabilidade das decisões é um requisito desejável.

4.4 Classificação multi-classe com o MNIST

A base de dados *MNIST* (*Modified National Institute of Standards and Technology*) é composta por 70.000 imagens em tons de cinza de dígitos manuscritos entre 0 e 9, cada uma com resolução de 28×28 pixels. O conjunto está dividido em 60.000 amostras para treinamento e 10.000 para teste.

Devido à natureza da arquitetura KAN, o treinamento sobre imagens de alta dimensão é bastante custoso em termos computacionais. Assim, as imagens foram reduzidas para 7×7 pixels com aplicação de *anti-aliasing*, o que tornou as simulações viáveis.

Para resolver o problema com a KAN, foi empregado um modelo de rede com duas camadas ocultas de 5 neurônios cada e uma camada de saída com 10 neurônios. Esse modelo foi treinado em três cenários distintos, cada um por 250 épocas:

- Treinamento com $grid = 10$ fixo durante todas as épocas;
- Treinamento com $grid = 5$ na primeira metade e refinamento para $grid = 10$ na segunda metade;
- Treinamento com $grid = 3$ no primeiro terço, refinamento para $grid = 5$ no segundo terço e, por fim, refinamento para $grid = 10$ no último terço.

Os modelos de MLP propostos para comparação foram:

- Modelo 1: duas camadas ocultas com 5 neurônios cada e camada de saída com 10 neurônios;
- Modelo 2: duas camadas ocultas com 63 e 41 neurônios, respectivamente, e camada de saída com 10 neurônios;
- Modelo 3: três camadas ocultas com 60, 34 e 25 neurônios, respectivamente, e camada de saída com 10 neurônios;
- Modelo 4: quatro camadas ocultas com 64, 64, 32 e 16 neurônios, respectivamente, e camada de saída com 10 neurônios;
- Modelo 5: cinco camadas ocultas com 128, 64, 32, 16 e 4 neurônios, respectivamente, e camada de saída com 10 neurônios.

O primeiro modelo de MLP foi definido para equiparar o número de neurônios do modelo KAN. Os modelos 2 e 3 foram projetados para equiparar aproximadamente o número de parâmetros treináveis da KAN. Já os modelos 4 e 5 buscavam avaliar o comportamento de MLPs mais profundas com o MNIST reduzido.

Para todos os modelos de MLP, utilizou-se a função de ativação ReLU nas camadas ocultas e Softmax na saída. Também foi aplicada regularização por *Dropout* com taxa de

30% entre as camadas ocultas. Tanto o modelo KAN quanto as MLPs foram treinados com taxa de aprendizado $\eta = 0.001$ e o otimizador Adam, configurado com $\beta_1 = 0.9$ e $\beta_2 = 0.999$.

A Figura 20 apresenta a evolução do custo nos diferentes treinamentos do modelo KAN. Nota-se que as perturbações introduzidas pelo refinamento de *grid* são visíveis ao longo do treinamento, mas o desempenho final é semelhante nos três cenários, sugerindo que o refinamento não impactou de forma relevante o resultado.

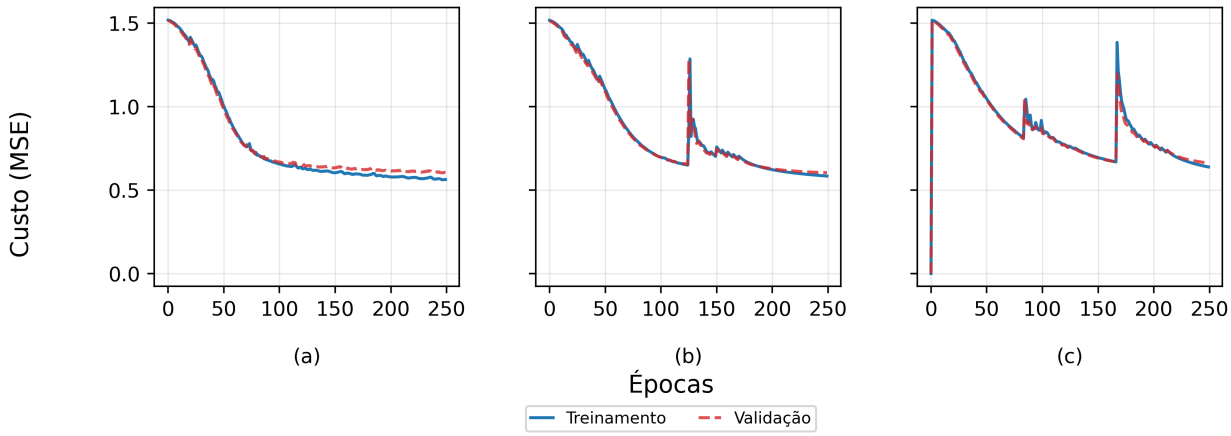

Figura 20: Evolução do custo durante o treinamento com o modelo KAN. (a) *Grid=10* fixo. (b) *Grid=5* na primeira metade e refinamento para *grid=10* na segunda metade. (c) *Grid=3* no primeiro terço, refinamento para *grid=5* no segundo terço e refinamento para *grid=10* no último terço.

A Figura 21 mostra a evolução do custo dos três primeiros modelos de MLP. O Modelo 1 (Fig. 21a) converge para uma perda maior que as dos Modelos 2 e 3 (Figs. 21b e 21c), o que indica baixa capacidade de representação, possivelmente pelo número reduzido de parâmetros treináveis. Embora não haja diferença marcante entre os Modelos 2 e 3, a acurácia final do Modelo 3 é superior, sugerindo que a adição de mais uma camada oculta favoreceu o treinamento, mesmo com número de parâmetros similar ao do Modelo 2.

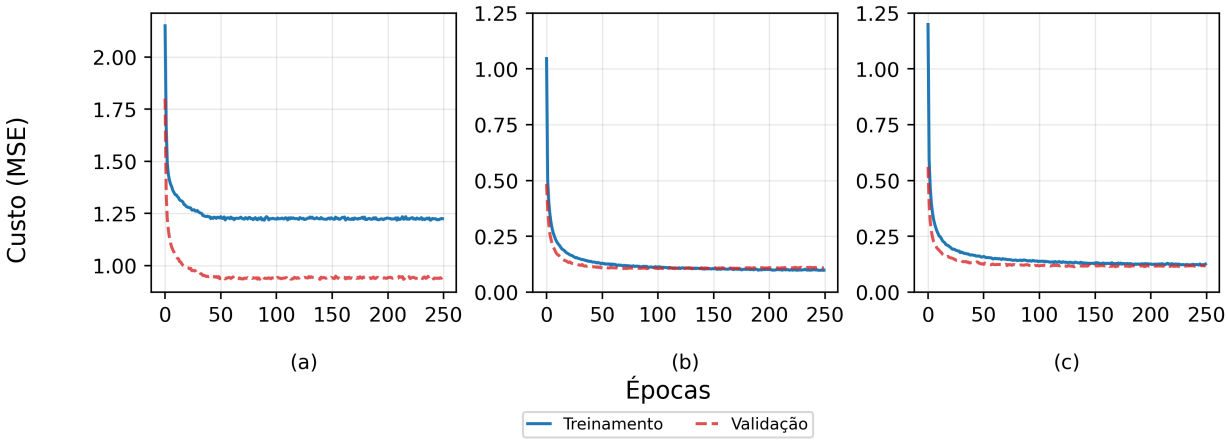

Figura 21: Evolução do custo durante o treinamento com as MLPs iniciais. (a) Modelo com número de neurônios equivalente à KAN. (b) Modelo com número de parâmetros equivalente. (c) Modelo com número de parâmetros equivalente, mas mais camadas ocultas.

Os resultados dos Modelos 4 e 5 estão ilustrados na Figura 22. O Modelo 4 (Fig. 22a) apresenta desempenho estável, enquanto o Modelo 5 (Fig. 22b) obteve resultados significativamente piores, sugerindo necessidade de ajustes adicionais, maior número de épocas ou até mesmo mais dados para convergir adequadamente.

