Alessandro Soares da Silva
Matrícula: 20231023705

1° LISTA DE DEEP LEARNING
FUNDAMENTOS MATEMÁTICOS

SUMÁRIO

1 - As métricas de distancia entre vetores são aplicadas nos estudos de aprendizagem de máquina como medidas de similaridade/dissimilaridade entre vetores que representam padrões. Apresente um estudo sobre as seguintes distancias: Distância Euclidiana, Distância de Minkowski, Distância City Block, Distância de Mahalanobis, Coeficiente de Correlação de Pearson e Similaridade Cosseno. Apresente neste estudo aplicações onde cada tipo de métricas de distância é mais adequada.

Sugestão de Aplicações: Classificação de padrões, Clustering, Reconhecimento de padrões, etc.

RESOLUÇÃO:

Distância Euclidiana

A **distância Euclidiana** é a distância entre dois pontos no espaço euclidiano. Por definição a distância euclidiana entre dois posntos P e Q é o comprimento do segmento de reta que liga eses pontos (PQ). Se P = $(p_1,p_2,...,p_n)$ e Q=$(q_1,q_2,...,q_n)$ são pontos em um espaço euclediano n-dimensional, então a distância entre P e Q é obtida pela fórmula de Pitágoras:

$$distancia\,(P,Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$$

$$distancia\,(P,Q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Em um espaço euclidiano n-dimensional a posição de um ponto trata-se de um vetor, dessa forma esse vetor (ponto no espaço) pode ser representado como um vetor euclidiano a partir da origem do espaço. Assim, o comprimento desse vetor pode ser obtido pela sua norma:

$$\|P\| = \sqrt{p_1^2 + p_2^2 + \ldots + p_n^2}$$

Logo, se um vetor é descrito como segmento de linha da origem do espaço euclidiano a um determinado ponto no espaço, o seu comprimento é a distância da origem do plano ao ponto especificado. Assim, a norma euclidiana é vista como a distância entre a origem e o ponto no espaço.

A distância euclidiana é inastifatória para muitas situações estatísticas. Isso ocorre devido a contribuição de cada coordenada ter o mesmo peso para o cálculo da distância. Quando estas coordenadas representam medidas provenientes de um processo que sofre flutuações aleatórias de diferentes magnitudes é desejável ponderar as coordenadas com grande variabilidade por menores pesos em relação àquelas com baixa variabilidade.

Propriedades da Distância Euclidiana:

A distância Euclidiana possui várias propriedades importantes:

1. **Positividade:** A distância entre dois pontos é sempre não negativa.
2. **Identidade dos Indiscerníveis:** A distância entre dois pontos é zero se e somente se os pontos forem idênticos.
3. **Simetria:** A distância entre dois pontos é a mesma, independentemente da ordem dos pontos.
4. **Desigualdade Triangular:** A distância entre dois pontos é sempre menor ou igual à soma das distâncias de um ponto a um terceiro ponto intermediário.

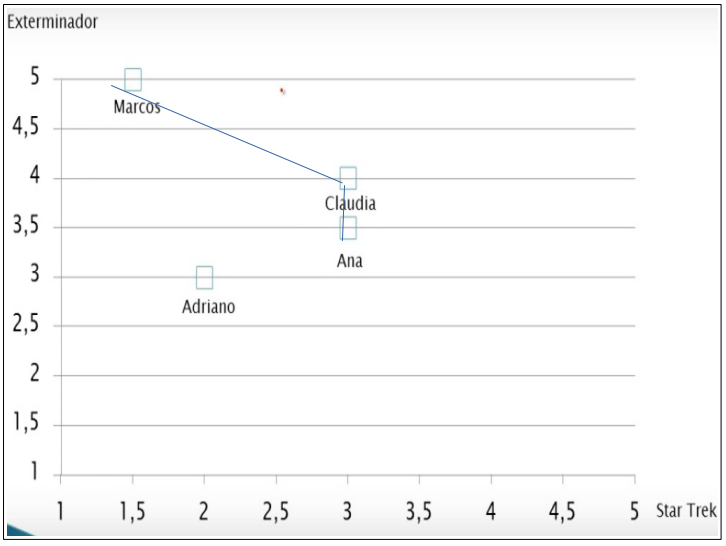
Aplicações da Distância Euclidiana:

A distância Euclidiana é aplicada em uma variedade de contextos em ciência de dados e matemática:

1. **Classificação de Padrões:** Usada em algoritmos como k-NN (k-vizinhos mais próximos) para determinar a similaridade entre instâncias.
2. **Reconhecimento de Padrões:** Pode ser usada para comparar características de objetos em reconhecimento de imagem.
3. **Análise de Dados:** Útil para medir a dispersão de dados e calcular distâncias entre pontos em espaços multidimensionais.
4. **Geometria Computacional:** Aplicada em problemas de geometria, como encontrar o ponto mais próximo em um espaço euclidiano.

Exemplo de Aplicação:

Suponha que temos um conjunto de dados que representam as notas que cada usuário deu para os filmes “Exterminador” e “Star Trek”.. Podemos usar a distância Euclidiana para medir a similaridade entre os usuários com base em suas avaliações. Essa medida de similaridade pode ser útil para classificar novos usuários.



