UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE DISCIPLINA REDES NEURAIS - DEEP LEARNING Iª LISTA DE EXERCÍCIO - 2024.1

Aluno: Vinícius Lamas von Sohsten
Matrícula: 20231011310
Professor: Adrião Duarte Doria Neto

1. Métricas de distância

A representação do conhecimento em uma rede neural é definida pelos valores dos pesos sinápticos da rede, sendo a representação do conhecimento a chave para seu desempenho. Apesar deste tema ser complicado, existem quatro regras para representação do conhecimento. A primeira regra diz respeito à similaridade entre entrada e representação da rede, para que assim uma informação possa ser classificada. Existem diversas medidas para determinar a similaridade das informações, a seguir, apresenta-se um estudo sobre algumas dessas métricas.

I. **Distância Euclidiana**: A distância Euclidiana entre dois vetores $\mathbf{x}_i$ e $\mathbf{x}_j$ em um espaço Euclidiano é a distância do segmento de reta que os une, definida pela seguinte fórmula:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| = \left[\sum_{k=1}^{m} (x_{ik} - x_{jk})^2\right]^{\frac{1}{2}} \tag{1}$$

onde m é a dimensão do espaço euclidiano $\mathbb{R}^m$, e x_{ik} e x_{jk} são os k-ésimos elementos dos vetores de entrada.

Essa métrica é adequada para problemas de classificação de padrão, onde a forma ou magnitude das diferenças entre padrões são importantes, por exemplo, reconhecimento de dígitos. Também pode ser utilizada em algoritmos de clustering como K-means.

II. **Distânca de Minkowski**: É uma generalização da distância Euclidiana que permite ajustar o parâmetro de ordem da fórmula, como mostra-se a seguir:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left[\sum_{k=1}^{m} |x_{ik} - x_{jk}|^p\right]^{\frac{1}{p}} \tag{2}$$

onde p, o parâmetro a ser ajustado, é um inteiro e $p \geqslant 1$, pois quando $p < 1$ a métrica viola a desigualdade triangular tornando a métrica inválida. Quando p assume o valor de 1 ou 2, a métrica corresponde à distância de Manhattan e a distância Euclidiana, respectivamente.

Essa métrica é bem semelhante ao caso da distância Euclidiana, sendo útil para classificar padrões, mas permitindo ajustar a sensibilidade às diferenças de magnitude das entradas. Também utiliza-se em clustering quando se deseja dar mais peso a certas características a outras.

III. **Distância City Block**: Também conhecida como distância de Manhattan, é a soma das diferenças absolutas das coodernadas entre dois pontos, um caso particular da distância de Minkowski, apresenta-se da seguinte maneira:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^{m} |x_{ik} - x_{jk}| \tag{3}$$

Aplicada em casos de classificação de padrões quando as característica têm diferentes escalas, pois a distância não é afetada por transformações lineares. Também em caso de clustering para agrupamento de dados esparsos em alta dimensão.

IV. **Distância de Mahalanobis**: Essa medida leva em consideração a covariância e os valores médios entre duas populações de dados. Descrita pela seguinte fórmula:

$$d_{ij}^2 = (\mathbf{x}_i - \boldsymbol{\mu}_i)\boldsymbol{\Sigma}^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_j) \tag{4}$$

onde Σ^{-1} é a inversa da matriz de covariância Σ, e $\boldsymbol{\mu}_i$ e $\boldsymbol{\mu}_j$ representam os valores médios do vetores $\mathbf{x}_i$ e $\mathbf{x}_j$, respectivamente, de tal maneira:

$$\boldsymbol{\mu}_i = E[\mathbf{x}_i] \tag{5}$$

onde E é o operador estatístico esperado.

Essa métrica se faz útil em problemas de classificação de padrões onde as variâncias das diferentes dimensões dos dados são importantes e não são iguais. Também pode ser aplicada em algoritmos de clustering como DBSCAN.

V. **Coeficiente de Correlação de Pearson**: Este coeficiente mede o grau de correlação linear entre duas variáveis, sendo denotado como ρ quando utilizado como parâmetro populacional e representado por r para uma estatística amostral. Se faz útil quando ambas as variáveis em estudo têm distribuição normal, pois caso não possírem tal distribuição o coeficiente pode ser afetado por valores extremos tornando a análise pelo coeficiente inadequada. O valor deste coeficiente assume apenas valores entre -1 e 1, para valores positivos tem-se a correlação positiva e o mesmo vale para valores negativos. Porém, no caso em que o coeficiente assume valor zero significa que o resultado é inconclusivo e deve-se procurar outros meios de investigar a correlação, pois pode existir uma dependência não linear. O coeficiente pode assumir as seguintes fórmulas:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{[\sum_{i=1}^{n}(x_i - \bar{x})^2][\sum_{i=1}^{n}(y_i - \bar{y})^2]}} \tag{6}$$

$$\boldsymbol{\rho}_{ab} = \frac{E[(Y_a - E[Y_a])(Y_b - E[Y_b])]}{\boldsymbol{\sigma}_a \boldsymbol{\sigma}_b} \tag{7}$$

onde x_i e y_i são os valores que estão contidos nos vetores $\mathbf{x}$ e $\mathbf{y}$, respctivamente, que se deseja medir a correlação, os valores médios dos vetores são representados por $\bar{x}$ e $\bar{y}$, respectivamente. Para a outra equação, vale salientar que o numerador representa a covariância dos dados e no denominador se encontram os desvios padrões dos respectivos dados representados pelo símbolo $\boldsymbol{\sigma}$, cujo o valor é a raíz quadrada da variância.

Esta métrica se mostra eficiente em casos de reconhecimento de padrões para encontrar a similaridade entre dois conjuntos de dados multidimensionais. Pelo mesmo motivo, para análise de dados ajuda a identificar padrões de correlação entre diferentes variáveis.

VI. **Similaridade Cosseno**: Esse é um caso simples utilizado para medir o cosseno do ângulo entre dois vetores no espaço, fazendo o produto interno dos vetores e dividindo pela norma dos vetores, da seguinte maneira:

$$\cos(\mathbf{x_i}, \mathbf{x_j}) = \frac{\mathbf{x}_i^T \mathbf{x}_j}{||\mathbf{x}_i||\,||\mathbf{x}_j||} \tag{8}$$

Útil para calcular a similaridade entre vetores de características em sistemas de recomendação e em sistemas de busca para medir a similaridades entre documentos e consultas de pesquisa.

2. Demonstrações com base na expansão em série de Taylor

a) Método do Gradiente da Descida mais Íngrime

Partindo da expressão em série de Taylor da função custo:

$$\mathscr{E}(\mathbf{w}(n) + \Delta\mathbf{w}(n)) = \mathscr{E}(\mathbf{w}(n)) + \mathbf{g}^t(\mathbf{w}(n))\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^t(n)\mathbf{H}(\mathbf{w}(n))\Delta\mathbf{w}(n) + O(||\Delta\mathbf{w}||^3) \tag{9}$$

Nesse método é realizado o truncamento da expansão de Taylor na primeira ordem em torno de $\mathbf{w}(n)$ para aproximar o valor da função custo. Dessa forma os temos de segunda ordem e maiores são desprezados, obtém-se:

$$\mathbf{w}(n) + \Delta\mathbf{w}(n) = \mathbf{w}(n+1) \tag{10}$$

$$\mathscr{E}(\mathbf{w}(n+1)) \simeq \mathscr{E}(\mathbf{w}(n)) + \mathbf{g}^t(\mathbf{w}(n))\Delta\mathbf{w}(n) \tag{11}$$

O algoritmo da descida mais íngrime é descrito formalmente por:

$$\Delta \mathbf{w}(n) = -\eta g(\mathbf{w}(n)) \Rightarrow \mathbf{w}(n+1) = \mathbf{w}(n) - \eta \mathbf{g}(\mathbf{w}(n)) \tag{12}$$

onde η é uma constante positiva chamada taxa de aprendizagem. Pode-se então substituir Eq. 12 na Eq. 11 e utilizar a Eq. 10 para obter:

$$\mathscr{E}(\mathbf{w}(n+1)) \leqslant \mathscr{E}(\mathbf{w}(n)) \Rightarrow \mathscr{E}(\mathbf{w}(n+1)) \simeq \mathscr{E}(\mathbf{w}(n)) - \eta \|\mathbf{g}(n)\|^2 \tag{13}$$

b) **Método de Newton**

Diferentemente do método do gradiente, o método de Newton faz uso da expansão de Taylor de segunda ordem da função de custo em torno do ponto $\mathbf{w}(n)$. Desta forma, obtém-se:

$$\mathscr{E}(\mathbf{w}(n+1)) - \mathscr{E}(\mathbf{w}(n)) = \Delta\mathscr{E}(\mathbf{w}(n)) \simeq \mathbf{g}^t(\mathbf{w}(n))\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^t(n)\mathbf{H}(\mathbf{w}(n))\Delta\mathbf{w}(n) \tag{14}$$

Para minimizar a variação da função custo $\Delta\mathscr{E}(\mathbf{w})$ deve-se diferenciar a Eq. 14 em relação a $\Delta\mathbf{w}$. Obtém-se, então:

$$\mathbf{g}(\mathbf{w}(n)) + \mathbf{H}(\mathbf{w}(n))\Delta\mathbf{w}(n) = \mathbf{0} \tag{15}$$

Agora, basta isolar $\Delta\mathbf{w}(n)$ e utilizar a relação da Eq. 10 para obter:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mathbf{H}^{-1}(\mathbf{w}(n))\mathbf{g}(\mathbf{w}(n)) \tag{16}$$

c) **Passo ótimo**

Pode-se utilizar a definição da Eq. 12 e substituir na Eq. 14, obtendo-se:

$$\mathscr{E}(\mathbf{w} - \eta\mathbf{g}(\mathbf{w})) \simeq \mathscr{E}(\mathbf{w}) - \eta\mathbf{g}^t(\mathbf{w})\mathbf{g}(\mathbf{w}) + \frac{1}{2}\eta^2\mathbf{g}^t(\mathbf{w})\mathbf{H}(\mathbf{w})\mathbf{g}(\mathbf{w}) \tag{17}$$

O passo ótimo ocorre quando:

$$\frac{d}{d\eta}(\mathscr{E}(\mathbf{w} - \eta\mathbf{g}(\mathbf{w})) = 0 \tag{18}$$

Portanto, ao derivar Eq. 17 em relação a η, obtém-se a relação do passo ótimo ou taxa de aprendizagem ótima, para uma $\mathbf{H}$ positiva, da seguinte maneira:

$$\eta^* = \frac{\mathbf{g}(\mathbf{w})\mathbf{g}^t(\mathbf{w})}{\mathbf{g}^t(\mathbf{w})\mathbf{H}(\mathbf{w})\mathbf{g}(\mathbf{w})} \tag{19}$$

d) **Valor do passo se o vetor gradiente estiver alinhado com o autovetor correpondeste ao autovalor máximo**

Se o vetor gradiente $\mathbf{g}(\mathbf{w})$ estiver alinhado com o autovetor de $\mathbf{H}(\mathbf{w})$ correspondente ao autovalor máximo λ_{max}, tem-se as seguintes condições:

$$\mathbf{g}(\mathbf{w}) = \alpha\mathbf{v}_{max} \tag{20}$$

$$\mathbf{H}(\mathbf{w})\mathbf{v}_{max} = \lambda_{max}\mathbf{v}_{max} \tag{21}$$

onde α é um escalar constante e $\mathbf{v}_{max}$ é o autovetor máximo da relação. Com isso, é possível obter o passo sob tais condições substituindo as Eq. 20 e Eq. 21 na Eq. 19, em que obtém-se:

$$\eta^* = \frac{\alpha^2\mathbf{v}_{max}^2}{\alpha^2\mathbf{v}_{max}\mathbf{H}\mathbf{v}_{max}} \tag{22}$$

$$\eta^* = \frac{\alpha^2\mathbf{v}_{max}^2}{\alpha^2\lambda_{max}\mathbf{v}_{max}^2} \tag{23}$$