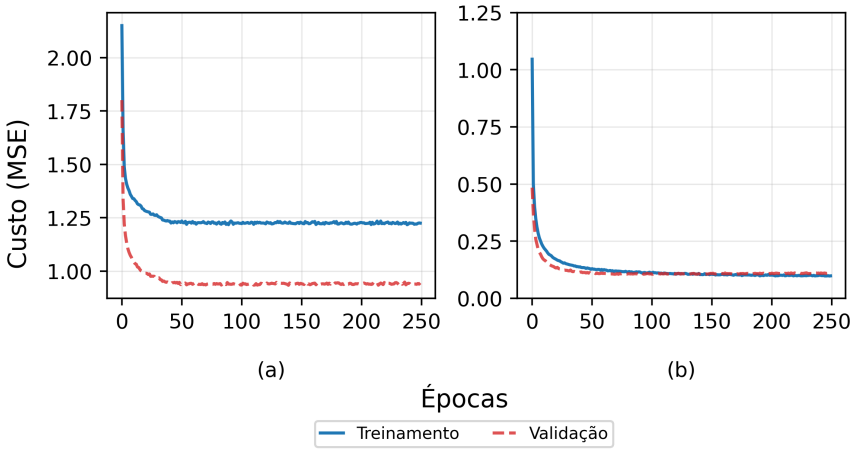

Figura 22: Evolução do custo durante o treinamento das MLPs mais profundas. (a) Modelo 4. (b) Modelo 5.

A Tabela 3 contém a acurácia e o número de parâmetros treináveis de todos os modelos treinados.

Tabela 3: Comparação entre modelos KAN e MLP.

Modelo/Treinamento	**Nº Parâmetros**	**Acc (%)**
KAN 1	6080	89,3
KAN 2	6080	88,94
KAN 3	6080	87,53
MLP 1	340	46,5
MLP 2	6194	73,12
MLP 3	6209	81,01
MLP 4	10138	82,6
MLP 5	17382	55,11

De forma geral, os experimentos indicam que a KAN apresentou desempenho superior aos modelos MLP equivalentes. Esse resultado sugere que, mesmo em cenários em que a interpretabilidade não seja o principal objetivo, a KAN pode oferecer ganhos relevantes de desempenho.

5 O algoritmo NoProp

Como uma alternativa ao algoritmo da retropropagação (*backpropagation*), foi proposto recentemente em [10] o algoritmo NoProp. Esse algoritmo não requer os cálculos progressivos e regressivos do *backpropagation* e funciona com base nos princípios seguidos pelos modelos de difusão para treinar cada camada de uma rede neural de forma independente, sem propagar gradientes. Segundo [10], as principais vantagens desse algoritmo são:

1. **Eficiência Computacional** – A retropropagação exige armazenamento e cálculo extensivo de gradientes. O NoProp remove grande parte dessa sobrecarga.

2. **Escalabilidade** – Como não há propagação reversa, o treinamento em redes muito profundas torna-se mais estável.

3. **Robustez** – Gradientes desaparecendo ou explodindo deixam de ser um problema central, pois não há retropropagação direta.

4. **Novas Perspectivas Biológicas** – O cérebro humano provavelmente não usa retropropagação. Métodos como o NoProp podem estar mais alinhados com a cognição biológica.

Durante o treinamento, cada camada da rede neural recebe um rótulo ruidoso e a entrada da rede, e prevê o rótulo alvo com base neles. Cada camada é treinada independentemente

das outras, utilizando uma perda de *denoising*. Isso elimina a necessidade da propagação durante o treinamento. No entanto, todas as camadas trabalham juntas durante a inferência.

Partindo de ruído Gaussiano, cada camada recebe um rótulo ruidoso produzido pela camada anterior e o refina (*denoising*). Esse processo é feito camada a camada até que, na última camada, a rede produza a classe correta. O objetivo é, dado um par (x,y) do conjunto de dados, construir um modelo capaz de prever o rótulo y a partir da entrada x. Em vez de simplesmente encontrar uma função $f(x) = y$, deseja-se treinar a rede neural para modelar um processo estocástico que transforma ruído aleatório em uma forma que permita estimar y.

5.1 Processo estocástico de difusão / *Denoising*

Esse processo é representado por p. A partir do ruído, ele é refinado em vários passos até chegar à representação final $z(T)$, usada para prever y. Matematicamente, trata-se da probabilidade conjunta de todas as representações ruidosas intermediárias $z(0), \ldots, z(T)$ do rótulo y dado x, ou seja,

$$p(z(t)_{t=0}^{T}|x) = \quad p(z(0)) \cdot \prod_{t=1}^{T} p(z(t)|z(t-1), x) \cdot p(y|z(T))$$

em que $p(z(0))$ descreve o ruído gaussiano padrão inicial, $p(z(t)|z(t-1), x)$ descreve como cada camada remove o ruído de sua entrada e $p(y|z(T))$ descreve como y é classificado com base na representação final $z(T)$. Esse termo, por sua vez, é parametrizado por uma rede neural $\hat{u}_{\theta_t}$, ponderada por escalares $a(t), b(t), c(t)$, levando a

$$z(t) = a(t)\, \hat{u}_{\theta_t}(z(t-1),x) \;+\; b(t) \cdot z(t-1) + \sqrt{c(t)}\, \epsilon(t)$$

com $\epsilon(t)$ sendo ruído gaussiano.

5.2 Processo reverso de ruído / Variacional posterior

Representado por q, a partir do rótulo y (na forma $u(y)$), esse processo modela como adicionar ruído passo a passo até obter $z(0)$, ou seja,

$$q(z(t)_{t=0}^{T}|y,x) = q(z(t)|y) \cdot \prod_{t=1}^{T} q(z(t-1)|z(t)).$$

Cabe observar que $q(z(t-1)|z(t))$ descreve o processo de difusão reversa que recupera representações ruidosas anteriores pela adição incremental de ruído. A distribuição condicional $q(z(T)|y)$ é dada por

$$q(z(T)|y) = \mathcal{N}(z(T)|\sqrt{\bar{\alpha}(T)}\, u(y), 1 - \bar{\alpha}(T)),$$

que representa uma distribuição gaussiana sobre a variável latente $z(T)$ com média $\sqrt{\bar{\alpha}(T)}u(y)$ e variância $1-\bar{\alpha}(T)$. $u(y)$ representa o rótulo "transformado" (*embedding*) e $\bar{\alpha}(T)$ diz o quanto de $u(y)$ permanece depois da adição de ruído. Além disso, a distribuição gaussiana sobre a variável latente $z(t-1)$ com média $\sqrt{\alpha(t-1)}z(t)$ e variância $1-\alpha(t-1)$ é dada por

$$q(z(t-1)|z(t)) = \mathcal{N}(z(t-1)|\sqrt{\alpha(t-1)}\, z(t), 1-\alpha(t-1)),$$

em que $\alpha(t-1)$ é um parâmetro de ruído que controla o quanto do sinal original é preservado no passo $t-1$.

5.3 Função de perda

No treinamento, o algoritmo NoProp tem por objetivo maximizar $\log p(y|x)$. Como isso é inviável computacionalmente, otimiza-se um limite inferior variacional (*evidence lower bound* – ELBO), ou seja,

$$\log p(y|x) \geq \text{ELBO}$$

ou ainda

$$\log p(y|x) \geq \mathrm{E}_{q(z(t)_{t=0}^{T}|y,x)} \left[\log p(z(t)_{t=0}^{T}|y,x)) - \log q(z(t)_{t=0}^{T}|y,x)\right].$$

A perda do algoritmo NoProp é derivada dessa expressão e pode ser escrita como a soma de três termos:

$$\begin{aligned}\mathcal{L}_{\text{NoProp}} =& \mathrm{E}_{z(T)\sim q}\left[-\log \hat{p}_\theta(y|z(T))\right] \\ &+ \mathcal{D}_{\text{KL}}(q(z(0)|y)||p(z(0))) \\ &+ \frac{T}{2}\eta \mathrm{E}_{t\sim\mathcal{U}(1,T)}\left[(\text{SNR}(t)-\text{SNR}(t-1))\|\hat{u}_{\theta_t}(z(t-1),x)-u(y)\|^2\right]. \end{aligned} \tag{27}$$

A derivação dessa expressão pode ser encontrada em [10]. O primeiro termo do lado direito representa a entropia cruzada, que mede quão precisa é a representação $z(T)$ para predizer corretamente y; o segundo termo é a divergência de Kullback–Leibler entre a distribuição de $z(0)$ e o ruído Gaussiano padrão, o que força que essas distribuições sejam similares para que o processo de difusão funcione corretamente; e o terceiro termo mede quão bem cada camada remove o ruído, comparando quão próximo sua saída está do rótulo transformado $u(y)$. Além disso, η é um hiperparâmetro e a razão sinal ruído é definida como

$$\text{SNR}(t) = \frac{\bar{\alpha}(t)}{1-\bar{\alpha}(t)}.$$

A SNR(t) aumenta a medida que o algoritmo se move para camadas posteriores (aumento de t), o que faz com que a perda aumente. Em outras palavras, o modelo penaliza mais os erros das camadas posteriores do que das camadas iniciais da rede.