Ana x Claudia

- $x_i = 3.0, 3.5$
- $y_i = 3.0, 4.0$

$$D(x,y)=\sqrt{(3.0-3.0)^2*(3.5-4.0)^2}$$
$$D(x,y)=0.5$$

$$D(x,y)=\sqrt{(3.0-1.5)^2*(3.5-5.0)^2}$$
$$D(x,y)=2.12$$

Uma outra forma de se obter a distancia acima é com a matriz transposta ou com a norma de uma vetor:

$$\mathbf{a} = \mathbf{p} - \mathbf{q} = (q_1 - p_1, q_2 - p_2, \cdots, q_n - p_n)$$

$$\mathbf{a^T a} = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \cdots \\ a_n \end{bmatrix} = a_1 \cdot a_1 + a_2 \cdot a_2 + \ldots + a_n \cdot a_n = \sum_{i=1}^{n} \left(a_i{}^2\right)$$

```
[43]  1 def bymatmult2(p,q):
      2     return np.sqrt(np.matmul(np.transpose(p-q),p-q))
      3
      4 bymatmult2(x1,x2)
```

```
[6]   1 def linalg_norm(p,q):
      2     return np.linalg.norm(p-q)
      3
      4 linalg_norm(x1,x2)
```

$$\|\mathbf{q} - \mathbf{p}\| = \sqrt{(\mathbf{q} - \mathbf{p}) \cdot (\mathbf{q} - \mathbf{p})}$$

$$\|\mathbf{a}\| = \sqrt{\mathbf{a} \cdot \mathbf{a}} = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2}$$

Distancia Minkowski

A Distância de Minkowski é uma métrica de distância generalizada que engloba várias outras métricas comuns, incluindo a Distância Euclidiana (quando p = 2) e a Distância Manhattan (quando p = 1). A Distância de Minkowski é definida pela fórmula:

$$D(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Onde:

- **x** e **y** são vetores n-dimensionais.
- **n** é o numero de dimensões dos vetores.
- **p** é um parâmetro que determina a ordem da métrica. Quando p = 1, é a distância de Manhattan, e quando p = 2, é a distância Euclidiana.

Embora a distância Eucledianа e distância de Minkowski sejam similares em muitos aspectos e compartilhem apliações comuns, existem situações em que uma é preferível em relação ao outra devido às suas prioridades específicas.

Euclidiana: é fundamental na geometria euclidianae na geometria analítica, sendo a métrica padrão para medir distância entre dois pontos em um espaço vetorial. Onde as dimesões são lineares e não têm correlação entre si.

Minkowski: é mais flexível do que a distância Euclidiana, pois permite ajustar o parâmetro *p,* o que pode ser útil em situações onde não se deseja tratar todas as dimenões dos dodos de forma igual. Dependendo do valor de *p*, o distância de Minkowski pode ser mais robusta contra outliers. Por exemplo, quando p > 2, o método dá menos peso a grandes desvios em dimensões individuais. Em conjuntos com dimensões que não têm a mesma importância ou escala, pode ser preferível, pois permite ajustar o parâmetro *p* para refletir a importância relativa das dimensões.

Exemplo de aplicação

Suponha que estamos trabalhando com um conjunto de dados que descreve diferentes tipos de veículos com base em suas características, como velocidade máxima, consumo de combustível por quilômetro, potência do motor e capacidade do tanque de combustível.

Nesse exemplo, cada veículo é representado por um vetor de características, e queremos calcular a distância entre dois veículos para determinar sua similaridade. No entanto, as diferentes características dos veículos podem ter escalas muito diferentes e podem ter importância variável na determinação da similaridade.

Ex:

Veículo A:

Velocidade máxima: 200 Km/h

Consumo de combustível por quilômetro: 8 L/Km

Potência do motor: 150 cavalos de potência

Capacidade do tanque de combustível: 60 litros

Veículo B:

Velocidade máxima: 180 Km/h

Consumo de combustível por quilômetro: 10 L/Km

Potência do motor: 200 cavalos de potência

Capacidade do tanque de combustível: 70 litros

Distância de Minkowski ($p = 3$):

A fórmula para a Distância de Minkowski com $p = 3$ entre dois vetores x e y é:

$$D(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^3\right)^{\frac{1}{3}}$$

Calculando a distância de Minkowski entre os veículos A e B com $p = 3$:

$$D_{\text{Minkowski}}(A, B) = \left(|200 - 180|^3 + |8 - 10|^3 + |150 - 200|^3 + |60 - 70|^3\right)^{\frac{1}{3}}$$

$$= \left(20^3 + (-2)^3 + (-50)^3 + (-10)^3\right)^{\frac{1}{3}}$$

$$= (8000 - 8 - 125000 - 1000)^{\frac{1}{3}}$$

$$= (-116008)^{\frac{1}{3}}$$

$$\approx -48.77$$

Este exemplo ilustra como a Distância de Minkowski, com um valor adequado de ***p***, pode fornecer uma medida de distância que leva em consideração as diferenças nas escalas e importâncias relativas das características dos dados.