Portanto, para esse cenário, o passo ótimo é:

$$\eta^* = \frac{1}{\lambda_{max}} \tag{24}$$

3. Matriz Hessiana

a) **Demonstração**

Essa demonstração pode ser realizada como mostrado nos seguintes passos:

$$\mathbf{H}(n-1) = \sum_{k=1}^{n-1} \mathbf{g}(k)\mathbf{g}^t(k) \tag{25}$$

$$\mathbf{H}(n) = \mathbf{g}(n)\mathbf{g}^t(n) + \sum_{k=1}^{n-1} \mathbf{g}(k)\mathbf{g}^t(k) \tag{26}$$

Substituindo Eq. 25 na Eq. 26, obtém-se o resultado desejado:

$$\mathbf{H}(n) = \mathbf{H}(n-1) + \mathbf{g}(n)\mathbf{g}^t(n) \tag{27}$$

b) **Lema da inversão matricial**

Partindo do resultado do lema da inversão matricial:

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{BC}(\mathbf{D} + \mathbf{C}^t\mathbf{BC})^{-1}\mathbf{C}^t\mathbf{B} \tag{28}$$

Para obter-se a forma recursiva da inversa da hessiana, pode-se utilizar o resultado do lema da inversão matricial utilizando as seguintes relações:

$$\mathbf{A} = \mathbf{H}(n) \tag{29}$$

$$\mathbf{B}^{-1} = \mathbf{H}(n-1) \tag{30}$$

$$\mathbf{C} = \mathbf{g}(n) \tag{31}$$

$$\mathbf{D} = 1 \tag{32}$$

Portando, obtém-se nos seguintes passos, a demonstração desejada:

$$\mathbf{H}^{-1}(n) = \mathbf{H}^{-1}(n-1) - \mathbf{H}^{-1}(n-1)\mathbf{g}(n)(1 + \mathbf{g}(n)\mathbf{H}^{-1}(n-1)\mathbf{g}(n))^{-1}\mathbf{g}^t(n)\mathbf{H}^{-1}(n-1) \tag{33}$$

$$\mathbf{H}^{-1}(n) = \mathbf{H}^{-1}(n-1) - \frac{\mathbf{H}^{-1}(n-1)\mathbf{g}(n)\mathbf{g}^t(n)\mathbf{H}^{-1}(n-1)}{1 + \mathbf{g}(n)\mathbf{H}^{-1}(n-1)\mathbf{g}(n)} \tag{34}$$

4. Função de múltiplas variáveis

a) **Representação na forma matricial**

A função dada é:

$$E(w_1, w_2, w_3) = w_1^2 - w_2^2 + 2w_3^2 - 0.1w_1w_2 + 0.5w_1w_3 + 0.08w_2w_3 - 0.2w_1 + 0.3w_2 - 0.7w_3 + 0.4 \tag{35}$$

A forma matricial é dada por:

$$E(\mathbf{w}) = \mathbf{w}^t\mathbf{A}\mathbf{w} + \mathbf{b}^t\mathbf{w} + a \tag{36}$$

onde $\mathbf{w} = [\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3]^t$, onde $a \in \mathbf{R}$, $\mathbf{b} \in \mathbf{R}^3$ e $\mathbf{A} \in \mathbf{R}^{3x3}$.

Além disso, para resolver esse problema, pode-se representar a matriz $\mathbf{A}$ e o vetor $\mathbf{b}$ das seguintes maneiras:

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \tag{37}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{38}$$

Realizando a substituição das Eq. 37 e Eq. 38 na Eq. 36 e realizando as devidas manipulações algébricas, obtém-se a seguinte igualdade polinomial:

$$E = A_{11}w_1^2 + A_{22}w_2^2 + A_{33}w_3^2 + (A_{12}+A_{21})w_1w_2 + (A_{13}+A_{31})w_1w_3 + (A_{23}+A_{32})w_2w_3 + b_1w_1 + b_2w_2 + b_3w_3 + a$$

(39)

Dessa maneira, pode-se resolver e obter:

$$\mathbf{A} = \begin{bmatrix} 1 & -0.05 & 0.25 \\ -0.05 & -1 & 0.04 \\ 0.25 & 0.04 & 2 \end{bmatrix} \tag{40}$$

$$\mathbf{b} = \begin{bmatrix} -0.2 \\ 0.3 \\ -0.7 \end{bmatrix} \tag{41}$$

$$a = 0.4 \tag{42}$$

Com isso, foi obtido a forma matricial da função.:

$$E(\mathbf{w}) = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} 1 & -0.05 & 0.25 \\ -0.05 & -1 & 0.04 \\ 0.25 & 0.04 & 2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + \begin{bmatrix} -0.2 & 0.3 & -0.7 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + a \tag{43}$$

b) **Determinar o gradiente**

Para determinar o gradiente da função dada, deve-se derivá-la parcialmente em relação a $\mathbf{w}$:

$$\mathbf{g}(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = 2\mathbf{A}\mathbf{w} + \mathbf{b}^t \tag{44}$$

$$\mathbf{g}(\mathbf{w}) = 2 \begin{bmatrix} 1 & -0.05 & 0.25 \\ -0.05 & -1 & 0.04 \\ 0.25 & 0.04 & 2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} + \begin{bmatrix} -0.2 & 0.3 & -0.7 \end{bmatrix} \tag{45}$$

$$\mathbf{g}(\mathbf{w}) = \begin{bmatrix} 2.3w_1 - 0.2 & 1.29w_2 + 0.3 & 2.29w_3 - 0.7 \end{bmatrix} \tag{46}$$

c) **Determinar matriz Hessiana**

Para determinar a matriz Hessiana da função dada é necessário derivar o seu gradiente parcialmente em relação a $\mathbf{w}$, ou seja, derivar duas vezes parcialmente a função custo em relação a $\mathbf{w}$. Obtém-se:

$$\mathbf{H}(\mathbf{w}) = \frac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}^2} = 2\mathbf{A} = \begin{bmatrix} 2 & -0.1 & 0.5 \\ -0.1 & -2 & 0.08 \\ 0.5 & 0.08 & 4 \end{bmatrix} \tag{47}$$

d) **Verifcar se a função é estritamente convexa**

Uma função estritamente convexa é aquela em que sua matriz Hessiana é definida positiva, isto é, todos os autovalores da matriz são positivos. Para descobrir os autovalores de forma rápida e eficiente, foi utilizado o módulo "linalg" da biblioteca Numpy em um código Python, obtém-se:

$$\boldsymbol{\lambda} = \begin{bmatrix} 4.12 & 1.88 & -2.00 \end{bmatrix} \tag{48}$$

Portanto, de acordo com a definição, a função não é estritamente convexa.

e) **Teste da segunda derivada para múltiplas dimensões**

Em um ponto crítico, isto é, em que $\mathbf{g}(\mathbf{w}) = \mathbf{0}$, pode-se examinar os autovalores da matriz Hessiana para determinar a condição desse ponto, da seguinte maneira:

I. Para o ponto ser um mínimo local, a matriz Hessiana deve ser definida positiva, todos seus autovalores devem ser positivos;

II. para o ponto ser um máximo local, a matriz Hessiana deve ser definida negativa, todos seus autovalores devem ser negativos;

III. para o caso de ser um ponto de sela, é necessário que ao menos um autovalor seja positivo e ao menos um autovalor seja negativo.

5. Estudo dos algoritmos de otimização

I. Gradiente Estocástico Descendente

O Gradiente Estocástico Descendente (Stochastic Gradient Descent - SGD) é um algoritmo de otimização que se baseia no método da descida mais íngreme, porém calcula o gradiente usando apenas um pequeno conjunto de exemplos escolhidos aleatoriamente chamado *minibatch*. É um algoritmo muito eficiente em termos computacionais, principalmente para modelos de grande conjuntos de dados, no entando o método de estimar o gradiente introduz fontes de ruídos que implica em menores taxas de convergência, ou seja, um aprendizado mais lento.

II. AdaGrad

Algoritmo do Gradiente Adaptativo (Adaptative Gradient Algorithm - AdaGrad) é um algoritmo de otimização que individualmente adapta a taxa de aprendizagem para todos os parâmetros do modelo dividindo a taxa pela raiz da soma acumulada dos quadrados dos gradientes anteriores para cada parâmetro. Isso faz com que parâmetros com maiores derivadas parciais do custo tenham sua taxa de apredizagem aumentada, enquanto parâmetros com menores derivadas parciais tenham sua taxa de aprendizagem diminuida relativamente. Isso torna o algoritmo eficaz em problemas em que gradientes possuem escalas muito diferentes e suaviza a convergência. Entretanto, na prática, acumular o quadrado dos gradientes pode resultar em diminuição excessiva da taxa de aprendizagem, fazendo o algoritmo performar bem alguns, mas não todos modelos de apredizagem de máquina.

III. RMSProp

Propagação da Raiz Quadrática Média (Root Mean Square Propagation - RMSProp) é um algoritmo que modifica o AdaGrad para performar melhor em situações não convexas, resolvendo o problema da diminuição muito rápida da taxa de aprendizado ao longo do tempo. RMSProp faz isso utilizando uma média móvel exponencialmente ponderada dos gradientes anteriores, basicamente utiliza uma média decaindo exponencialmente para tirar a importância de valores passados extremos para que possa convergir mais rapidamente. Isto é possível, pois o algoritmo faz uso de um novo hiperparâmetro que controla o tamanho da escala da média móvel. Esse algortimo mostra-se eficaz e prático para otimização de redes neurais profundas.

IV. Adam

Estimativa de Momento Adptativa (Adaptive Momento Estimation - Adam) é um algoritmo de otimização estocástico adptativo que combina as ideias do SGD com momento (variação do SGD que inclui um parâmetro para acelerar a convergência) e RMSProp. Adam faz uso de dois momentos, um linear e um quadrático, para atualizar a taxa de aprendizagem com base numa relação desses momentos com suas respectivas tavas de decaimento exponencial, portanto esse algoritmo possui dois hiperparâmetros.

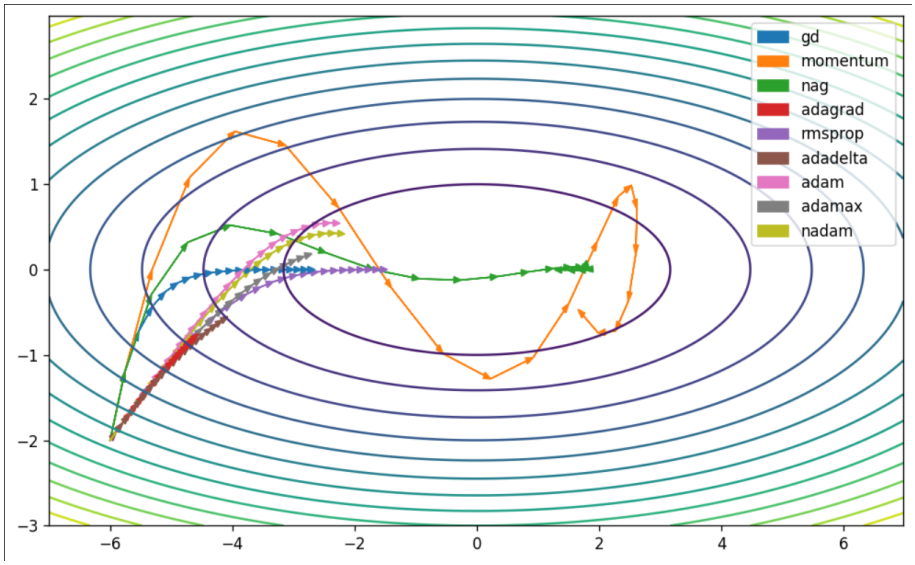


Figure 1. Comparativo dos algoritmos em um problema 2D.

6. Funções de ativação

I. Análise comparativa das funções de ativação e suas respectivas derivadas

Cada uma dessas funções tem suas próprias características e vantagens, que podem influenciar significativamente o desempenho e o comportamento da rede neural. A seguir é realizada uma análise comparativa dos gráficos dessas funções, bem como de suas respectivas derivadas.