5.4 Treinamento e inferência

Durante o treinamento, cada camada aprende a remover ruído de $u(y)$ sem necessidade de uma propagação direta (*forward*) ou reversa (*backward*) pela rede inteira. Para um par (x,y), obtém-se $u(y)$ a partir de uma matriz W_{embed}. Adiciona-se ruído em $u(y)$ para obter $z(t)$, e cada camada $\hat{u}_\theta(z(t-1), x)$ é treinada independentemente para prever $u(y)$ a partir de x e $z(t-1)$. A perda é calculada e os parâmetros da rede são atualizados para maximizar essa perda, usando um otimizador.

Na inferência, a rede com T camadas recebe inicialmente o ruído $z(0)$. Cada camada utiliza sequencialmente a saída $z(t-1)$ da camada anterior, que junto com a entrada x da rede produz a representação ruidosa seguinte $z(t)$. Isso resulta em uma sequência de representações dada por $z(0), z(1), \cdots z(T-1), z(T)$. No passo final $t = T$, $z(T)$ é passada por um decisor para gerar a predição $\hat{y}$, como esquematizado na Figura 23.

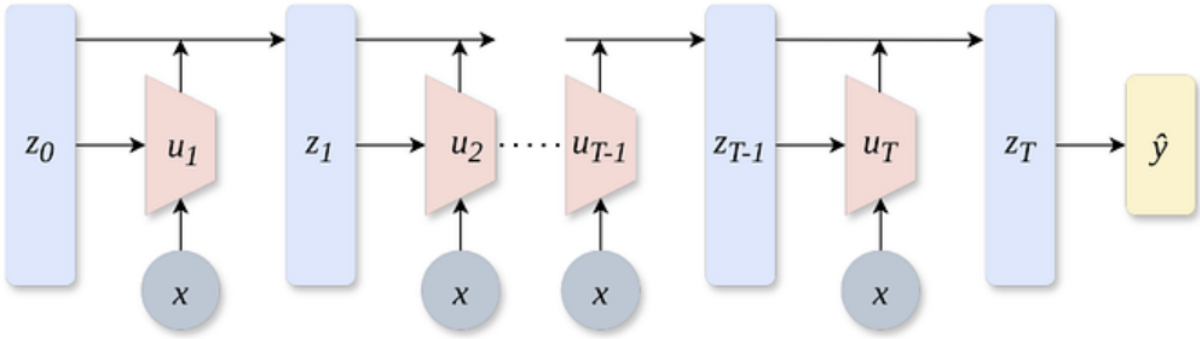

Figura 23: Esquema do algoritmo NoProp. $z(0) = z_0$ é um ruído gaussiano, que é transformado sucessivamente até obter $z(T) = z_T$. Cada camada, representada por $u(t) = u_t$, recebe a entrada x da rede. A camada final consegue obter a predição $\hat{y}$ do rótulo y a partir de $z(T)$. Fonte: [10].

5.5 Arquitetura das camadas

Cada camada $\hat{u}_\theta(z(t-1),x)$ consiste em um caminho convolucional para processar x; um caminho totalmente conectado para processar $z(t-1)$ com conexões residuais; e camadas adicionais totalmente conectadas para gerar *logits*. Os *logits* passam por uma função *softmax*, produzindo uma distribuição sobre as classes. A saída final é obtida calculando-se uma soma ponderada dos *embeddings* de classe, como esquematizado na Figura 24.

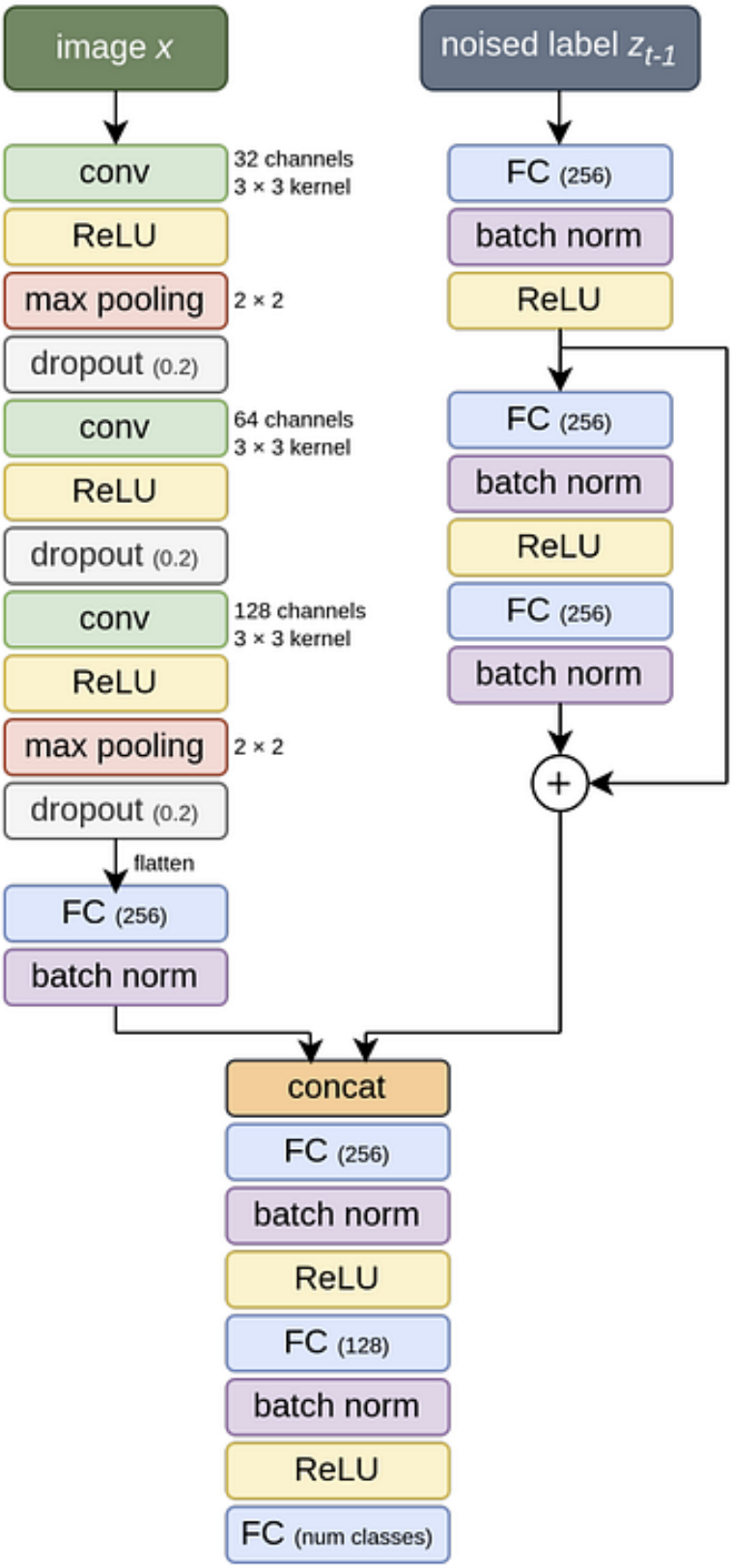

Figura 24: Arquitetura de cada camada Fonte: [10].