Distancia City Block

A Distância de City Block, também conhecida como Distância de Manhattan ou Norma L_1. A Distância de City Block entre dois pontos **p** e **q** em um espaço n-dimensional é a soma das diferenças absolutas de suas coordenadas:

$$D(p, q) = \sum_{i=1}^{n} |p_i - q_i|$$

Onde:

p e *p* são vetores n-dimensionais representando os pontos.

n é o número de dimensões dos vetores.

A distância de City Block recebe esse nome por que é a distância que você teria que percorrer em uma cidade com quarteirões ortogonais, com Manhattan. Ela calcula a distância entre dois pontos percorridos ao longo dos eixos (horizontal e vertical), sem levar em concideração linhas retas "diretas". Essa media é sensível apenas às mudanças de coordenadas ao longo dos eixos, não considerando a inclinação ou direção direta entre os postos.

Aplicações do City Block

- **Sensibilidade à orientação:** Em muitos casos, a orientação dos eixos pode ser uma consideração importante. Por exemplo, em problemas de trajetória em uma grade, a disância da city block é mais natural, pois as movimentações são restritas a movimentos ortogonais.
- **Simplicidade de cálculo:** é mais simples de calcular, ela envolve apenas somas e subtrações de coordenadas, enquanto a distância euclidiana requer cálculo de raízes quadradas.
- **Resistência a outliers:** a city block é menos sensível a outliers. Isso significa que pontos extremos não afetarão tanto a medida de distância, o que pode ser útil em conjuntos de dados com valores discrepantes.
- **Adequação a restrições de movimento:** Em problemas onde os movimentos são restritos a certas direções (como em redes de transporte ou em problemas de roteamento), a distância city block pode ser mais adequada, pois reflete melhor as opções de movimento disponíveis.

Vamos considerar um exemplo simples de aplicação da distância da city block em um contexto bidimensional, como um problema de navegação em uma grade. Suponha que temos dois pontos A e B em um plano cartesiano:

- Ponto A: coordenadas (2, 3)
- Ponto B: coordenadas (5, 7)

Para calcular a distância da city block entre esses dois pontos, usamos a fórmula:

$$\text{Distância} = |x_B - x_A| + |y_B - y_A|$$

Onde x_A, y_A são as coordenadas do ponto A e x_B, y_B são as coordenadas do ponto B.

Substituindo os valores, temos:

$$\text{Distância} = |5 - 2| + |7 - 3|$$
$$\text{Distância} = |3| + |4|$$
$$\text{Distância} = 3 + 4$$
$$\text{Distância} = 7$$

Essa distância pode ser interpretrada como um número total de quarteirões que precisamos percorrer para chegar do ponto A ao pontoB, movendo-se apenas para cima, para baixo, para a esquerda, para a direita em uma grade. Essa interpretação é intuitiva e útil em muitos cenários práticos, como planejamento de rotas.

Distância de Mahalanobis

A métrica de distância de Mahalanobis é uma medida de distância entre um ponto e um conjunto de pontos em um espaço multidimensional. É uma generalização da métrica Euclidiana e é especialmente útil quando os dados têm diferentes escalas ou covariâncias entre si.

A fórmula para calcular a distância de Mahalanobis entre um ponto x e um conjunto de pontos representados por uma matriz de dados x é dada por:

$$distancia\,(x,y) = \sqrt{(x-y)^T S^{-1} (x-y)}$$

Onde:

- x é o vetor de dados do ponto individual.
- μ é o vetor de médias dos dados (vetor médio).
- S é a matriz de covariância dos dados.

Essa métrica de distância leva em consideração as correlações entre as diferenças variáveis dos dados, ajustado a escala e a orinetação dos dodos.

Vamos considerar um exemplo simples para ilustrar:

Suponha que temos dois pontos em um espaço bidimensional com as seguintes coordenadas:

Ponto 1: $(x_1, y_1) = (3,4)$

Ponto 2: $(x_2, y_2) = (6,8)$

E suponha que a matriz de covariância dos dados seja:

$$\mathbf{S} = \begin{bmatrix} 4 & 2 \\ 2 & 3 \end{bmatrix}$$

$$\mathbf{S}^{-1} = \frac{1}{8}\begin{bmatrix} 3 & -2 \\ -2 & 4 \end{bmatrix} = \begin{bmatrix} \frac{3}{8} & -\frac{1}{4} \\ -\frac{1}{4} & \frac{1}{2} \end{bmatrix}$$

E o vetor de médias seja:

$$\mu = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

$$\mathbf{S} = \begin{bmatrix} 4 & 2 \\ 2 & 3 \end{bmatrix}$$

Podemos calcular sua inversa:

$$\mathbf{S}^{-1} = \frac{1}{|S|}\begin{bmatrix} 3 & -2 \\ -2 & 4 \end{bmatrix}$$

Para o Ponto 1:

$$\mathbf{x}_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$\mu = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

$$\mathbf{x}_1 - \mu = \begin{bmatrix} 3-5 \\ 4-6 \end{bmatrix} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