- **Função Sigmoíde**
  A função sigmoide possui uma forma de "S" suave e contínua, como mostrado na Fig. 2. Ela mapeia qualquer valor de v para o intervalo (0, 1), o que a torna adequada para problemas de classificação binária, onde a saída desejada está entre 0 e 1. A derivada da função sigmoide atinge valores máximos em torno de $v = 0$ e diminui para zero em direção aos extremos do intervalo. Isso significa que a taxa de mudança é alta em torno do ponto médio do intervalo e diminui à medida que nos aproximamos de 0 ou 1. Isso pode causar o problema de desaparecimento do gradiente, especialmente em redes profundas.
- **Função Tangsigmoíde**
  A função tangente hiperbólica é semelhante à sigmoide, mas sua faixa de saída está entre -1 e 1, como mostrado na Fig. 3. Ela é simétrica em torno de $v = 0$ e possui uma forma de "S" suave. Ela é útil em problemas de classificação e regressão. A derivada da tangente hiperbólica também atinge seus valores máximos em torno de $v = 0$, mas é mais pronunciada do que a derivada da sigmoide. Isso pode ajudar a mitigar o problema do desaparecimento do gradiente em comparação com a função sigmoide.
- **Função ReLu**
  A função ReLU é linear e não linear apenas em um lado, onde retorna 0 para valores de v negativos e v para valores positivos, como mostrado na Fig. 4. Ela é simples e computacionalmente eficiente, e tem sido amplamente utilizada em redes neurais profundas. A derivada da ReLU é simples: ela retorna 0 para valores negativos de v e 1 para valores positivos de v. Isso significa que não há desaparecimento do gradiente para valores positivos de v, tornando a ReLU uma escolha popular para redes profundas.

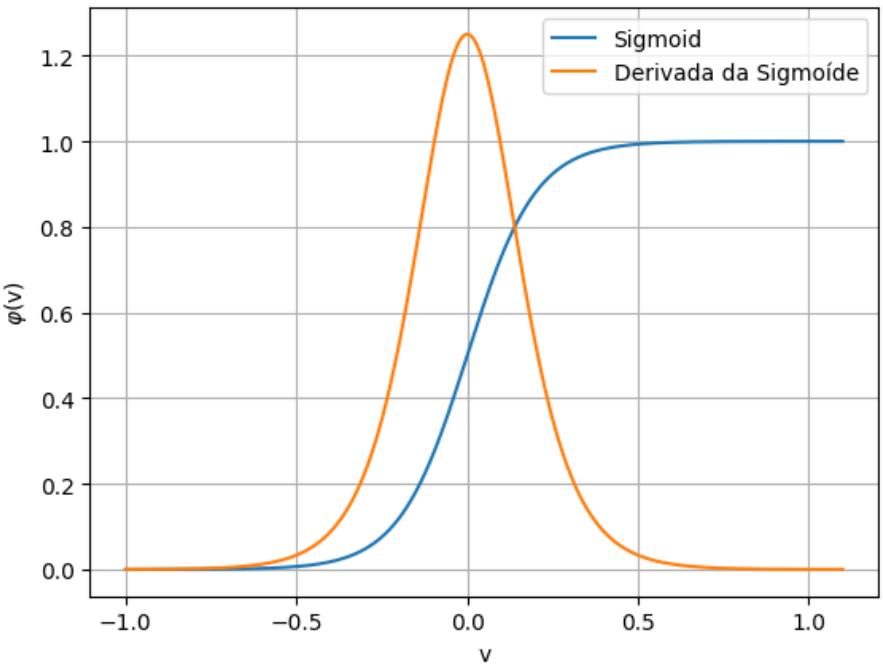


Figure 2. Função de Ativação Sigmoíde e sua Derivada

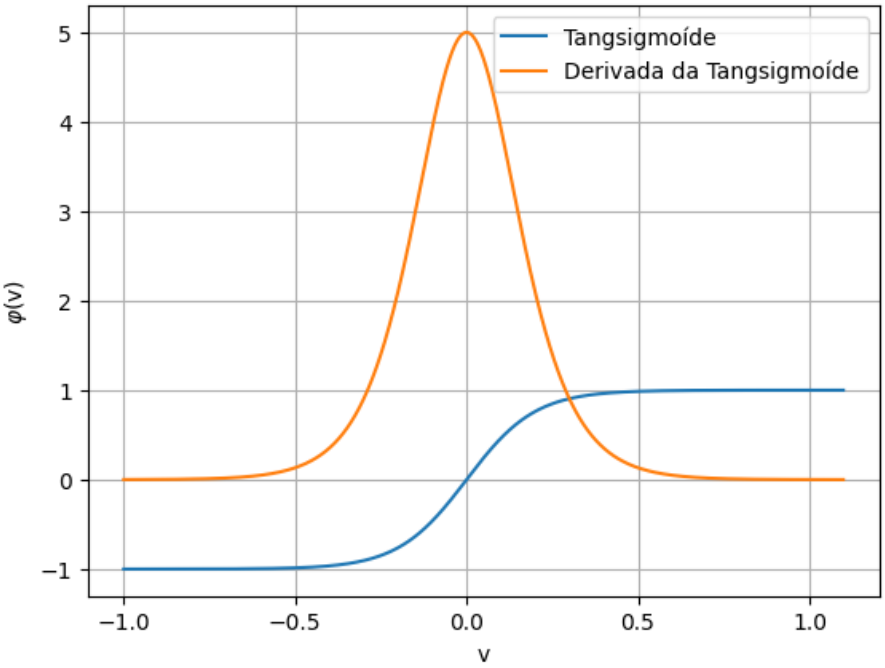


Figure 3. Função de Ativação Tangsigmoíde e sua Derivada

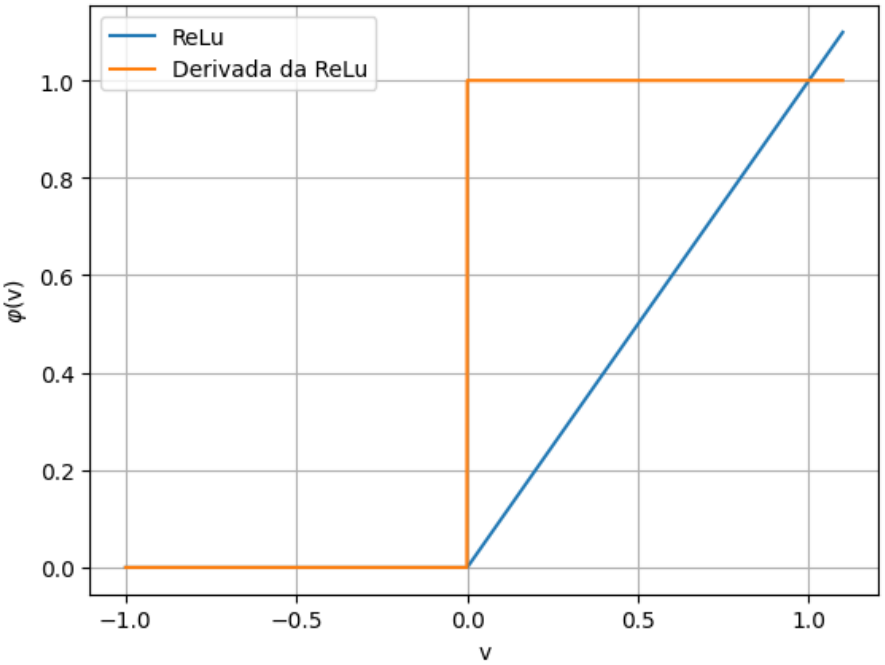


Figure 4. Função de Ativação ReLu e sua Derivada

II. **Derivada função sigmoíde**

$$\varphi(v) = \frac{1}{1+e^{-av}} \quad \Rightarrow \quad \frac{d\varphi(v)}{dv} = \frac{\frac{d1}{dv}\cdot(1+e^{-av}) - \frac{d(1+e^{-av})}{dv}\cdot 1}{(1+e^{-av})^2} = \frac{ae^{-av}}{(1+e^{-av})^2} \tag{49}$$

$$\Rightarrow \frac{d\varphi(v)}{dv} = \frac{a}{1+e^{-av}}\left(\frac{1+e^{-av}-1}{1+e^{-av}}\right) = \frac{a}{1+e^{-av}}\left(1-\frac{1}{1+e^{-av}}\right) = a\varphi(v)[1-\varphi(v)] \tag{50}$$

III. **Derivada função tangsigmoíde**

$$\varphi(v) = \frac{1-e^{-av}}{1+e^{-av}} \tag{51}$$

$$\Rightarrow \frac{d\varphi(v)}{dv} = \frac{\frac{d(1-e^{-av})}{dv}\cdot(1+e^{-av}) - \frac{d(1+e^{-av})}{dv}\cdot(1-e^{-av})}{(1+e^{-av})^2} = \frac{ae^{-av}(1+e^{-av}) + ae^{-av}(1-e^{-av})}{(1+e^{-av})^2} \tag{52}$$

$$\Rightarrow \frac{a(e^{-av}+e^{-2av}+e^{-av}-e^{-2av})}{(1+e^{-av})^2} = \frac{a(e^{-av}+e^{-2av}+e^{-av}-e^{-2av})}{(1+e^{-av})^2} \tag{53}$$

$$\Rightarrow \frac{a[(1+2e^{-av}+e^{-2av})-(1-2e^{-av}+e^{-2av})]}{2(1+e^{-av})^2} = \frac{a[(1+e^{-av})^2-(1-e^{-av})^2]}{2(1+e^{-av})^2} \tag{54}$$

$$\frac{d\varphi(v)}{dv} = \frac{a}{2}\left[1-\frac{(1-e^{-av})^2}{(1+e^{-av})^2}\right] = \frac{a}{2}[1-\varphi^2(v)] \tag{55}$$

7. Funções de saída

I. **Classificação de padrões com duas classes**

Para problemas de classificação binária a função de ativação mais comumente utilizada na camada de saída é a **função sigmoíde** (ou logística). Esta unção tem a propriedade de mapear os valores de entrada para o intervalo [0,1], o que é útil para representar a probabilidade de pertencer a uma das duas classes. função é dada pela seguinte fórmula:

$$\mathtt{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{56}$$

II. **Classificação de padrões com múltiplas classes**

Em casos de classficação com mais de duas classes, o mais comum é utilizar a **função softmax**. A função softmax é uma generalização da função sigmoíde para múltpilas classes. Ela transforma os valores de entrada em uma distribuição de probabilidade sobre as clases, de modo que a soma das saídas seja igual a 1. A função é dada pela seguinte fórmula:

$$\mathtt{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{57}$$

III. **Problema de regressão (aproximação de funções)**

Para problemas de regressão, onde o objetivo é prever um valor contínuo, a função de ativação mais comumente usada na camada de saída é a **função linear**. Esta função simplesmente passa os valores de entrada sem alterá-los. Ela é dada pela fórmula:

$$f(x) = x \tag{58}$$

8. Método Max-Likelihood em problema de distribuição gaussiana multivariada

Para modelar um conjunto de dados com várias variáveis, pode-se utilizar a distribuição gaussiana multivariada. Esta distribuição é uma generalização da distribuição gaussiana univariada para múltiplas dimensões. Ela é caracterizada por dois parâmetros principais: o vetor de médias μ e a matriz de covariância Σ. O vetor de médias μ especifica o centro da distribuição, enquanto a matriz de covariância Σ controla a forma e a orientação da distribuição em torno de μ. Para determinar os melhores valores desse dois parâmetros para representar o conjunto de dados se usa o método da máxima verossimilhança (Max-Likelihood). Para isso, utiliza-se a função de densidade de probabilidade da distribuição gaussian multivariada, definada da seguinte maneira:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-\frac{D}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})) \tag{59}$$

onde $\mathbf{x} \in \mathbb{R}^D$. Para simplificar futuras equações, escreve-se $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

Para formular o aprendizado de parâmetro via máxima verossimilhança, assume-se um dataset $\mathcal{X} = \{\mathbf{x_1}, ..., \mathbf{x}_N\}$, onde $\mathbf{x}_n$, $n = 1, ..., N$ possuem uma distribuição $p(\mathbf{x})$. O objetivo é encontrar uma boa aproximação da distribuição a partir de uma quantidade K de componentes da mistura das distribuições. Sendo esses parâmetros as médias $\boldsymbol{\mu}_k$, a covariância $\boldsymbol{\Sigma}_k$ e o peso de cada distribuição na mistura π_k. Pode-se compactar todos esses parâmetros em $\theta = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k : k = 1, ..., K\}$. A função de verossimilhança é o produto das densidades de probabilidade de cada exemplo nos dados de treinamento, dada a distribuiçõ gaussiana multivariada:

$$p(\mathcal{X}|\theta) = \prod_{n=1}^{N} p(\mathbf{x_n}|\theta), \quad p(\mathbf{x}_n|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{60}$$