A seguir, o NoProp é testado em problemas de classificação.

6 Aplicação do NoProp em problemas de classificação

Nesta seção, avalia-se o desempenho do NoProp em três problemas de classificação distintos: o problema das meias-luas, o *Statlog* (*German Credit Data*) e, por fim, o MNIST com dimensões reduzidas. Em todos os casos, a abordagem considerada foi o algoritmo NoProp em tempo discreto. Cabe observar que o impacto de diferentes hiperparâmetros no treinamento da rede para cada problema proposto também foi analisado.

6.1 NoProp aplicado ao problema das meias-Luas

Para este problema, foram definidas duas configurações distintas para os blocos de *denoising*. A primeira consiste em um modelo com apenas uma camada densa, contendo dois neurônios de saída. A segunda configuração é composta por duas camadas densas ocultas, com 64 e 32 neurônios, respectivamente, além de uma camada de saída com dois neurônios. Em ambas as configurações, foram realizados testes com diferentes números de blocos de *denoising*, considerando $T = 5$ e $T = 10$, e o treinamento individual de cada bloco foi com o *Backpropagation* utilizando a taxa de aprendizado $\eta = 0.001$ e Adam com $\beta_1 = 0.9$ e $\beta_2 = 0.999$.

Na primeira configuração, foram testados dois cenários:

1. No primeiro, a dimensão do vetor de latência (*embedding dim*) foi igualada à dimensão da entrada, possibilitando a comparação entre as operações de soma e concatenação dos vetores de ruído à entrada. A evolução da acurácia durante o treinamento nesses casos está ilustrada na Figura 25 (a) e (b).

2. No segundo cenário, utilizou-se um vetor de latência de dimensão 16 concatenado à entrada. A Figura 26 apresenta os resultados obtidos nesse caso.

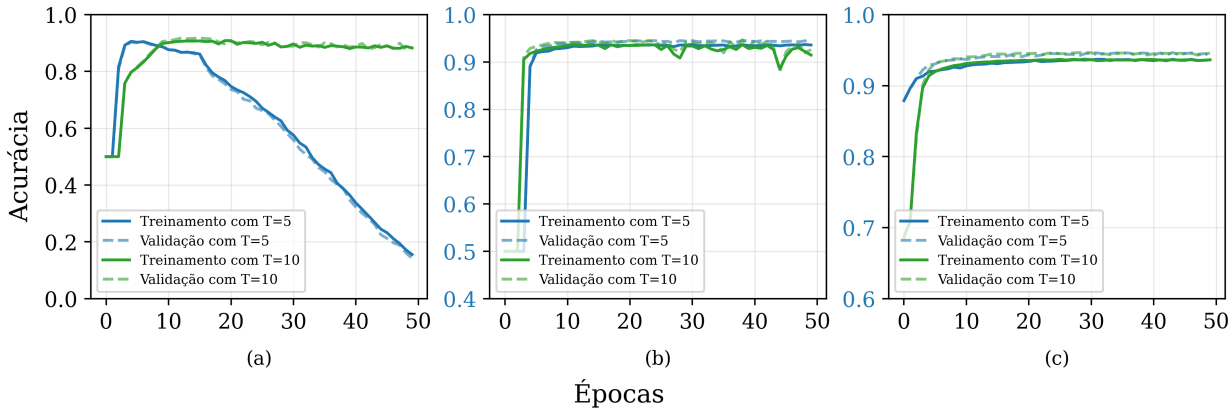

Figura 25: Evolução da acurácia durante o treinamento com a primeira configuração. (a) Cenário com soma dos ruídos à entrada. (b) Cenário com concatenação dos ruídos à entrada. (c) Cenário com vetor de latência de dimensão 16.

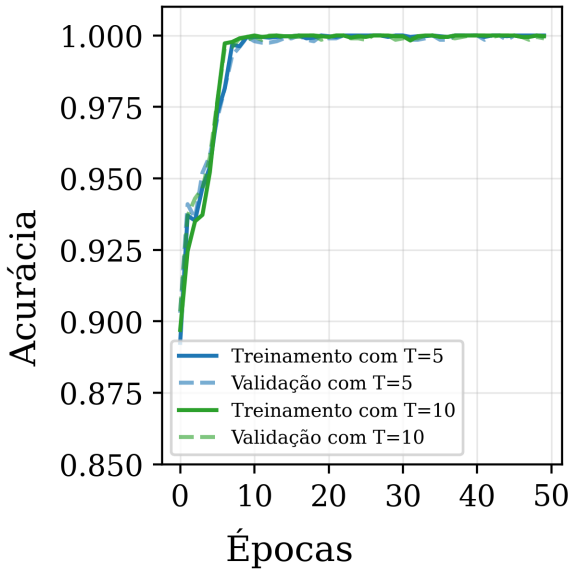

Figura 26: Evolução da acurácia durante o treinamento com a segunda configuração.

A Tabela 4 resume as configurações avaliadas e os resultados finais obtidos.

Tabela 4: Resultados dos modelos com diferentes configurações no problema das meias-luas.

Configuração	T	Operação na Entrada	Acc (%)	Parâmetros
1	5	Soma	14,15	40
1	10	Soma	88,55	70
1	5	Concatenação	94,45	60
1	10	Concatenação	92,60	110
1	5	Concatenação	94,30	256
1	10	Concatenação	94,55	446
2	5	Concatenação	100,0	12.316
2	10	Concatenação	100,0	24.566

Os resultados indicam que o NoProp não apresenta bom desempenho quando os vetores de ruído são somados diretamente à entrada, ainda que essa operação pudesse reduzir o número de parâmetros treináveis. Por outro lado, a concatenação mostrou-se mais eficaz, alcançando acurácias elevadas apesar do número maior de parâmetros treináveis.

Além disso, verificou-se que o número de blocos de *denoising* (T) influencia apenas marginalmente a velocidade de convergência, podendo trazer ganhos mais expressivos apenas em arquiteturas mais simples. No entanto, no geral, esse hiperparâmetro não altera substancialmente a acurácia final.

Por fim, observou-se que, embora seja possível atingir acurácia de 100% com a configuração mais complexa, o custo em termos de parâmetros treináveis é significativamente maior que as outras propostas já verificadas. Essa característica pode se tornar um obstáculo em cenários com recursos computacionais limitados, nos quais a paralelização do treinamento não seja viável.

6.2 NoProp aplicado ao Problema de Classificação Financeira

Utilizou-se novamente a base de dados *Statlog* (*German Credit Data*) para o problema de classificação no contexto financeiro. Nesta etapa, os dados foram previamente transformados para o formato *one-hot encoding*, a fim de facilitar o treinamento dos modelos.

Foram avaliadas três configurações distintas para os blocos de *denoising*:

1. A primeira consiste em um modelo simples, com apenas uma camada de dois neurônios de saída;

2. A segunda inclui uma camada oculta com 16 neurônios e uma camada de saída com dois neurônios;

3. A terceira apresenta duas camadas ocultas, com 64 e 32 neurônios, respectivamente, além da camada de saída.

Em todas as configurações foram realizados experimentos com $T = 5$ e $T = 20$ combinado com $embedding\_dim = 61$ e $embedding\_dim = 128$. Apenas operação de concatenação entre entrada e ruído foi utilizada. Para os modelos com camadas ocultas, aplicou-se *Dropout* de 20% como forma de regularização. O treinamento individual de cada bloco foi conduzido com *Backpropagation*, utilizando taxa de aprendizado $\eta = 0.001$ e o otimizador Adam com $\beta_1 = 0.9$ e $\beta_2 = 0.999$.

A Figura 27 mostra os resultados obtidos para a primeira configuração. Observa-se que a variação de hiperparâmetros em (b) e (c) não produziu diferenças significativas em relação a (a), tanto na evolução do treinamento quanto na acurácia final, que se manteve praticamente inalterada.