Para o Ponto 2:

$$\mathbf{x}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$

$$\mathbf{x}_2 - \mu = \begin{bmatrix} 6-5 \\ 8-6 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Para o Ponto 1:

$$D(\mathbf{x}_1) = \sqrt{(\mathbf{x}_1 - \mu)^T \mathbf{S}^{-1} (\mathbf{x}_1 - \mu)}$$
$$= \sqrt{\begin{bmatrix} -2 & -2 \end{bmatrix} \begin{bmatrix} \frac{3}{8} & -\frac{1}{4} \\ -\frac{1}{4} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} -2 \\ -2 \end{bmatrix}}$$
$$= \sqrt{\begin{bmatrix} -2 & -2 \end{bmatrix} \begin{bmatrix} -\frac{3}{4} - \frac{1}{2} \\ \frac{1}{2} - 1 \end{bmatrix}}$$
$$= \sqrt{\begin{bmatrix} 1 \\ -1 \end{bmatrix}^T \begin{bmatrix} -\frac{3}{4} - \frac{1}{2} \\ \frac{1}{2} - 1 \end{bmatrix}}$$
$$= \sqrt{(1 \times -\frac{3}{4} - 1 \times 1)}$$
$$= \sqrt{-\frac{3}{4} - 1}$$
$$= \sqrt{-\frac{7}{4}}$$
$$= \frac{\sqrt{7}}{2}$$

Para o Ponto 2:

$$D(\mathbf{x}_2) = \sqrt{(\mathbf{x}_2 - \mu)^T \mathbf{S}^{-1} (\mathbf{x}_2 - \mu)}$$
$$= \sqrt{\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} \frac{3}{8} & -\frac{1}{4} \\ -\frac{1}{4} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix}}$$
$$= \sqrt{\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} \frac{3}{8} - \frac{1}{2} \\ -\frac{1}{4} + 1 \end{bmatrix}}$$
$$= \sqrt{\begin{bmatrix} -\frac{1}{8} \\ \frac{3}{4} \end{bmatrix}^T \begin{bmatrix} -\frac{1}{8} \\ \frac{3}{4} \end{bmatrix}}$$
$$= \sqrt{(-\frac{1}{8} \times -\frac{1}{8}) + (\frac{3}{4} \times \frac{3}{4})}$$
$$= \sqrt{\frac{1}{64} + \frac{9}{16}}$$
$$= \sqrt{\frac{25}{16}}$$
$$= \frac{5}{4}$$

Portanto, a distância de Mahalanobis entre os ponto (3,4) e (6,8) é ($7^{1/2}$/2) para o Ponto 1 e 5/4 para o ponto 2.

Coeficiente de Correlação de Pearson

O coeficiente de correlação de Person é uma medida estatística que avalia a relaçao linear entre duas variáveis contínuas. Ele fornece uma medida da força e da direção linear entre duas variáveis contínuas. O coeficiente de correlação varia de -1 a +1, onde:

- +1 indica uma correlação positiva perfeita, o que significa que as duas variáveis estão direatamente relacionadas em uma relação linear positiva.

- -1 indica im correlação negativa perfeita, o que significa que as duas variáveis estão inversamente relacionadas em uma relação linear negativa.

0 indica ausência de correlação linear entre as variáveis.

O Coeficiente de Correlação de Pearson é amplamente utilizado em diversas áreas, como estatística, economia, psicologia, ciências sociais e muitas outras. Ele é umas ferramenta valiosa para explorar e entender as relações entre variáveis em conjuntos de dados.
A fórmula para calcular o Coefiente de Correlação de Pearson entre duas variáveis X e Y é dada por:

$$r = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$

Onde r é o Coeficiente de Correlação de Pearson.
Xi e Yi são os valores individuais das duas variáveis.
X' e Y'são as médias das duas variáveis, respectivamente.

Exemplo de Aplicação:

Vamos considerar um exemplo hipotético para demosntrar o uso do Coeficiente de Correlação de Pearson:

Suponha que estamos estudando a relação entre as horas de estudo semanais (variável X) e as pontuações em um exame final (variável Y) de um grupo de estudantes. Coletados os seguintes dados:

X: 10, 8, 12, 14, 6
Y: 85, 70, 90, 92, 65

Calulando o Coeficiente de Correlação de Pearson usando esses dados.
Para calcular o Coeficiente entre as horas de estudo semanais X e as pontuações em um exame final Y com os dados fornecidos, primeiro precisamos calcular as médias (X' e Y') e, em seguida, aplicar a fórmula do coeficiente de correlação.