Para encontrar os parâmetros θ^*_{ML} que maximizam a função de verossimilhança, pode-se maximizar o logaritmo da função de verossimilhança (pois o logaritmo é uma função monótona crescente). Isso simplifica o cálculo e não afeta a localização dos máximos. Então, calcula-se:

$$\log p(\mathcal{X}|\theta) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{61}$$

Está sendo considerado o caso genérico de densidade desejada, porém se considerar uma única distribuição gaussiana como densidade desejada, a soma de k desaparece, e o log pode ser aplicado diretamente, resultando:

$$\log \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{D}{2} \log(2\pi) - \frac{1}{2} \log |\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mu) \tag{62}$$

Para maximizar a função log-verossimilhança, tomamos as derivadas parciais em relação a $\boldsymbol{\mu}$ e $\boldsymbol{\Sigma}$, e igualamos a zero para encontrar os valores ótimos. Para $\boldsymbol{\mu}$:

$$\frac{\partial}{\partial \mu} \log p(\mathcal{X}|\mu, \boldsymbol{\Sigma}) = 0 \Rightarrow \sum_{n=1}^{N} \frac{\partial}{\partial \mu} \left[-\frac{D}{2} \log(2\pi) - \frac{1}{2} \log |\boldsymbol{\Sigma}| - \frac{1}{2}(\mathbf{x}_n - \mu)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \mu) \right] = 0 \tag{63}$$

$$\Rightarrow \sum_{n=1}^{N} \boldsymbol{\Sigma}^{-1}(\mathbf{x}_n - \mu) = 0 \Rightarrow \sum_{n=1}^{N} (\mathbf{x}_n - \mu) = 0 \tag{64}$$

$$\Rightarrow \hat{\mu} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n = \bar{\mathbf{x}} \tag{65}$$

Para $\boldsymbol{\Sigma}$:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}^{-1}} \log p(\mathcal{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0 \Rightarrow \sum_{n=1}^{N} \frac{\partial}{\partial \boldsymbol{\Sigma}^{-1}} \left[-\frac{D}{2} \log(2\pi) + \frac{1}{2} \log |\boldsymbol{\Sigma}^{-1}| - \frac{1}{2} tr[(\mathbf{x}_n - \mu)(\mathbf{x}_n - \mu)^T \boldsymbol{\Sigma}^{-1}] \right] = 0 \tag{66}$$

$$\Rightarrow \frac{N}{2}\Sigma - \frac{1}{2} \sum_{n=1}^{N} (\mathbf{x}_n - \mu)(\mathbf{x}_n - \mu)^T = 0 \tag{67}$$

$$\Rightarrow \hat{\Sigma} = \frac{1}{N} \sum_{n=1}^{m} (\mathbf{x}_n - \hat{\mu})(\mathbf{x}_n - \hat{\mu})^T \tag{68}$$

9. Implementações Computacionais de Redes Neurais

9.1 MLP para função multivariada

Para resolver esse problema, foi contruído um código em Python utilizando a biblioteca TensolFlow para realizar os cálculos da arquitetura definida de uma rede perceptron de múltiplas camadas. Este é um problema de regressão em que há duas variáveis de entrada e uma de saída, dada pela seguinte função:

$$f(\mathbf{x}) = 16x_1^2 + x_1x_2 + 8x_2^2 - x_1 - x_2 + \ln(1 + x_1^2 + x_2^2) \tag{69}$$

A arquitetura escolhida, após alguns testes, foi de 2:10:10:1, ou seja, duas camadas ocultas de 10 neurônios cada para resolver o problema. O conjunto de dados é separado em 80% para treinamento e 20% para validação. As camadas ocultas utilizam a função de ativação ReLu, por se tratar de um problema de regressão, e a camada de saída é linear. O algoritmo de otimização utilizado foi o Adam devido a sua boa adaptabilidade ao problema. A função custo escolhida foi a dos mínimos médios quadráticos, que é um bom parâmetro para casos de regressão em que dados muito distantes são alamente penalizados e dados mais próximos da função são menos penalizados. A figura a seguir mostra o diagrama da rede neural:

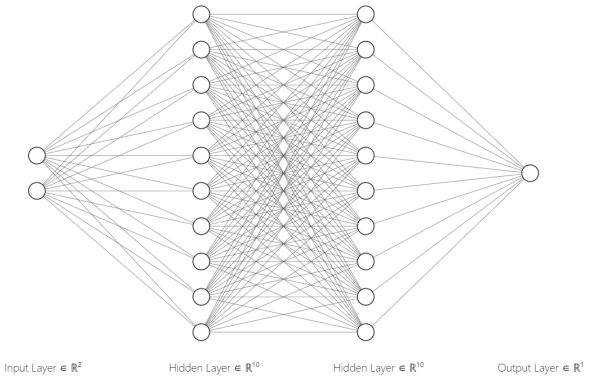


Figure 5. Arquitetura de rede perceptron de múltiplas camadas 2:10:10:1

Ao rodar o código que encontra-se no apéndice, com 500 epochs e uma arquitetura relativamente simples, obtém-se um resultado consideravelmente bom com um erro baixo e uma função bem aproximada da real, como mostrado nas Fig. 6 e Fig. 7 a seguir:

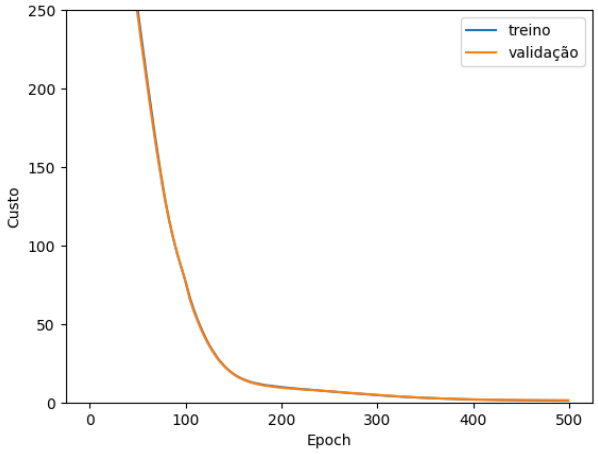


Figure 6. Custo para teste e validação para 2:10:10:1.

Observa-se que uma rede neural relativamente simples é capaz de solucionar esse problema com um valor final de erro baixo. Porém, foi realizado um teste utilizando uma rede neural profunda com a seguinte arquitetura 2:50:100:200:100:50:10, ou seja, bem mais complexa. Com esta mudança há resultados notórios, como convergência em um número menor de epochs e uma função prevista bem mais próxima da função real, como mostrado nas Fig. 8 e Fig. 9.

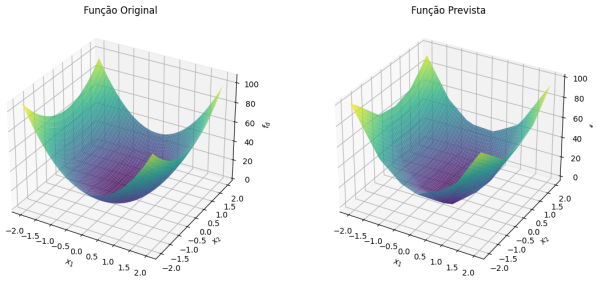


Figure 7. Representação 3D da função resultado treinada pelo modelo para 2:10:10:1.

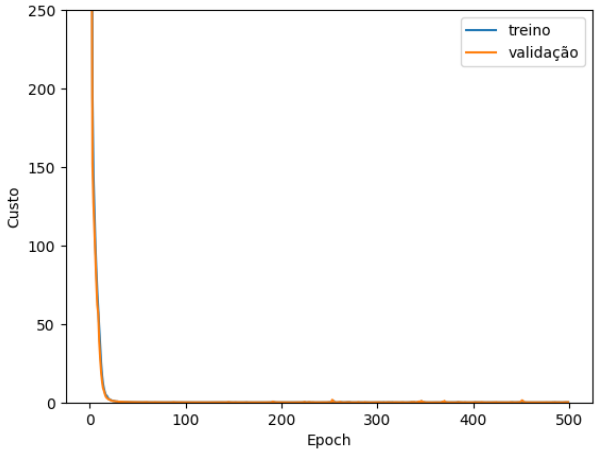


Figure 8. Custo para o caso da rede mais complexa.

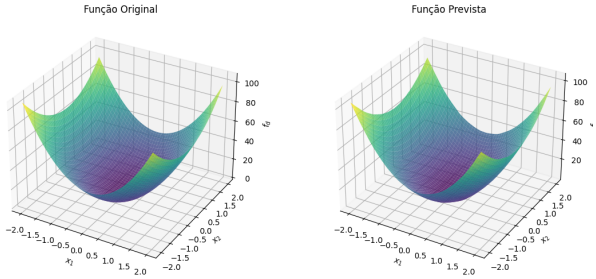


Figure 9. Representação 3D da função prevista pela rede mais complexa.

9.2 Problema das espirais

As curvas das espirais são dadas pelas seguintes equações:

$$x = \frac{\theta}{4}\cos\theta, \quad y = \frac{\theta}{4}\sin\theta, \quad \theta \geqslant 0 \tag{70}$$

$$x = (\frac{\theta}{4} + 0.8)\cos\theta, \quad y = (\frac{\theta}{4} + 0.8)sin\theta, \quad \theta \geqslant 0 \tag{71}$$

As equações resultam na Fig. 10. Este é um problema de classificação relativamente complexo para uma rede neural, portanto se fazendo necessário múltiplas camadas ocultas. Para resolver esse problema, foi feito um código em Python utilizando TensorFlow que encontra-se no apéndice. A arquitetura utilizada foi de uma rede neural de múltiplas camadas em forma de funil com duas variáveis de entrada e apenas uma de saída com 6 camadas ocultas formadas da seguinte maneira 2:500:400:300:200:100:50:1. Foi escolhida essa arquitetura após diversos testes, tendo esse o melhor resultado dentro dos limites computacionais aceitáveis. As camadas ocultas utilizam a função de ativação ReLu para que a derivada da função seja maior que zero e o resultado não atinga divergência. Já a camada de saída utiliza a função de ativação sigmoide, devido o tipo de problema ser de classificação, essa função indica a probabilidade de cada classe. O algoritmo de otimização foi o Adam e a função custo a entropia cruzada binária, também devido ao problema de classificação.Após rodar o código, que obteve ótimos resultados, tem-se a seguinte matriz de confusão para os dados de teste e o seguinte gráfico do custo durante o treinamento, mostrado nas Fig. 11 e Fig. 12, respectivamente.

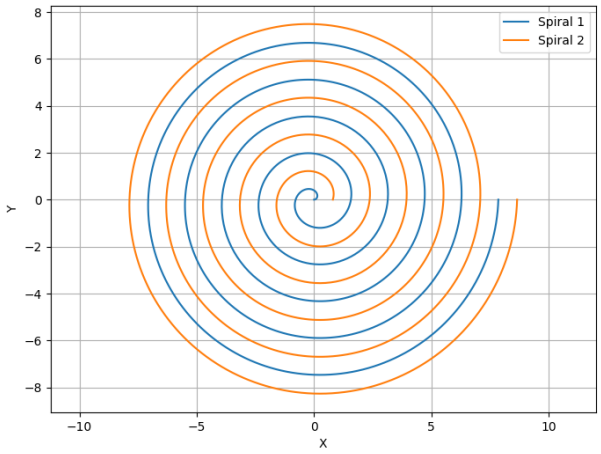


Figure 10. Espirais geradas pelas equações.

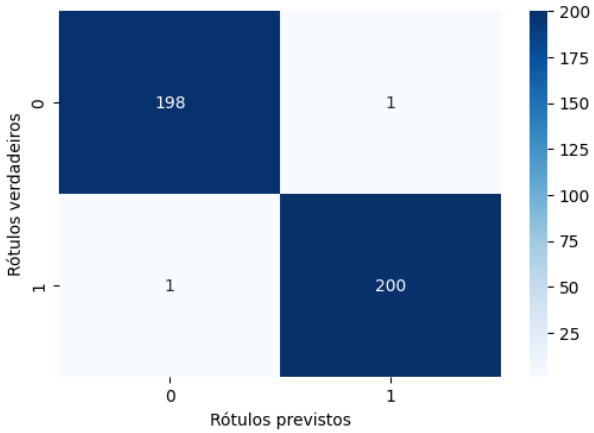


Figure 11. Matriz de confusão dos dados de teste.