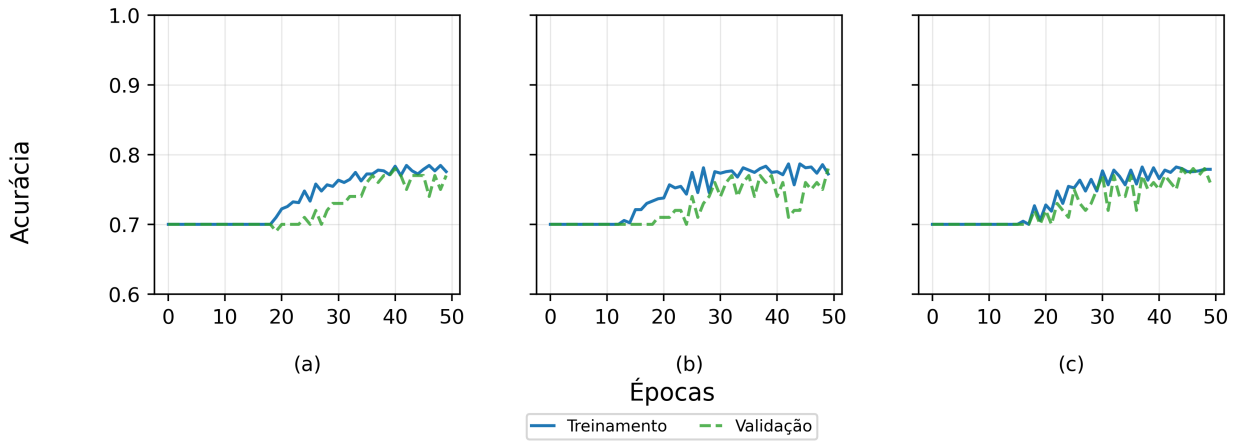

Figura 27: Evolução da acurácia durante o treinamento com a primeira configuração proposta para a classificação financeira. (a) $T = 5$, $embedding\_dim = 61$. (b) $T = 5$, $embedding\_dim = 128$. (c) $T = 20$, $embedding\_dim = 61$.

No treinamento da segunda configuração, apresentado na Figura 28, nota-se uma pequena diferença em (b) e uma melhora marginal em (c) em comparação com (a). Entretanto, no geral, os resultados de validação são ligeiramente inferiores aos obtidos pela primeira configuração.

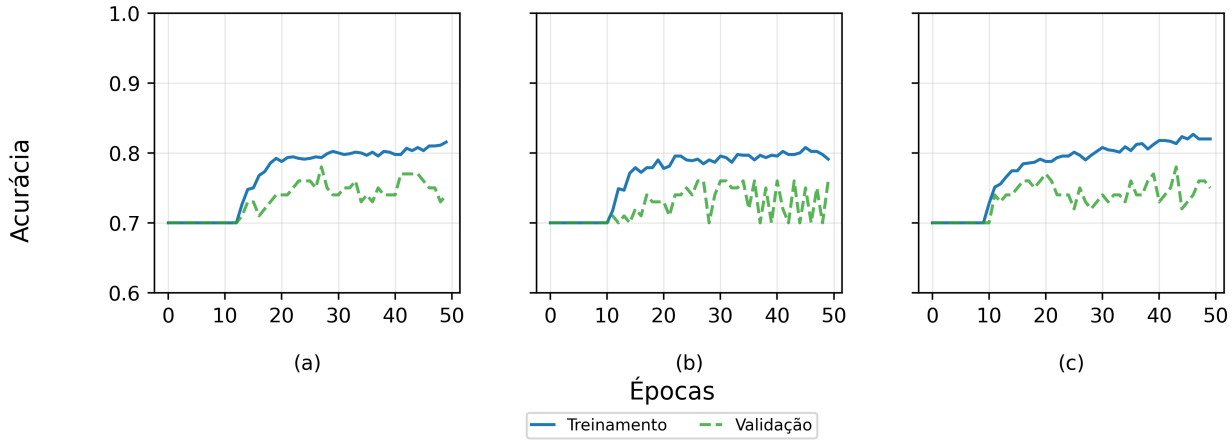

Figura 28: Evolução da acurácia durante o treinamento com a segunda configuração proposta. (a) $T = 5$, *embedding_dim* $= 61$. (b) $T = 5$, *embedding_dim* $= 128$. (c) $T = 20$, *embedding_dim* $= 61$.

A Figura 29 apresenta os resultados da terceira configuração. Nesse caso, os modelos sofrem de *overfitting*. A inclusão de mais blocos de (a) para (b) reduziu a velocidade de convergência, mas não houve diferença perceptível entre (a) e (c). A acurácia nos dados de validação foi pior em comparação às demais configurações, justamente em razão do *overfitting*.

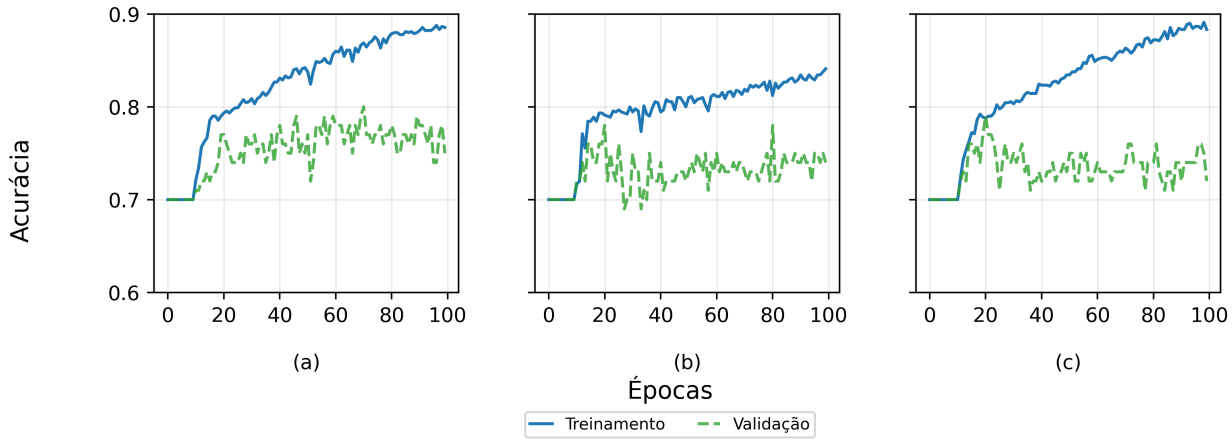

Figura 29: Evolução da acurácia durante o treinamento com a terceira configuração proposta. (a) $T = 5$, *embedding_dim* $= 61$. (b) $T = 5$, *embedding_dim* $= 128$. (c) $T = 20$, *embedding_dim* $= 61$.

A Tabela 5 resume os resultados obtidos em todas as configurações avaliadas.

Tabela 5: Resumo dos resultados para as diferentes configurações avaliadas.

Configuração	**embedding_dim**	**T**	**Acc (%)**	**Parâmetros**
1	61	5	77	1476
1	128	5	78	2414
1	61	20	76	5166
2	61	5	74	10416
2	128	5	76	16044
2	61	20	75	40926
3	61	5	75	45776
3	128	5	74	67484
3	61	20	72	182366

De forma geral, os resultados são consistentes com os obtidos em outras redes analisadas. Destaca-se que os modelos com blocos de camada única apresentaram o melhor desempenho, o que é promissor, pois sugere a possibilidade de treinar blocos individualmente sem a necessidade do *backpropagation* para esse problema em específico.

6.3 NoProp aplicado ao MNIST com Dimensões Reduzidas

O NoProp foi inicialmente avaliada no problema de classificação de imagens do MNIST, demonstrando a viabilidade de realizar tarefas desse tipo com uma arquitetura altamente paralelizável e sem a necessidade de propagação global do erro. Entretanto, no artigo original, os blocos de *denoising* são compostos por redes convolucionais relativamente complexas, cujo treinamento não é trivial.

Com o intuito de verificar se é possível resolver o problema do MNIST utilizando blocos mais simples, foram definidas três configurações compostas apenas por camadas densas:

1. Primeiro, uma configuração simples, com apenas uma camada de saída contendo dez neurônios;

2. Segundo, uma configuração com duas camadas ocultas com 64 e 16 neurônios, respectivamente, seguidas de uma camada de saída com dez neurônios. Entre as camadas ocultas, aplicaram-se normalização Batch e regularização *Dropout* com taxa de 20%;

3. Terceiro, uma configuração com quatro camadas ocultas com 256, 128, 64 e 16 neurônios, respectivamente, seguidas de uma camada de saída com dez neurônios. Entre as camadas ocultas, também foram aplicados Batch Normalization e *Dropout* com taxa de 20%.