1. **Cálculo das médias ($\bar{X}$ e $\bar{Y}$):**

$$\bar{X} = \frac{10+8+12+14+6}{5} = \frac{50}{5} = 10$$
$$\bar{Y} = \frac{85+70+90+92+65}{5} = \frac{402}{5} = 80.4$$

2. **Cálculo do numerador da fórmula:**

$$\sum (X_i - \bar{X})(Y_i - \bar{Y}) = (10-10)(85-80.4) + (8-10)(70-80.4) + (12-10)(90-80.4) + (14-10)(92-80.4) + (6-10)(65-80.4)$$
$$= (0)(4.6) + (-2)(-10.4) + (2)(9.6) + (4)(11.6) + (-4)(-15.4)$$
$$= 0 + 20.8 + 19.2 + 46.4 + 61.6$$
$$= 147.6$$

3. **Cálculo dos denominadores da fórmula:**

$$\sum (X_i - \bar{X})^2 = (10-10)^2 + (8-10)^2 + (12-10)^2 + (14-10)^2 + (6-10)^2$$
$$= 0 + 4 + 4 + 16 + 16 = 40$$
$$\sum (Y_i - \bar{Y})^2 = (85-80.4)^2 + (70-80.4)^2 + (90-80.4)^2 + (92-80.4)^2 + (65-80.4)^2$$
$$= (4.6)^2 + (-10.4)^2 + (9.6)^2 + (11.6)^2 + (-15.4)^2$$
$$= 21.16 + 108.16 + 92.16 + 134.56 + 237.16$$
$$= 593.2$$

4. **Substituição na fórmula do Coeficiente de Correlação de Pearson:**

$$r = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$
$$= \frac{147.6}{\sqrt{40 \times 593.2}}$$
$$\approx \frac{147.6}{\sqrt{23728}}$$
$$\approx \frac{147.6}{154.11}$$
$$\approx 0.957$$

Portanto, o Coeficiente de Correlação de Pearson entre as horas de estudo semanais e as pontuações no exame final é aproximadamente 0.957. Isso indica uma correlação positiva forte entre essas duas variáveis, o que sugere que quanto mais horas de estudo, maior a tende a ser a pontuação no exame final.

Similaridade Cosseno

Similaridade cosseno é uma medida de similaridade amplamente utilizada em mineração de texto, recuperação de informação e aprendizado de máquina, especialmente quando lidamos com dados de alta dimensionalidade, como vetores de palavras em processamento de linguagem natural (NPL). Ela mede o cosseno do ângulo entre dois vetores de característica e fornece uma medida de quão semelhantes são esses vetores, independentemente de sua magnitude.

Fórmula da Similaridade Cosseno:

O cosseno de dois vetores diferentes de zero pode ser derivado da fórmula do produto de ponto euclidiano:

$$AB = ||A||\,||B||\,cos\theta$$

Sejam A e B dois vetores de características, a similaridade cosseno entre eles é dada pela fórmula:

$$\text{Similaridade}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|}$$

Onde:
A . **B** é o produto escalar entre os vetores **A** e **B**.
$||\mathbf{A}||$ e $||\mathbf{B}||$ são as normas dos vetores **A** e **B**, respectivamente.

- Se a similaridade cosseno é 1, os vetores são identicos.
- Se a similaridade cosseno é 0, os vetores são ortogonais (não tem nenhuma semelhança).
- Se a similaridade cosseno é -1, os vatores são opostos.

Aplicação da Similaridade Cosseno:

- Recuperação de Informações: Usado para comparar a similaridade entre consultas e documentos em sistemas de busca de texto.
- Agrupamento de Documentos: Ajuda a agrupar documentos semelhantes com base em seus vetores de características.
- Recomendação de itens: Utilizado para calcular a similaridade entre itens e usuários em sistemas de recomendação.
- Análise de Sentimento: Ajuda a medir a similaridade entre vetores de características de texto para análise de sentimentos.
- Classificação de Texto: utilizado para comparar a similaridade entre vetores de características de texto para classificação de documentos.

Exemplo de Aplicação

Considerando que temos dois vetores de características representando dois documentos:

A = [2,1,0,1,0]
B = [1,1,1,0,1]

Para calcular a similaridade cosseno entre esses dois vetores, primeiro precisamos calcular o produto escalar e as normal dos vetores.

1. **Produto Escalar $\mathbf{A} \cdot \mathbf{B}$:**
$$\mathbf{A} \cdot \mathbf{B} = (2 \times 1) + (1 \times 1) + (0 \times 1) + (1 \times 0) + (0 \times 1)$$
$$= 2 + 1 + 0 + 0 + 0 = 3$$
2. **Normas dos Vetores $\|\mathbf{A}\|$ e $\|\mathbf{B}\|$:**
$$\|\mathbf{A}\| = \sqrt{2^2 + 1^2 + 0^2 + 1^2 + 0^2} = \sqrt{6}$$
$$\|\mathbf{B}\| = \sqrt{1^2 + 1^2 + 1^2 + 0^2 + 1^2} = \sqrt{4} = 2$$

Agora é possível calcular a similaridade cosseno usando a fórmula:

$$\text{Similaridade}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{3}{\sqrt{6} \times 2}$$

$$= \frac{3}{2\sqrt{6}}$$

$$\approx 0.612$$

Portanto, a similaridade cosseno entre os vetores **A** e **B** é aproximadamente 0.612. Isso indica que os dois vetores têm uma boa semelhança, mas não são idênticos.