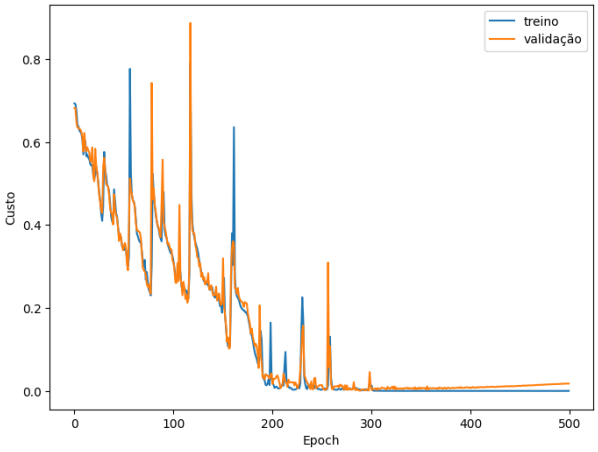


Figure 12. Custo vs epochs do treinamento.

Pode-se agora utilizar o módelo para prever as classes de uma grade de pontos, para testar efetivamente e visualmente o modelo, como mostrado na Fig. 13.

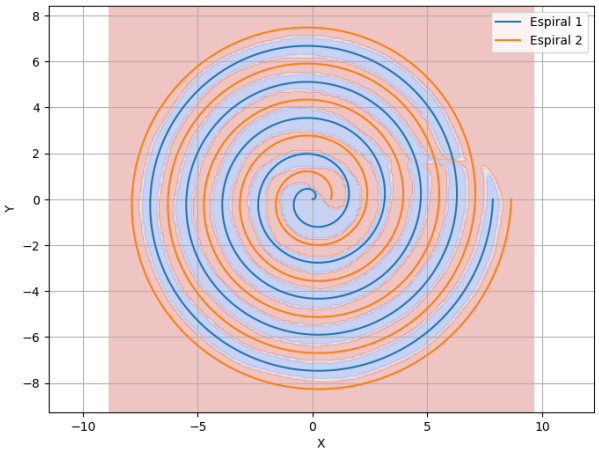


Figure 13. Previsão do modelo.

9.3 Problema de classificação de 5 padrões

Este problema propõe uma geometria de um quadrado com 4 semicírculos dispostos dentro dele de tal forma que 5 área delimitadas são geradas. Pode-se visualizar a geometria do problema na Fig. 14. As equações que delimitam a região azul, amarela, verde e vermelha são descritas pelas Eq. 72, Eq. 73, Eq. 74, Eq. 75, respectivamente, e o quadrado pela Eq. 76.

$$-\sqrt{1-x^2}+1 \leqslant y \leqslant \sqrt{1-(x-1)^2}, \quad 0 < x < 1 \tag{72}$$

$$-\sqrt{1-(x-1)^2} \leqslant y \leqslant \sqrt{1-x^2}-1, \quad 0 < x < 1 \tag{73}$$

$$-\sqrt{1-(x+1)^2} \leqslant y \leqslant \sqrt{1-x^2}-1, \quad -1 < x < 0 \tag{74}$$

$$-\sqrt{1-x^2}+1 \leqslant y \leqslant \sqrt{1-(x+1)^2}, \quad -1 < x < 0 \tag{75}$$

$$x = 1, \quad x = -1, \quad y = 1, \quad y = -1 \tag{76}$$

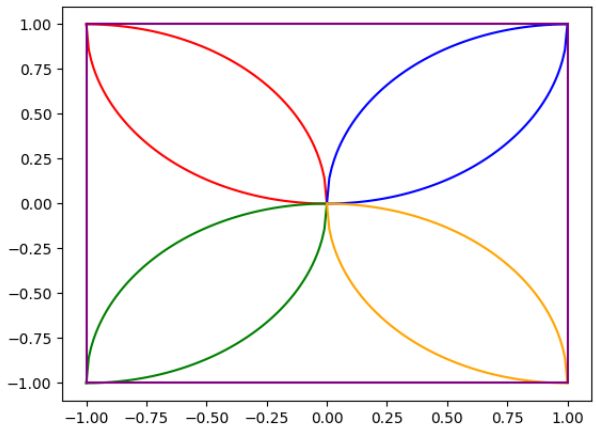


Figure 14. Geometria proposta pela problema.

Então, com a geometria definida, pode-se gerar os dados do problema de classificação, 10000 pontos foram gerados aleatoriamente no total, de tal forma que a distribuição dos pontos por classe pode ser visualizado no histograma da Fig. 15 e também pode-se visualizar os pontos gerados na própria geometria na Fig. 16. Naturalmente, há mais pontos gerados na classe 5, devido possuir maior área na geometria.

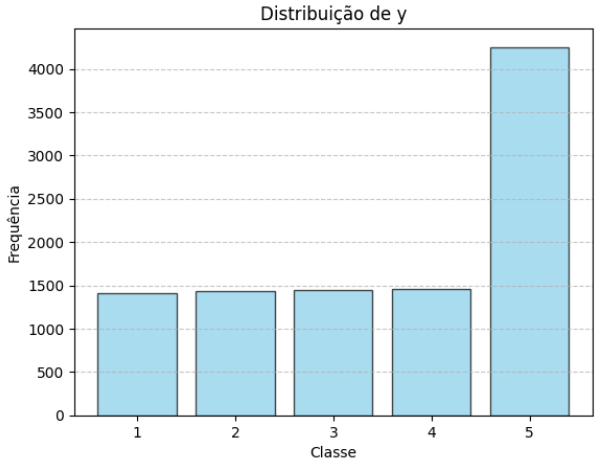


Figure 15. Histograma da distribuição dos pontos gerados por classe.

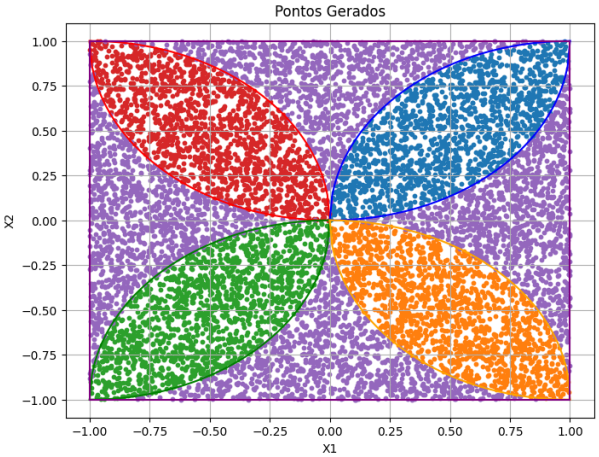


Figure 16. Pontos gerados na geometria.

Após gerar os pontos, os conjuntos de treinamene e teste fora divididos, de tal maneira que 20% foram destinados para o treinamento e 80% para o teste. Foi decidido esta divisão, pois há um grande número de pontos gerados, portanto é interessante que apenas uma quantidade suficiente de pontos seja destinada ao treinamento para que faça sentido o teste do modelo, com pontos excessivos de treinamento, basicamente qualquer modelo será capaz de resolver. Após diversos testes, encontrou-se uma arquitetura de numeros de neurônios e camadas equilibradas de 2:50:50:50:50:6. Durante os testes de diferentes modelos, observou-se que um número elevado de camadas causava ruído na função custo, atrapalhando a convergência, e um número elevados de neurônios impossiblitava de o modelo atingir valores mais baixos de erro. O algoritmo otimizador utilizado foi o Adam e a função custo foi a entropia cruzada categórica esparsa, esta que é mais indicada para problem de classficação de multiclasses, também é a responsável por existir 6 neurónios na camida de saída, mesmo existindo apenas 5 padrões para classificar, devido a como o limite do seu cálculo é realizada. As funções de ativação foram ReLu nas camadas ocultas e na camada de saída foi a Softmax que proporciona a probabilidade de em qual classe está o conjunto de entrada de forma eficaz. Por fim, o modelo é treinado por 500 epochs e um tamanho de lote de 500. O erro do teste foi de 0,059 e a acurácia de 97,5%. O código realizado em python utilizando Tensorflow pode ser encontrado no apéndice. Os resultados são mostrados a seguir:

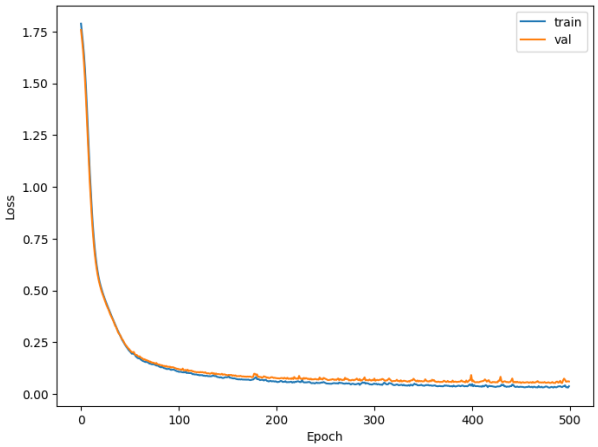


Figure 17. Função custo.

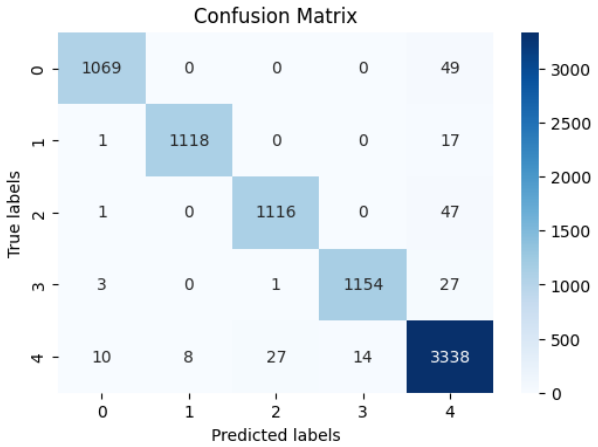
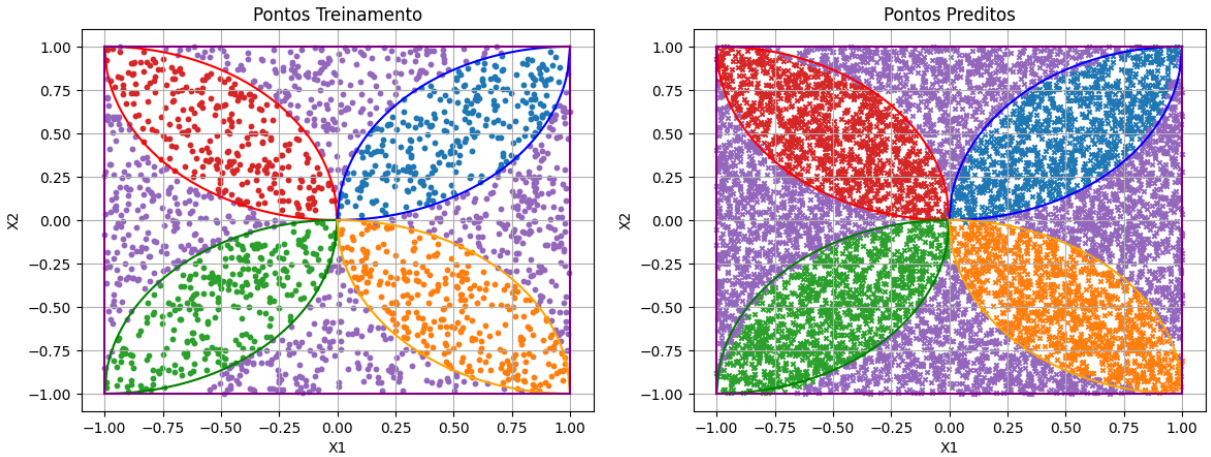


Figure 18. Matriz de confusão.



(a) Pontos de treinamento

(b) Pontos preditos

Figure 19. Comparação entre os pontos entregues para treinamento e os pontos de teste preditos pelo modelo.