Todos os modelos foram treinados com $T = 10$ blocos de *denoising* e dimensão de embedding $embedding\_dim = 128$, mantidos fixos. Apenas operação de concatenação entre entrada

e ruído foi utilizada. O treinamento individual de cada bloco foi realizado com *Backpropagation*, utilizando taxa de aprendizado $\eta = 0.001$ e o otimizador Adam, com parâmetros $\beta_1 = 0.9$ e $\beta_2 = 0.999$. Para viabilizar a comparação com outras arquiteturas avaliadas e reduzir o custo computacional, a base de dados foi pré-processada, redimensionando as imagens para 7×7 com *anti-aliasing*.

A Figura 30 apresenta a evolução da acurácia durante o treinamento para as três configurações propostas. Observa-se que os modelos testados não foram capazes de alcançar bons resultados no MNIST reduzido. Além disso, o aumento da profundidade das redes internas dos blocos de *denoising* levou, paradoxalmente, a um desempenho ainda pior.

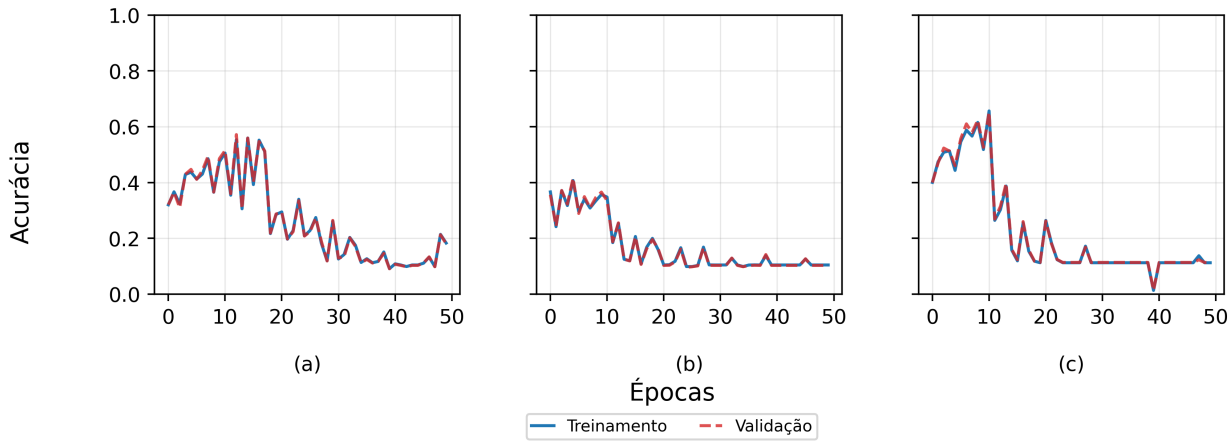

Figura 30: Evolução da acurácia durante o treinamento no MNIST reduzido. (a) Primeira configuração. (b) Segunda configuração. (c) Terceira configuração.

Esses resultados sugerem que o NoProp, quando aplicado a blocos exclusivamente densos, não é capaz de resolver adequadamente o problema de classificação no MNIST com dimensões reduzidas. Além disso, tal limitação é particularmente desinteressante, uma vez que ainda não existem métodos consistentes para treinar blocos individuais sem o uso de *backpropagation*. Em comparação com as outras redes analisadas, o NoProp apresentou os piores resultados.

7 Conclusões

A aplicação da KAN em diferentes cenários de classificação e regressão demonstrou resultados bastante expressivos, destacando-se principalmente pela capacidade de extrair representações simbólicas compactas e interpretáveis. Nos problemas de regressão, a rede mostrou-se capaz de aproximar funções lineares e não lineares com boa precisão, fornecendo expressões

analíticas muito próximas das formas originais, o que a diferencia de arquiteturas tradicionais como as MLPs. Já nos problemas de classificação, como o das meias-luas e o MNIST reduzido, a KAN apresentou desempenho superior ou, no mínimo, competitivo em relação às MLPs equivalentes, evidenciando seu potencial tanto em termos de acurácia quanto de interpretabilidade. Em aplicações práticas, como o caso do *Statlog* (*German Credit Data*), embora a acurácia obtida tenha sido inferior à da MLP, a KAN mostrou-se promissora por possibilitar a extração de expressões simbólicas que permitem avaliar a relevância de variáveis de entrada, algo de grande valor em contextos onde a transparência do modelo é fundamental.

No caso do NoProp, os resultados obtidos confirmam que a arquitetura é conceitualmente interessante, pois abre caminho para o treinamento altamente paralelizável de redes sem a necessidade de propagação global do erro. Entretanto, os experimentos mostraram que, quando os blocos de *denoising* são compostos apenas por camadas densas, o desempenho ainda é limitado, especialmente em tarefas mais complexas como o MNIST. Observou-se que, em problemas mais simples, como o das meias-luas ou a classificação financeira, o NoProp pode atingir bons resultados com arquiteturas relativamente pequenas. No entanto, o fato de que cada bloco ainda precisa ser treinado com *backpropagation* limita o alcance prático da abordagem. Seria mais promissor se os blocos de *denoising* pudessem ser simplificados de tal forma que métodos alternativos de treinamento pudessem ser aplicados em nível local, eliminando completamente a dependência do *backpropagation* em todos os níveis e reforçando o caráter altamente paralelizável do método.

De forma geral, os objetivos propostos neste trabalho foram alcançados, uma vez que tanto a KAN quanto o NoProp foram avaliados em diferentes contextos e comparados a MLPs, revelando suas principais vantagens e limitações. Ainda assim, abrem-se caminhos interessantes para trabalhos futuros: no caso do NoProp, investigar estratégias alternativas de treinamento dos blocos de *denoising*; no caso da KAN, explorar novas aplicações, como a solução de equações diferenciais ordinárias (EDOs), nas quais sua capacidade interpretativa poderia se mostrar ainda mais relevante. Essas extensões poderiam não apenas reforçar os resultados obtidos, mas também expandir significativamente o leque de aplicações dessas arquiteturas.

A Método dos Mínimos Quadrados Não Linear

Dado um conjunto de dados $\{(x_i, y_i)\}_{i=1}^{n}$, deseja-se encontrar os parâmetros $\boldsymbol{\theta}$ que minimizam a soma dos quadrados dos resíduos, expressa de forma matricial como

$$S(\boldsymbol{\theta}) = \|\mathbf{y} - f(\mathbf{X}, \boldsymbol{\theta})\|^2. \tag{28}$$

A minimização de $S(\boldsymbol{\theta})$ normalmente requer métodos iterativos, pois não há solução fechada para problemas não lineares [13]. Um dos métodos mais utilizados é o método de Gauss-

Newton, que aproxima a função por uma expansão de Taylor de primeira ordem e resolve um sistema linear iterativamente [3].

Para Exemplificar, considere um modelo genérico da forma

$$g(x) = a + bf(cx + d), \tag{29}$$

em que $\boldsymbol{\theta} = [a\ b\ c\ d]$ representa os parâmetros a serem ajustados e f é uma função não linear qualquer [1].

Dado um conjunto de observações $\{(x_i, y_i)\}$, define-se o vetor de resíduos como

$$\mathbf{r}(\boldsymbol{\theta}) = \mathbf{y} - g(\mathbf{x}, \boldsymbol{\theta}). \tag{30}$$

A matriz Jacobiana $\mathbf{J}(\boldsymbol{\theta})$, contendo as derivadas parciais dos resíduos em relação aos parâmetros, é definida como

$$\mathbf{J}_{ij} = \frac{\partial r_i}{\partial \theta_j}. \tag{31}$$

Para este modelo, os elementos da matriz Jacobiana são dados por:

$$\mathbf{J} = \begin{bmatrix} -1 & -f'(cx_1 + d) & -bf'(cx_1 + d)x_1 & -bf'(cx_1 + d) \\ -1 & -f'(cx_2 + d) & -bf'(cx_2 + d)x_2 & -bf'(cx_2 + d) \\ \vdots & \vdots & \vdots & \vdots \\ -1 & -f'(cx_n + d) & -bf'(cx_n + d)x_n & -bf'(cx_n + d) \end{bmatrix}. \tag{32}$$

A atualização iterativa dos parâmetros ocorre resolvendo o sistema

$$(\mathbf{J}^T\mathbf{J})\Delta\boldsymbol{\theta} = \mathbf{J}^T\mathbf{r}, \tag{33}$$

em que $\Delta\boldsymbol{\theta}$ representa a correção nos parâmetros e $\mathbf{r}$ é o vetor de resíduos [12].