2) Considere a função $E(\mathbf{w})$ onde $\mathbf{w}=[w_1, w_2, \ldots, w_n]^t$ é um vetor com múltiplas variáveis. Usando a expansão em série de Taylor a função pode ser expressa como $E(\mathbf{w}(n)+\Delta\mathbf{w}(n)) = E(\mathbf{w}(n)) + \mathbf{g}^t(\mathbf{w}(n))\Delta\mathbf{w}(n) + \frac{1}{2}\Delta\mathbf{w}^t(n)\mathbf{H}(\mathbf{w}(n))\Delta\mathbf{w}(n) + O(\|\Delta\mathbf{w}\|^3)$, onde $\mathbf{g}(\mathbf{w}(n))$ é o vetor gradiente local definido por $\mathbf{g}(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ e $\mathbf{H}(\mathbf{w})$ é matriz Hessiana, definida por $\mathbf{H}(\mathbf{w}) = \frac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}^2}$.

Demonstre com base na expansão em série de Taylor:

a-) que o método do gradiente da descida mais íngreme é dado por

$\mathbf{w}(n+1) = \mathbf{w}(n) - \eta\mathbf{g}(\mathbf{w}(n))$.

Resposta:

O método do gradiente de descida mais ingreme é um algoritmo de otimização usado para encontrar o mínimo de uma função. Ele utiliza iterativamente a variável de otimização na direção oposta ao gradiente da função objetivo, multiplicando por uma taxa de aprendizado.

$$w(n+1)=w(n)-\eta\nabla f(w(n))$$

Onde:

η é a taxa de aprendizado

∇f (w(n)) é o gradiente de f avaliado em w(n).

Usando a série de Taylor para f (w(n+1)) em torno de w (n), podemos escrever:

$$f(w+\Delta w)\approx f(w)+\nabla F(w)^T(\Delta w)+\frac{1}{2}(\Delta w)^T\nabla^2 f(w)(\Delta w)+O(\|w\|)^3$$

$$f(w(n+1))\approx f(w(n))+\nabla f(w(n)^T)(w(n+1)-w(n))+\frac{1}{2}(w(n+1)-w(n))^T\nabla^2 f(w(n))(w(n+1)-w(n))$$

Sabemos que queremos minimizar f (w), então buscamos um w(n+1) tal que f (w(n+1)) < f (w(n)), o que significa que queremos mover w na direção oposta ao gradiente. Vamos considerar apenas o primeiro termo da série de Taylor, que é uma aproximação de primeira ordem:

$$f(w(n+1))\approx f(w(n))+\nabla f(w(n))^T(w(n+1)-w(n))$$

Para minimizar f (w), queremos que w (n+1) seja tal que f (w(n+1)) < f(w(n)). Isso significa que queremos um w(n+1) tal que $\nabla f(w(n))^T (w(n+1)) - w(n) < 0$, ou seja, o produto interno entre o gradiente e a diferença de vetores deve ser negativo. Expandindo w(n+1) usando a atualização do método do gradiente de descida mais ingreme, temos:

$$\nabla f(w(n))^T(w(n)-\eta\nabla f(w(n)-w(n)))$$

$$\nabla f(w(n))^T w(n)-\eta\nabla f(w(n))^T\nabla f(w(n))-\nabla f(w(n))^T w(n)$$

$$-\eta\nabla f(w(n))^T\nabla f(w(n))=-\eta(\|\nabla f(w(n))\|)^2$$

Para minimizar a função objetivo, queremos que esse produto interno seja negativo. Isso significa que a direção do gradiente dado por ∇f(w(n)) e a diferença de vetores (w(n+1) - w(n)) devem estar em direções opostas. Isso garante que estamos nos movendo na direção que reduz o valor d função custo.

b-) que o método de Newton é dado por

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

O método de Newton para encontrar o mínimo de uma função escalar f(w) é dado por:

$w(n+1) = w(n) - \alpha H^{-1} (w (n)) \nabla f(w(n))$

Onde:

w(n) é o vetor de parâmetro no n-ésimo passo.
α é um fator de aprendizado (também chamado de tamanho do passo).
∇ F(w(n)) é o gradiente da função f em relação a w no ponto w(n).
H(w(n)) é a matriz hessiana de f em relaçao a w no ponto w(n).

Aqui, estamos interessados no caso em que α = 1, que é comumente usado no método de Newton. Substituindo α = 1 na equação acima.

A partir da expressão da série de Taylor de segunda ordem aplicada a função objetivo f(w). Vamos revisar a série de Taylor de segunda ordem:

$$f(w(n+1)) \approx f(w(n)) + \nabla f(w(n)^T)(w(n+1) - w(n)) + \frac{1}{2}(w(n+1) - w(n))^T \nabla^2 f(w(n))(w(n+1) - w(n))$$

Onde:
f (w) é a função objetivo que queremos minimizar.
∇f (w) é o gradiente de f no ponto w.
∇^2f (w) ou H(w) é a matriz hessiana de f no ponto w.

Para encontrar um mínimo locla da função, queremos encontrar o ponto w onde f(w) é mínimo. Isso ocorre quand o gradiente é zero: ∇f(w) = 0.