10. Aprendizado Profundo para predições de Simulações Aerodinâmicas

A tese de Crichigno, *Deep Learning for prediction of Aerodynamic Simulations*, faz uma aplicação de Redes Neurais Artificiais (RNNs) na área de aerodinâmica, com o objetivo de desenvlver uma RNN de Perceptron de múltiplas camadas para prever os resultados de simulações de Dinâmica dos Fluidos Computacional (CFD) de um aerofólio bidimensional. O CFD é uma ferramente extremamente valiosa para análises aerodinÂmicas no geral, atulamente existem diversos *softwares*, tanto comerciais como código aberto, capazes de simular modelos complexos de fluido e geometria. No estudo são implementadas duas RNNs, uma para prever os coeficientes aerodinâmicos de sustentação (CL) e de arrasto (CD) do aerofólio. E uma segunda RNN para prever o campo de velocidado ao redos do aerofólio.

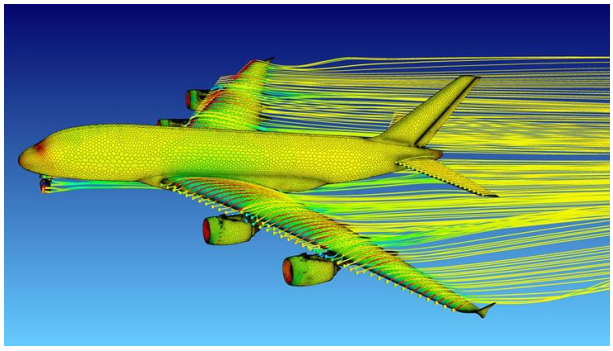


Figure 20. Exemplo CFD de aeronave.

10.1 Predição dos Coeficientes Aerodinâmicos

Primeiramente, para uma boa implementação de Perceptron de Múltiplas Camadas (MLP) é necessário um bom conjunto de dados para treinar, validar e testas, para isso o estudo dispões de 7056 resultados de simulações CFD. Essa primeira MLP utiliza como parâmetros de entrada as condições do escoamento: ângulo de ataque α, número de Mach M_∞ e número de Reynolds Re_∞.

DADOS DE ENTRADA	TARGETS
$[\alpha, M_\infty, Re_\infty]_1$	$[C_L, C_D]_1$
$[\alpha, M_\infty, Re_\infty]_2$	$[C_L, C_D]_2$
$\vdots$	$\vdots$
$[\alpha, M_\infty, Re_\infty]_N$	$[C_L, C_D]_N$

Table 1. Conjunto de dados para os coeficientes aerodinâmicos, N = 7056 amostras

10.1.1 Preparação do conjunto de dados

A divisão para o conjunto de dados do estudo é: Treinamento 70%, Validação 20% e Teste 10%. Outra ação realizada pelo estudo consiste em normalizar os dados de entrada, uma vez que a magnitude dos parâmetros varia consideravelmente em ordem de magnitude. Isso pode levar a estimativas de importância discrepantes para cada parâmetro, prejudicando os resultados do treinamento. A normalização é realizada em relação ao valor médio e ao desvio padrão do conjunto de Treinamento apenas. Esta abordagem é adotada porque o conjunto de treinamento compreende as amostras nas quais a rede aprende o padrão que liga a entrada à saída.

10.1.2 Função Custo

A função custo escollhida pelo estudo é a do Erro Quadrático Médio (MSE), vinto que este é um problema de regressão, então essa é uma métrica comumenta adotada.

10.1.3 Algoritmo de Treinamento

O algoritmo adotado para essa primeira MLP é o algoritmo Adam com uma taxa de aprediagem dinâmica.

10.1.4 Hiperparâmetros: Tamanho de lote

O estudo realiza *grid search* para encontrar os melhores valores para alguns hiperparâmetros selecionados. Um deles é o tamanho do lote, com os valores de 25, 50, 100 e 150. Esse hiperparâmetro coloca na balança a acurácia e o tempo de treinamento.

10.1.5 Hiperparâmetros: Arquitetura da Rede

A MLP do estudo possui 3 entradas e 2 saídas, e busca varías a quantidade de camadas ocultas e números de neurônios para econtrar uma arquitetura otimizada. Todas as camadas ocultas utilizam a função de ativação ReLu e a camada de saída é linear. As arquiteturas testadas foram:

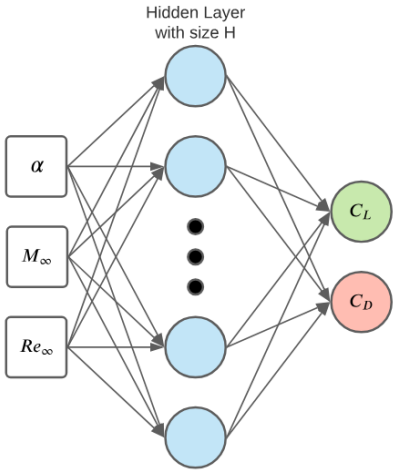


Figure 21. Arquitetura de Única Camada Oculta

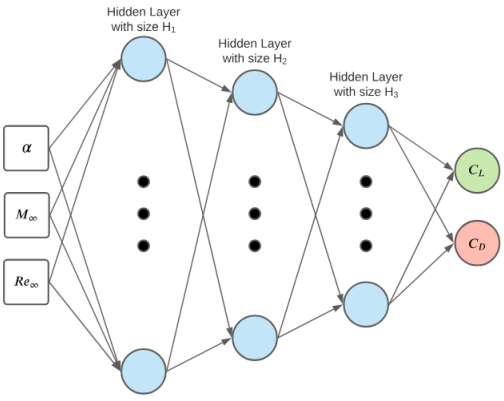


Figure 22. Arquitetura de Funil

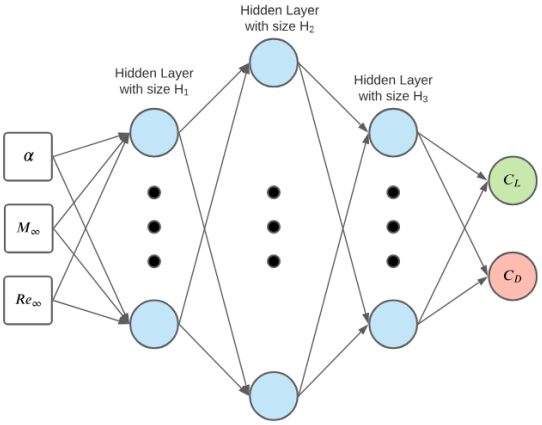


Figure 23. Arquitetura Rhombus

Camadas Ocultas						
H_1	H_2	H_3	H_4	H_5	H_6	H_7
40						
100						
40	20					
100	10					
80	40	20				
160	80	40	20			
20	40	20				
20	40	80	40	20		
20	40	80	160	80	40	20

Table 2. Quantidade de neurônios por camada oculta.

10.2 Resultados

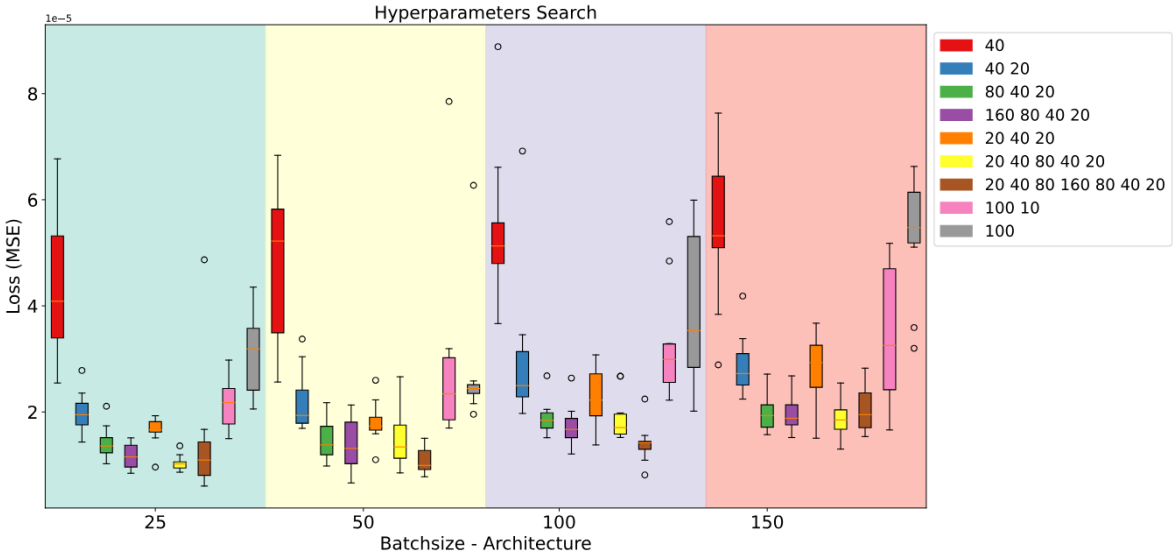


Figure 24. Resultados do gridsearch para o MLP dos coeficientes C_L e C_D.

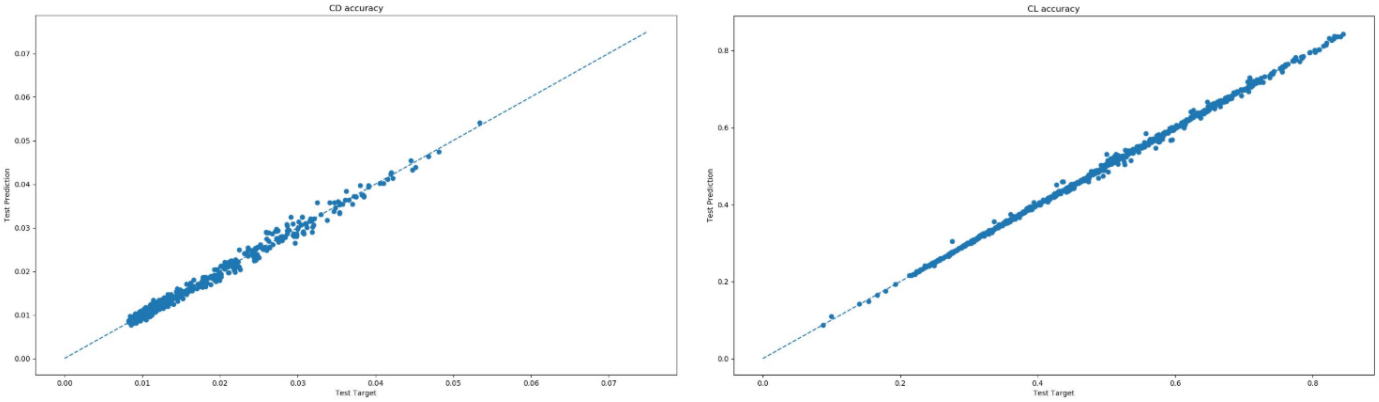


Figure 25. Acurácia para a predição de teste para C_L (direita) e C_D (esquerda).

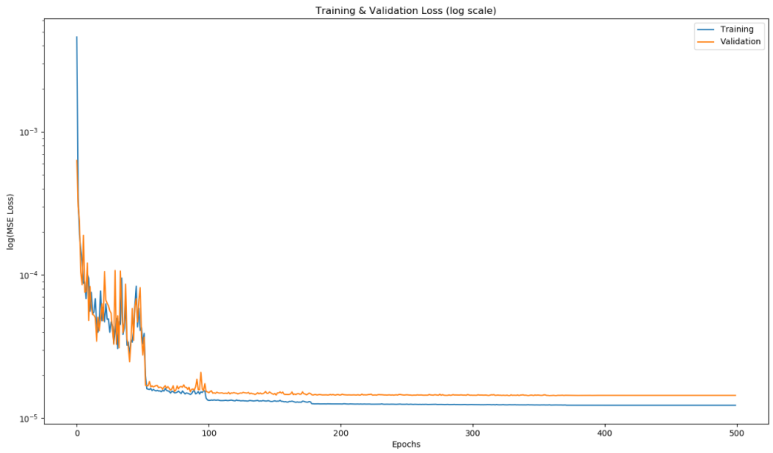


Figure 26. Custo vs Epochs da MLP.