Os parâmetros são então atualizados como

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}. \tag{34}$$

Para iniciar o processo iterativo, é necessário fornecer um chute inicial $\boldsymbol{\theta}^{(0)}$. A escolha de um bom valor inicial pode impactar significativamente a convergência do método. Em geral, aproximações baseadas em conhecimento prévio do problema podem ser utilizadas para definir esse valor [1]. Esse processo é repetido até que a variação nos parâmetros seja suficientemente pequena ou um critério de parada seja atingido.

O método de Gauss-Newton se mostra eficiente quando os resíduos são aproximadamente lineares em relação aos parâmetros, garantindo uma boa convergência na maioria dos casos práticos [3].

Referências

[1] D. M. Bates and D. G. Watts. *Nonlinear Regression Analysis and Its Applications*. John Wiley & Sons, 1988.

[2] C. M. Bishop. *Deep Learning: Foundations and Concepts*. Springer, 2024.

[3] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[5] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press.

[6] S. Haykin. *Neural networks and learning machines*. Prentice Hall, Upper Saddle River, 3 edition, 2009.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, https://arxiv.org/abs/1412.6980, 2014.

[8] A. N. Kolmogorov. On the representation of continuous functions of several variables as superpositions of continuous functions of a smaller number of variables. *Dokl. Akad. Nauk*, 1956.

[9] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[10] Q. Li, Y. W. Teh and R. Pascanu. NoProp: Training Neural Networks without Back-propagation or Forward-propagation. *arXiv*, https://arxiv.org/abs/2503.24322, 2025.

[11] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljacic, T. Y. Hou, and M. Tegmark. KAN: Kolmogorov-Arnold networks. *arXiv*, https://arxiv.org/abs/2404.19756, 2024.

[12] J. J. Moré. The levenberg-marquardt algorithm: implementation and theory. *Numerical Analysis*, pages 105–116, 1978.

[13] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2006.

[14] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[15] D. E. Rumelhart, G. E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[16] L. Schumaker. *Spline Functions: Computational Methods*. SIAM, 2015.

Anexo 1 - Artigo aceito no SBrT 2025

Comparações entre a rede Kolmogorov-Arnold e a rede perceptron multicamada

Pedro H. S. Soares, Renato Candido e Magno T. M. Silva

***Resumo*— Neste trabalho, as clássicas redes perceptron multicamada (MLP) são comparadas com as recém-propostas redes Kolmogorov-Arnold (KAN) em um problema de regressão e outro de classificação binária. A KAN, baseada em *splines* ajustáveis, apresenta uma maior interpretabilidade às custas de uma complexidade computacional elevada e uma convergência mais lenta.**

***Palavras-Chave*— aprendizado de máquina, redes neurais, teorema de Kolmogorov-Arnold, *splines*, interpretabilidade.**

***Abstract*— In this work, classical multilayer perceptron (MLP) networks are compared with the newly proposed Kolmogorov-Arnold networks (KAN) in a regression problem and a binary classification problem. KAN, based on adjustable splines, offers greater interpretability at the cost of higher computational expense and slower convergence.**

***Keywords*— machine learning, neural networks, Kolmogorov-Arnold theorem, splines, interpretability.**

I. Introdução

Nos últimos anos, o aprendizado de máquina vem ganhando destaque no desenvolvimento tecnológico, influenciando como hardwares e softwares são projetados e mudando radicalmente a interação homem-máquina [1]–[3]. Em aprendizado profundo, a rede perceptron multicamada (*multilayer perceptron* – MLP) é protagonista na solução de problemas não lineares, respaldada pelo teorema da aproximação universal [1]. No entanto, a profundidade dessas redes dificulta o entendimento de como o problema é resolvido [1]–[3]. Em outras palavras, as redes MLP conseguem de fato resolver problemas complexos, mas as soluções em si são uma "caixa-preta", dificultando ou mesmo impedindo sua interpretabilidade.

A fim de possibilitar a interpretabilidade dos modelos, foi proposta recentemente a rede Kolmogorov-Arnold (*Kolmogorov-Arnold network* – KAN), inspirada pelo teorema de representação homônimo [4]. Enquanto as redes MLP possuem funções de ativação fixas nos neurônios, as KANs consideram as funções de ativação como parâmetros ajustáveis. Os pesos são substituídos por funções univariadas parametrizadas como *splines*. Essa mudança faz com que as KANs possam superar as MLPs em termos de precisão e interpretabilidade em alguns casos [4].

Neste trabalho, a KAN é comparada com a MLP em um problema de regressão e em outro de classificação binária. O artigo está organizado da seguinte forma. Na Seção II, a KAN é revisitada. Na Seção III, são mostrados os resultados da comparação com a MLP e na Seção IV, as conclusões.

Pedro H. S. Soares, Renato Candido e Magno T. M. Silva, Depto. de Engenharia de Sistemas Eletrônicos, Escola Politécnica, Universidade de São Paulo, São Paulo, SP, Brasil, emails: soares.pedro@usp.br; renatocan@lps.usp.br; magno.silva@usp.br. Este trabalho foi financiado pela CAPES (código de financiamento 001) e pelo CNPq (122941/2024-1, 303826/2022-3 e 404081/2023-1).

II. Revisitando a KAN

O teorema de representação de Kolmogorov-Arnold estabelece que dada uma função contínua multivariada $f : [0,1]^n \to \mathbb{R}$, então existe um conjunto de funções contínuas univariadas $\{\phi_{q,p}\}$ e $\{\Phi_q\}$ tais que f pode ser representada como [4], [5]

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right). \quad (1)$$

A motivação desse teorema é utilizar funções univariadas para reduzir a complexidade no cálculo de funções multivariadas.

A KAN se baseia nesse teorema, considerando funções *B-splines*. Essas funções, definidas em intervalos específicos denominados intervalos de nó, são construídas a partir da combinação linear de polinômios de grau baixo e utilizadas para aproximar curvas contínuas de maneira suave [6]. Assim, a combinação linear com coeficientes c_i das funções de base $B_{i,k}$ levam à *B-spline* de grau k, ou seja,

$$\phi_{q,p}(x_p) \approx \sum_{i=0}^{m} c_i B_{i,k}(x_p), \quad (2)$$

em que as funções de base

$$B_{i,k}(x_p) = \frac{x_p - t_i}{t_{i+k} - t_i} B_{i,k-1}(x_p) + \frac{t_{i+k+1} - x_p}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x_p),$$

são definidas com a fórmula de Cox-deBoor, t_i é o i-ésimo nó e $B_{\ell,0} = 1$ para $t_\ell \leq x \leq t_{\ell+1}$ e $B_{\ell,0} = 0$, caso contrário, $\ell = 0, \cdots, m$. Para $\ell > m$, assume-se $B_{\ell,k}(x_p) = 0$ [4], [6].

Na KAN, cada função de ativação $\phi_{q,p}(x_p)$ é aproximada por uma *B-spline* como em (2) e portanto, representada por um conjunto de $m + 1$ coeficientes c_i. No ajuste desses coeficientes, empregam-se algoritmos de treinamento similares aos da MLP, como o algoritmo *backpropagation* [1]. É importante garantir que as funções base sejam iguais para todas funções de ativação. Para isso, é necessário definir o número de intervalos de nós, denominado *grid*, e o grau das funções base como hiperparâmetros. Também é necessário verificar durante o treinamento o domínio das *B-splines* e atualizar os limites dos nós para que comporte os dados de treinamento [4].