Uma vez que estamos procurando um mínimo, podemos ignorar o termo de primeira ordem ∇f (w(n)) (w(n+1) – w(n)), pois ∇f (w(n)) é zero no mínimo. Assim, a expressão se simplifica para:

$$f(w(n+1)) = f(w(n)) + \frac{1}{2}(w(n+1) - w(n))^T Hf(w(n))(w(n+1) - w(n))$$

Se considerarmos Δw = w(n+1) – w(n), podemos escrever a expressão como:

$$f(w(n) + \Delta w) = f(w(n)) + \frac{1}{2} \Delta w^T Hf(w(n)) \Delta w$$

Agora, queremos minimizar f(w(n)+Δw) em relação a Δw, então podemos tomar o gradiente dessa expressão em relação a Δw e igualála a zero:

$$\frac{\partial}{\partial w}(f(w(n)+\frac{1}{2}\Delta w^T Hf(w(n))\Delta w))=0$$

Agora podemos resolver essa equação para Δw. Essa derivada nos leva ao ponto onde a função f é minimizada em relação a w(n). Após resolver essa equação, obtemos:

$$f(w(n))+\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta w_i H_{ij}(w(n))\Delta w_j$$

$$\frac{\partial}{\partial w_k}(f(w(n))+\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta w_i H_{ij}(w(n))\Delta w_j)$$

$$\frac{\partial}{\partial w_k}(\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta w_i H_{ij}(w(n))\Delta w_j)$$

$$\frac{1}{2}\sum_{i=1}^{n}(\frac{\partial}{\partial w_k}(\Delta w_i H_{ij}(w(n))\Delta w_j)+\frac{\partial}{\partial w_k}(\Delta w_i H_{ij}(w(n))\Delta w_i))$$

$$\frac{1}{2}\sum_{i=1}^{n}(H_{ij}(w(n))\Delta w_j+\Delta w_i\frac{\partial H_{ij}(w(n))}{\partial w_k}\Delta w_j+H_{ij}(w(n))+\Delta w_j\frac{\partial H_{ji}(w(n))}{\partial w_k}\Delta w_i)$$

$$\frac{1}{2}(\sum_{i=1}^{n}H_{ik}(w(n))\Delta w_i+\sum_{i=1}^{n}H_{ki}(w(n))\Delta w_k)$$

$$\frac{1}{2}\sum_{i=1}^{n}(H_{ik}(w(n))+H_{ki}(w(n))\Delta w_i)$$

Agora igualamos a zero:

$$\frac{1}{2}\sum_{i=1}^{n}(H_{ik}(w(n))+H_{ki}(w(n))\Delta w_i)=0$$

$$\frac{1}{2}(\sum_{i=1}^{n}H_{ik}(w(n))\Delta w_i+\sum_{i=1}^{n}H_{ki}(w(n))\Delta w_i)=0$$

$$\frac{1}{2}*2(\sum_{i=1}^{n}H_{ik}(w(n))\Delta w_i)=0$$

$$(\sum_{i=1}^{n}H_{ik}(w(n))\Delta w_i)=0$$

$$(H_{ik}(w(n))\Delta w_k)=0$$

$H_{ij}(w(n))$ = para i ≠ k. Portanto, a matriz Hessiana Hf(w(n)) é diagonal. Então, podemos escrever:

- Hf(w(n)) = diagonal (H_{11}, H_{22}, H_{nn})

Portanto, a expressão do gradiente se torna:

$$\frac{\partial}{\partial w_k}(f(w(n))+\frac{1}{2}\sum_{i=1}^{n} H_{ij}(w(n))(\Delta w_i)^2)=0$$

Após resolvermos essa equação teremos:

$\Delta w = - H^{-1} (w(n)) \nabla f (w(n))$

Portanto, substituindo Δw de volta na expressão original, obtemos:

$w(n+1) = w(n) – H^{-1} (w(n)) \nabla f (w(n))$

c-) que o passo ótimo é dado por $\mathbf{g(w)g^t(w)/g^t(w)H(w)g(w)}$, assumindo a matriz **H** definada positiva

Resposta:
Para encontrar o passo ótimo α que minimiza a função f (w + αp), onde p é a direção do vetor, usamos a direção de Newton. Vamos começar a expansão de segunda ordem da série de Taylor para a função f (w + αp) em torno de w, onde p é a direção do vetor e α é o passo:

$$f(w+\alpha p)\approx f(w)+\nabla f(w)^T \alpha p+\frac{1}{2}\alpha^2 p^T H(w) p$$

Queremos encontrar o valor de α que minimiza essa expressão. Para isso, vamos calcular a derivada em relação a α e igualá-la a zero:

$$\frac{\partial}{\partial w_\alpha} f(w+\alpha p)=\nabla f(w+\alpha p)^T p+\alpha p^T H(w+\alpha p) p$$

Vamos reorganizar esta equação para isolar α:

$$\alpha p^T H(w+\alpha p) p=-\nabla f(w+\alpha p)^T p$$

Agora, para simplificar a notação, vamos denotar g(w) = ∇f(w). Isso nos dá:

$$\alpha p^T H(w+\alpha p) p=-g(w+\alpha p)^T p$$

Como H(w) é definida positiva, então H(w+αp) tambem é definida positiva. Isso imploca que o termo p^T H(w+αp)p é sempre positivo. Assim temos:

$$\alpha=-(\frac{g(w+\alpha p) p}{p^T H(w+\alpha p) p}) ou -(\frac{g(w)^T p}{p^T H(w) p})$$

O sinal negativo é incorporado na interpretação do passo ótimo, pois a direção do vetor é escolhido de a forma a minimizar a função objetivo, e portanto o movimento ao longo dessa direção é na direção oposta ao gradiente.

d-) Qual será o valor do passo sob as condições acima se vetor gradiente estiver alinhado com a autovetor correspondente ao autovalor máximo.