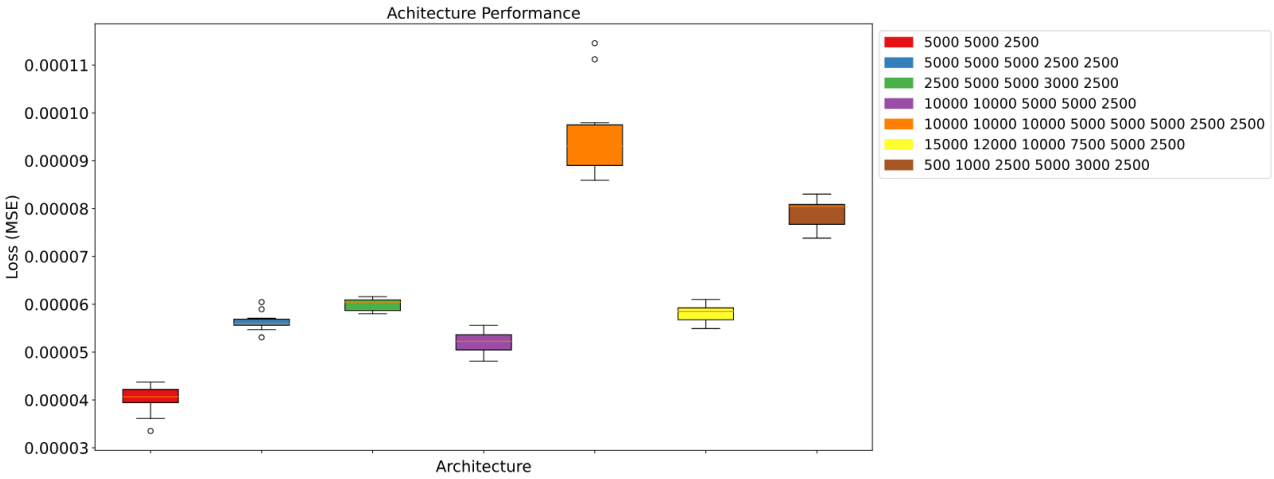


Figure 27. Resultados da gridsearch para a MLP da imagem cinza.

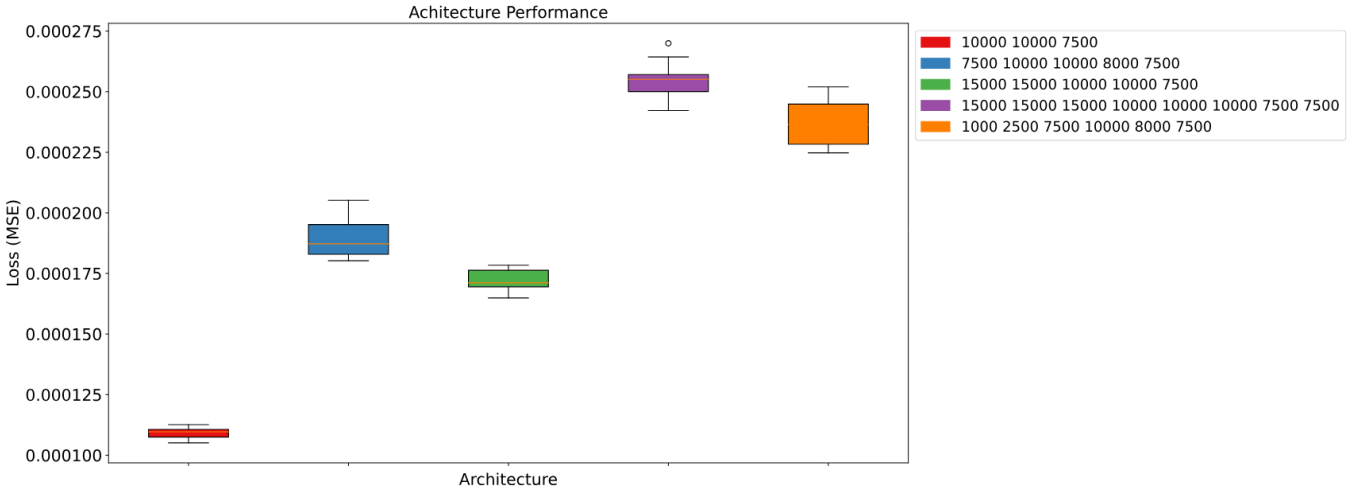


Figure 28. Resultados da gridsearch para a MLP da imagem colorida.

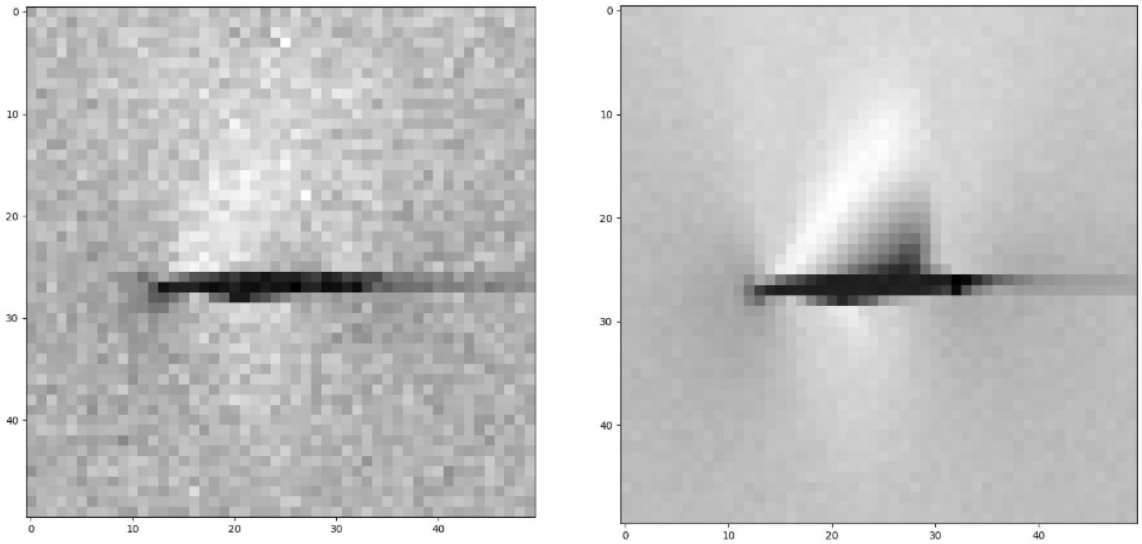


Figure 29. Predição do teste no começo do treinamento (esquerda) e após 500 epochs.

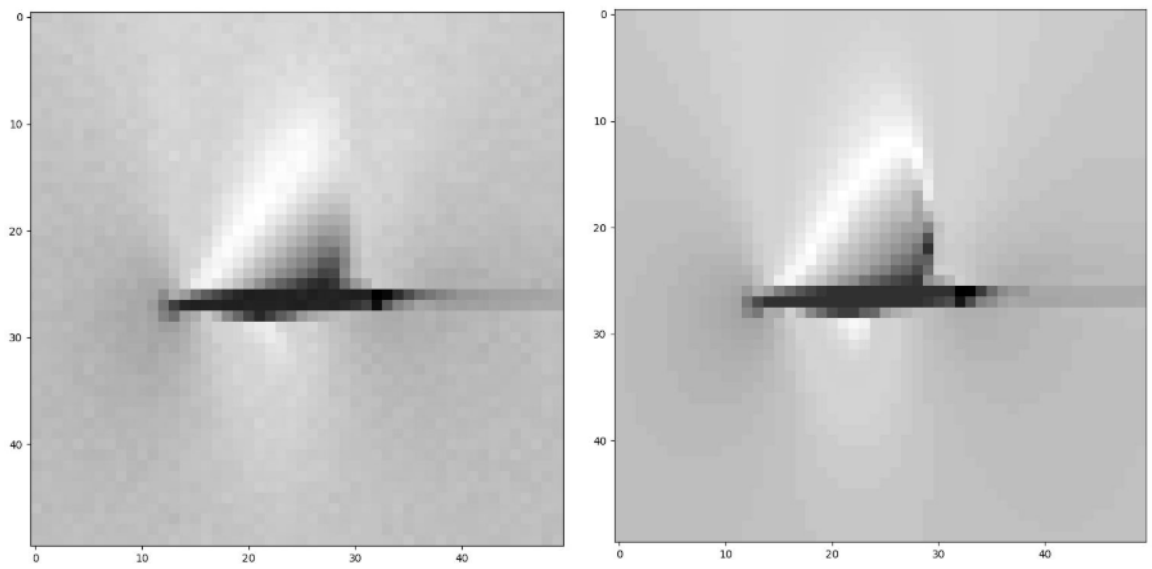


Figure 30. Comparação da imagem após o treinamento (esquerda) com a imagem desejada (direita).

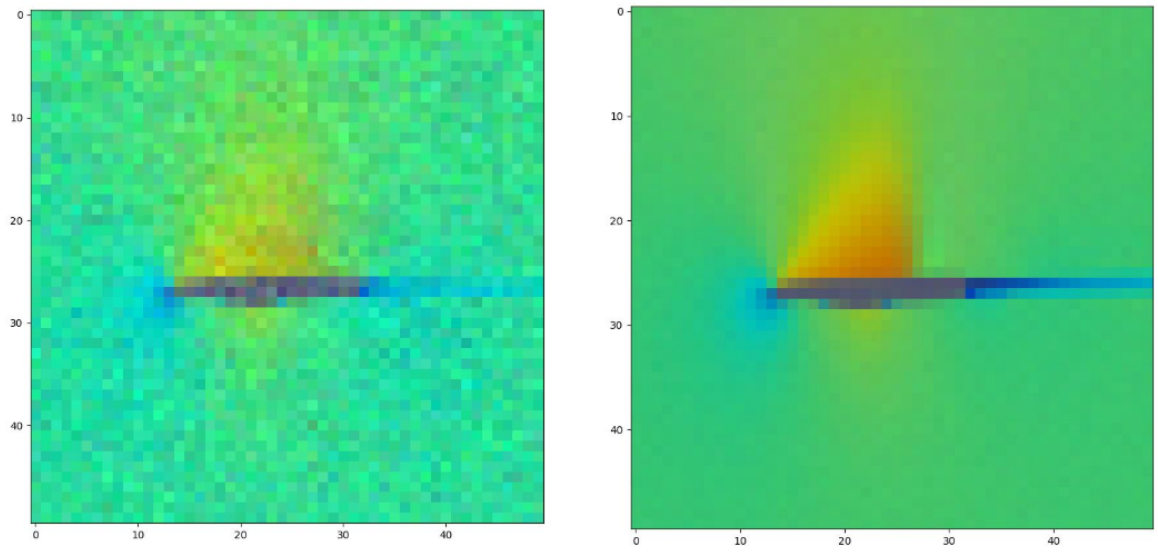


Figure 31. Predição do teste no começo do treinamento (esquerda) e após 500 epochs.

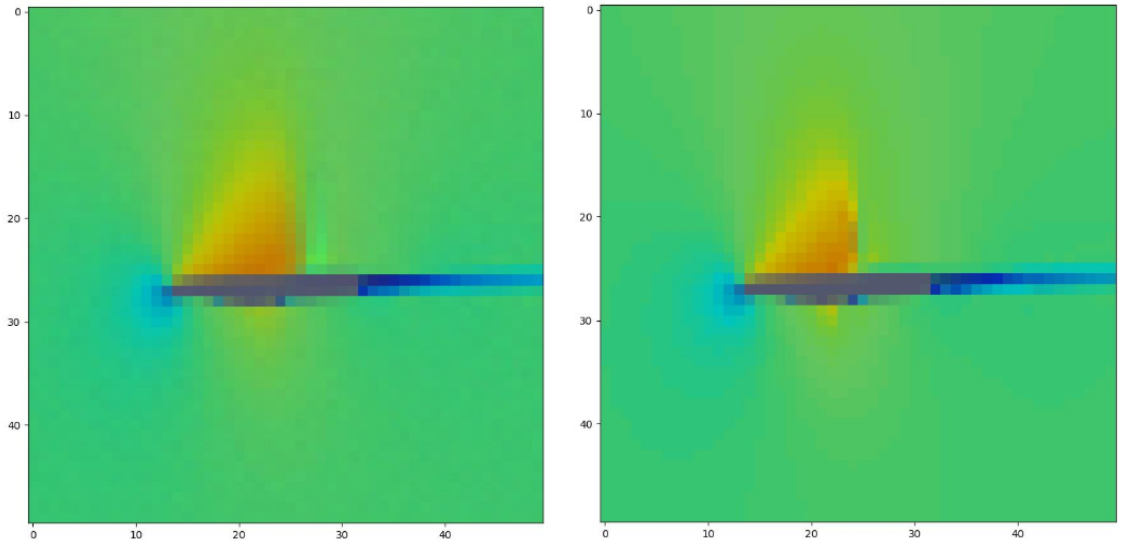


Figure 32. Comparação da imagem após o treinamento (esquerda) com a imagem desejada (direita).