A KAN se assemelha à MLP, pois ambas possuem uma estrutura de camadas conectadas, em que todos os neurônios de uma camada estão ligados aos neurônios das camadas subsequentes. A principal diferença está na forma como a ativação de cada neurônio é calculada: na MLP, a entrada é ponderada por pesos ajustáveis e a saída passa por uma função de ativação fixa, enquanto na KAN, cada componente da entrada é processado por uma função de ativação ajustável baseada em *B-splines*, e a saída é obtida por uma soma simples dessas transformações, conforme ilustrado na Figura 1.

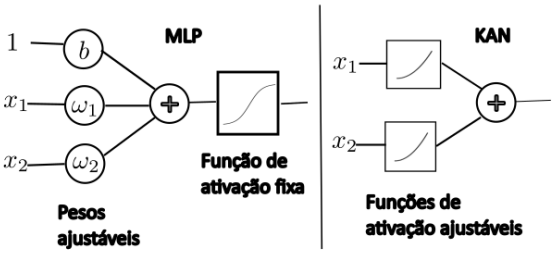

Fig. 1. Comparação entre as ativações de uma MLP e de uma KAN.

III. Resultados

Considerou-se inicialmente um problema de regressão com o objetivo de aproximar a função $f(x, y) = \sin^2(x + y)$, utilizando 4000 pontos (x, y), gerados aleatoriamente no domínio $[-\pi, \pi] \times [-\pi, \pi]$. A KAN e a MLP foram configuradas com três camadas, contendo 3, 2, 1 e 24, 25, 1 neurônios, respectivamente. Ambas foram treinadas por 3000 épocas com o otimizador Adam ($\beta_1 = 0.9$ e $\beta_2 = 0.999$) [7]. Na KAN, consideraram-se ainda $k = 3$ e o valor do *grid* iniciando com 3 e dobrando a cada 1000 épocas. Esse aumento, conhecido como extensão de *grid* [4], resultou no aumento do número de parâmetros treináveis a cada 1000 épocas. Na MLP, foram usadas ReLU nos neurônios das camadas ocultas e tangente hiperbólica para o neurônio de saída. Essas configurações permitem que o números de parâmetros treináveis de ambas as redes estejam próximos. Para comparação, também considerou-se um modelo da KAN com grid fixo igual a 12.

Em termos de custo computacional, na KAN com *grid* variável foram utilizados 462 parâmetros ao final de 130 s de treinamento, enquanto na MLP, foram treinados 463 parâmetros em 134 s. Apesar do mesmo custo neste caso, a raiz do erro quadrático médio (*root mean-square error* – RMSE) atingida pela KAN é de 0,0529 e o coeficiente de determinação (R^2) é de 0,9776, enquanto para a MLP essas métricas foram de 0,0228 e 0,9958, respectivamente. Na Figura 2-(a), é possível observar que a KAN apresentou uma convergência mais lenta que a MLP. Apesar do número de parâmetros finais da KAN com *grid* variável ser o mesmo da KAN com *grid* fixo, a extensão de *grid* ao longo do treinamento é essencial para se obter um melhor desempenho. No entanto, isso causa um aumento da função custo (pico da curva verde em torno da época 2000), mas que logo volta ao patamar de antes da alteração do *grid*. Esses resultados sugerem que a KAN apresenta resultados promissores em termos de desempenho para um custo computacional semelhante ao da MLP, embora a MLP ainda consiga um desempenho um pouco superior.

No problema de classificação binária, considerou-se o *German Credit Dataset* [8], que contém 1000 instâncias compostas por 20 variáveis. A variável-alvo assume valor $d = 1$ para clientes confiáveis e $d = -1$ para aqueles considerados de risco. Para se obter o mesmo custo computacional, a KAN e a MLP foram configuradas com 2 e 3 camadas, contendo $2, 1$ e $20, 13, 1$ neurônios, respectivamente. Consideraram-se para a KAN $k = 3$ e *grid* = 5, enquanto para a MLP, ReLU nos neurônios das camadas ocultas e tangente hiperbólica para o de saída. Ambas foram treinadas com otimizador Adam ($\beta_1 = 0.9$ e $\beta_2 = 0.999$) ao longo de 1000 épocas.

A KAN atingiu uma acurácia de 68% e a MLP de 77%. Novamente, a MLP convergiu mais rápido que a KAN, como pode ser observado na Figura 2-(b). Em termos de custo computacional, a MLP treinou 707 parâmetros por 17,3 s, enquanto a KAN treinou 714 parâmetros por 33,2 s. Apesar do desempenho inferior que o da MLP, simplificando a KAN com a técnica de poda ($\theta = 0{,}1$) como em [4] e utilizando o método dos mínimos quadrados não linear [9], foi possível extrair uma expressão de saída da KAN. As funções de ativação treinadas foram ajustadas considerando três funções candidatas: afim, cosseno e tangente hiperbólica. Para selecionar o melhor ajuste de cada função de ativação, foi utilizado o maior valor de coeficiente de determinação. Ao final do processo, a composição das funções resultou em

$$\begin{aligned}\hat{d} = &-0{,}025\cos\left(-0{,}032x_2 + 1{,}842\cdot 10^{-4}x_5 + 6{,}374\right)\\ &+ 0{,}210\tanh(2{,}151x_{13} + 1{,}235x_2 + 2{,}198x_4\\ &- 0{,}009x_5 + 45{,}244) - 0{,}116.\end{aligned}$$

Essa expressão é relativamente compacta e leva à mesma acurácia da KAN com os dados de teste. Ela possibilita verificar a relevância de cada componente da entrada (x_k, $k = 1, \cdots, 20$) na decisão de crédito, algo que não se consegue facilmente com a MLP. Isso faz da KAN um modelo promissor quando se deseja buscar interpretabilidade.

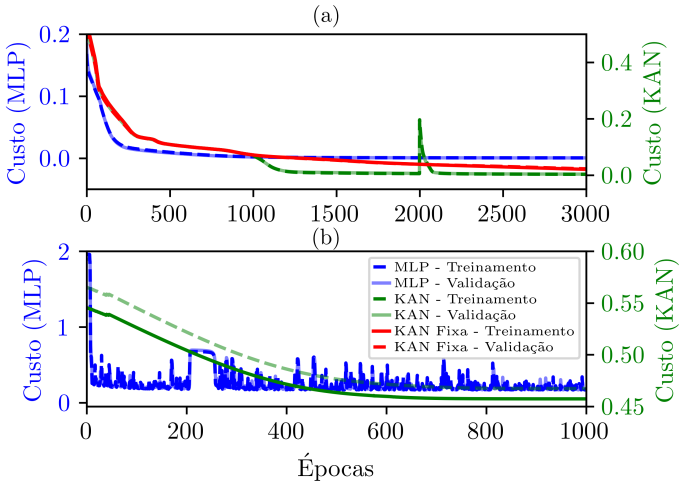

Fig. 2. Evolução do custo durante o treinamento para os problemas de (a) regressão e (b) classificação binária.

IV. Conclusões

Quando se busca interpretabilidade, a KAN é uma boa alternativa à rede MLP, apesar da convergência mais lenta no treinamento e do desempenho inferior em alguns casos.

Referências

[1] S. Haykin, *Neural networks and learning machines*, Prentice Hall, 2009.
[2] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016.
[3] C. M. Bishop, *Deep Learning*, Springer, 2024.
[4] Z. Liu *et al.*, "KAN: Kolmogorov-Arnold networks," *arXiv*, https://arxiv.org/abs/2404.19756, 2024.
[5] A. N. Kolmogorov, "On the representation of continuous functions of several variables as superpositions of continuous functions of a smaller number of variables," *Dokl. Akad. Nauk*, 1956.
[6] L. Schumaker, *Spline Functions: Computational Methods*, SIAM, 2015.
[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv*, https://arxiv.org/abs/1412.6980, 2014.
[8] H. Hofmann, "Statlog (German Credit Data)," UCI Machine Learning Repository, 1994, DOI: https://doi.org/10.24432/C5NC77.
[9] D. M. Bates and D. G. Watts, *Nonlinear Regression Analysis and Its Applications*, John Wiley & Sons, 1988.