Appendices

A Código Python questão 9-1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Definir a fun  o alvo
def f_d(x1, x2):
    return 16*x1**2 + x1*x2 + 8*x2**2 - x1 - x2 + np.log(1 + x1**2 + x2**2)

# Gerar dados de entrada
np.random.seed(42)
n_samples = 1000
X = np.random.uniform(-2, 2, (n_samples, 2))

# Calcular os valores de sa da correspondentes
y = np.array([f_d(x[0], x[1]) for x in X])

# Dividir os dados em conjuntos de treinamento e valida  o
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Construir o modelo da rede neural
model = Sequential([
    Dense(10, activation='relu', input_shape=(2,)),
    Dense(10, activation='relu'),
    Dense(1)  # Camada de sa da
])

# Compilar o modelo
model.compile(optimizer='Adam', loss='mean_squared_error')

# Treinar o modelo
history = model.fit(X_train, y_train, epochs=500, batch_size=50, validation_data=(X_val, y_val
                                                    ))

# Visualizar a curva do erro de treinamento e valida  o
plt.plot(history.history['loss'], label='treino')
plt.plot(history.history['val_loss'], label='valida  o')
plt.ylim([0,250])
plt.xlabel('Epoch')
plt.ylabel('Custo')
plt.legend()
plt.show()

# Gerar uma grade de pontos para plotagem
x1_values = np.linspace(-2, 2, 100)
x2_values = np.linspace(-2, 2, 100)
x1_grid, x2_grid = np.meshgrid(x1_values, x2_values)
X_grid = np.column_stack([x1_grid.ravel(), x2_grid.ravel()])

# Calcular as previs es da rede neural para a grade de pontos
y_pred = model.predict(X_grid).reshape(x1_grid.shape)

fig = plt.figure(figsize=(12, 5))

ax1 = fig.add_subplot(1, 2, 1, projection='3d')
ax1.plot_surface(x1_grid, x2_grid, f_d(x1_grid, x2_grid), cmap='viridis', alpha=0.8)
ax1.set_title('Fun  o Original')
ax1.set_xlabel('$x_1$')
ax1.set_ylabel('$x_2$')
ax1.set_zlabel('$f_d$')

ax2 = fig.add_subplot(1, 2, 2, projection='3d')
ax2.plot_surface(x1_grid, x2_grid, y_pred, cmap='viridis', alpha=0.8)
ax2.set_title('Fun  o Prevista')
ax2.set_xlabel('$x_1$')
ax2.set_ylabel('$x_2$')
```

```
ax2.set_zlabel('$f_d$')

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

B Código Python questão 9-2

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Definir as equa  es das espirais
def spiral_1(theta):
    x = (theta / 4) * np.cos(theta)
    y = (theta / 4) * np.sin(theta)
    return x, y

def spiral_2(theta):
    x = ((theta / 4) + 0.8) * np.cos(theta)
    y = ((theta / 4) + 0.8) * np.sin(theta)
    return x, y

# Gerar pontos das espirais
theta_1 = np.linspace(0, 10*np.pi, 1000)
x1, y1 = spiral_1(theta_1)

theta_2 = np.linspace(0, 10*np.pi, 1000)
x2, y2 = spiral_2(theta_2)

# Plotar as espirais
plt.figure(figsize=(8, 6))
plt.plot(x1, y1, label='Spiral 1')
plt.plot(x2, y2, label='Spiral 2')
plt.title('Spirals')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.axis('equal')

# Gerar dados para espiral 1
spiral_1_data = np.column_stack((x1, y1))
labels_1 = np.zeros(len(theta_1))

# Gerar dados para espiral 2
spiral_2_data = np.column_stack((x2, y2))
labels_2 = np.ones(len(theta_2))

# Combinar dados e r tulos
X = np.concatenate((spiral_1_data, spiral_2_data), axis=0)
y = np.concatenate((labels_1, labels_2))

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Construir o modelo da rede neural
model = Sequential([
    Dense(500, activation='relu', input_shape=(2,)),
    Dense(400, activation='relu'),
    Dense(300, activation='relu'),
    Dense(200, activation='relu'),
    Dense(100, activation='relu'),
    Dense(50, activation='relu'),
    Dense(1, activation='sigmoid')  # Camada de sa da
])

# Compilar o modelo
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Treinar o modelo
history = model.fit(X_train, y_train, epochs=500, batch_size=50, validation_data=(X_test,
                                                    y_test), verbose=0)

# Avaliar o modelo
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_accuracy}')

# Gerar previs es
y_pred = model.predict(X_test)
y_pred_binary = np.round(y_pred)

# Calcular matriz de confus o
conf_matrix = confusion_matrix(y_test, y_pred_binary)
print('Confusion Matrix:')
print(conf_matrix)

# Plotar matriz de confus o
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Visualizar a curva do erro de treinamento e valida  o
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Gerar uma grade de pontos para fazer previs es
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# Prever as classes para cada ponto na grade
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plotar as espirais reais
plt.figure(figsize=(8, 6))
plt.plot(x1, y1, label='Espiral 1')
plt.plot(x2, y2, label='Espiral 2')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.axis('equal')

# Plotar as regi es previstas pela rede neural
plt.contourf(xx, yy, Z, alpha=0.3, cmap='coolwarm')
plt.show()
```

C Código Python questão 9-3

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

def plot_problem():
    xp = np.linspace(0,1,100) #Espa o de pontos x>0
    xm = np.linspace(-1,0,100) #Espa o de pontos x<0

    ycm = -np.sqrt(1-xm**2)+1 # Curva superior direta
    ycp = -np.sqrt(1-xp**2)+1 # Curva superior esquerda
```

```
    ybm = np.sqrt(1-xm**2)-1 # Curva inferior esquerda
    ybp = np.sqrt(1-xp**2)-1 # Curva superior direita
    ydc = np.sqrt(1-(xp-1)**2) # Curva direita superior
    ydb = -np.sqrt(1-(xp-1)**2) # Curva direita inferior
    yeb = -np.sqrt(1-(xm+1)**2) # Curva esquerda inferior
    yec = np.sqrt(1-(xm+1)**2) # Curva esquerda superior

    plt.plot(xm,ycm, 'r') # Curva superior direta
    plt.plot(xm,yec, 'r') # Curva esquerda superior
    plt.plot(xp,ycp, 'b') # Curva superior esquerda
    plt.plot(xp,ydc, 'b') # Curva direita superior
    plt.plot(xm,ybm, 'g') # Curva inferior esquerda
    plt.plot(xm,yeb, 'g') # Curva esquerda inferior
    plt.plot(xp,ybp, 'orange') # Curva superior direita
    plt.plot(xp,ydb, 'orange') # Curva direita inferior

    #plot do quadrado
    plt.plot([-1, -1], [-1, 1], 'purple')
    plt.plot([1, 1], [-1, 1], 'purple')
    plt.plot([-1, 1], [-1, -1], 'purple')
    plt.plot([-1, 1], [1, 1], 'purple')

    plt.show()

plot_problem()
# Fun   o para gerar os dados
def generate_data(num_samples):
    X = np.zeros((num_samples, 2))
    y = np.zeros(num_samples, dtype=int)

    for i in range(num_samples):
        x1 = np.random.uniform(-1, 1)
        x2 = np.random.uniform(-1, 1)

        if x1 >= 0 and x1 <=1:
            if x2 >= -np.sqrt(1-x1**2)+1 and x2 <= np.sqrt(1-(x1-1)**2):
                y[i] = 1
            elif x2 >= -np.sqrt(1-(x1-1)**2) and x2 <= np.sqrt(1-x1**2)-1:
                y[i] = 2
            else:
                y[i] = 5
        elif x1 >=-1 and x1 <0:
            if x2 >= -np.sqrt(1-(x1+1)**2) and x2 <= np.sqrt(1-x1**2)-1:
                y[i] = 3
            elif x2 >= -np.sqrt(1-x1**2)+1 and x2 <= np.sqrt(1-(x1+1)**2):
                y[i] = 4
            else:
                y[i] = 5
        else:
            y[i] = 5


        X[i] = [x1, x2]

    return X, y

# Gerar dados
num_samples = 10000
X, y = generate_data(num_samples)


# Plotar histograma de y
plt.hist(y, bins=np.arange(1, 7) - 0.5, rwidth=0.8, color='skyblue', edgecolor='black', alpha=
                                                    0.7)
plt.xticks(range(1, 6))
plt.xlabel('Classe')
plt.ylabel('Frequ ncia')
plt.title('Distribui  o de y')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

# Plotar os pontos gerados
plt.figure(figsize=(8, 6))
```

```
for label in range(1, 6):
    plt.scatter(X[y == label][:, 0], X[y == label][:, 1], label=f'Classe {label}', s=10)


plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Pontos Gerados')
plt.grid(True)
plot_problem()
plt.show()
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import seaborn as sns

# Dividir os dados em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.8, random_state=42)

# Construir o modelo da rede neural
model = Sequential([
    Dense(50, activation='relu', input_shape=(2,)),
    Dense(50, activation='relu'),
    Dense(50, activation='relu'),
    Dense(50, activation='relu'),
    Dense(6, activation='softmax')  # Camada de sa da
])

# Compilar o modelo
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['
                                            sparse_categorical_accuracy'])

# Treinar o modelo
history = model.fit(X_train, y_train, epochs=500, batch_size=500, validation_data=(X_test,
                                            y_test), verbose=0)
# Avaliar o modelo
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_accuracy}')

# Gerar previs es
y_pred = model.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)

# Calcular matriz de confus o
conf_matrix = confusion_matrix(y_test, y_pred_labels)
print('Confusion Matrix:')
print(conf_matrix)

# Plotar matriz de confus o
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g')
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()

# Visualizar a curva do erro de treinamento e valida  o
plt.figure(figsize=(8, 6))
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()


# Plotar os pontos previstos
for label in range(1, 6):
```

```
    plt.scatter(X_test[y_pred_labels == label][:, 0], X_test[y_pred_labels == label][:, 1],
                                                label=f'Predito {label}', marker='x', s=10)

plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Pontos Preditos')
#plt.legend()
plt.grid(True)
plot_problem()
plt.show()

# Plotar os pontos de cada classe no conjunto de teste
for label in range(1, 6):
    plt.scatter(X_train[y_train == label][:, 0], X_train[y_train == label][:, 1], label=f'
                                                Classe {label}', s=10)

plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Pontos Treinamento')
#plt.legend()
plt.grid(True)
plot_problem()
plt.show()
```

Texto apenas para inserir referências Goodfellow *et al.* (2016) Haykin (2009) Deisenroth *et al.* (2020) Crichigno (2021) Bourret (2018) Lenail (2019) class (2019)

D REFERENCES

Bourret, X., 2018. "Maximum likelihood estimators - multivariate gaussian". https://stats.stackexchange.com/questions/351549/maximum-likelihood-estimators-multivariate-gaussian. Accessed 05 May 2024.

class, S.C., 2019. "Cs231n: Convolutional neural networks for visual recognition". https://cs231n.github.io/neural-networks-1/actfun. Accessed 05 May 2024.

Crichigno, N., 2021. *Deep Learning for prediction of Aerodynamic Simulations*. Master's thesis, POLITECNICO DI TORINO.

Deisenroth, M.P., Faisal, A.A. and Ong, C.S., 2020. *MATHEMATICS FOR MACHINE LEARNING*. Cambridge University Press. Https://mml-book.com.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Haykin, S., 2009. *Neural Networks and Learning Machines*. Pearson.

Lenail, A., 2019. "Nn-svg: Publication-ready neural network architecture schematics". https://alexlenail.me/NN-SVG/index.html. Accessed 05 May 2024.