Resposta:
Se o vetor gradiente estiver alinhado com o autovetor correspondente ao autovalor máximo da matriz Hessiana H(w), isso significa que a direção do gradiente e a direção da maior curvatura da dunção são as mesmas. Nesse caso, a função cresce mais rapidamente na direção do gradiente.

$\alpha = - g'(\alpha)/g''(\alpha)$.

Se o vetor gradiente estiver alinhado como o autovetor correspondente ao autovalor máximo fr H(w), isso significa que:

$\nabla f(w) = \lambda_{max} v_{max}$, onde λ_{max} é o autovalor máximo e v_{max} é o autovetor correspondente.

Nesse caso, a segunda derivada $g''(\alpha)$ será igual ao autovalor máximo λ_{max}
Logo o valor de passo ótimo α será:

$$g''(a) = p^T \nabla^2 f(w) p = p^T H(w) p = \lambda_{max}$$

$$\alpha = -\left(\frac{\nabla f(w) p}{p^T H(w) p}\right) = \eta = \frac{\alpha^2 (V_{max})^2}{\alpha^2 \lambda_{max} (V_{max})^2}$$

$$\eta = \frac{1}{\lambda_{max}}$$

3-) A matriz Hessiana pode ser expressa aproximadamente como $\mathbf{H}(n) = \sum_{k=1}^{n} \mathbf{g}(k)\mathbf{g}^t(k)$, onde $\mathbf{g}(\mathbf{w}(n))$ é o vetor gradiente local definido por $\mathbf{g}(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$, sendo E (w) a função custo.
a-) Demonstre que $\mathbf{H}(n) = \mathbf{H}(n-1) + \mathbf{g}(n)\mathbf{g}^t(n)$

Esta equação $H(n)=H(n-1)+g(n)g^T(n)$ é uma forma de atualização iterativa da matriz Hessiana H em algoritmos de otimização, como o método de Newton ou suas variantes. Essa atualização é feita para melhorar locais sobre a curvatura da função de custo.
A expressão $g(n)g^T(n)$ representa o produto interno do vetor gradiente com seu transposto, resultando em uma matriz simétrica em cada direção.
Portanto, a equação $H(n)=H(n-1)+g(n)g^T(n)$ significa que a matriz Hessiana na iteração *n* é atualizada adicionando o produto externo do vetor gradiente na iteração *n*consigo mesmo transposto à matriz Hessiana da iteração anterior *(n-1)*.

Como a matriz Hessiana na iteração n é $H(n)=\frac{\partial^2 E}{\partial w \partial w^T}$ onde E é a função de custo e w é o vetor de parâmetros.

O vetor gradiente local $g(n)$ é definido como: $g(n)=\frac{\partial E}{\partial w}$

Assim, produto externo $g(n)g^T(n)$ resultante em uma matriz: $g(n)g^T(n)=(\frac{\partial E}{\partial w})(\frac{\partial E}{\partial w})^T$

$$H(n)=\begin{bmatrix} H_{11} & H_{12} & \cdots & H_{1m} \\ H_{21} & H_{22} & \cdots & H_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ H_{m1} & H_{m2} & \cdots & H_{mm} \end{bmatrix} =$$

$$H(n-1)=\begin{bmatrix} H_{11}^{(n-1)} & H_{12}^{(n-1)} & \cdots & H_{1m}^{(n-1)} \\ H_{21}^{(n-1)} & H_{22}^{(n-1)} & \cdots & H_{2m}^{(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ H_{m1}^{(n-1)} & H_{m2}^{(n-1)} & \cdots & H_{mm}^{(n-1)} \end{bmatrix} + g(n)g^T(n) = \begin{bmatrix} (g(n)_1)^2 & g(n)_1 g(n)_2 & \cdots & g(n)_1 g(n)_m \\ g(n)_2 g(n)_1 & (g(n)_2)^2 & \cdots & g(n)_2 g(n)_m \\ \vdots & \vdots & \ddots & \vdots \\ g(n)_m g(n)_1 & g(n)_m g(n)_2 & \cdots & (g(n)_m)^2 \end{bmatrix}$$

b-) Usando o lema da inversão matricial tal que se $\mathbf{A}=\mathbf{B}^{-1}+\mathbf{CDC}^t$ então $\mathbf{A}^{-1}=\mathbf{B}-\mathbf{BC}(\mathbf{D}+\mathbf{C}^t\mathbf{BC})^{-1}\mathbf{C}^t\mathbf{B}$, sendo $\mathbf{A}$ e $\mathbf{B}$ matrizes definidas positivas, $\mathbf{C}$ e $\mathbf{D}$ são duas outras matrizes, demonstre que

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

Isto é uma forma iterativa para gerar a matriz Hessiana com base na informação do gradiente, fazendo $\mathbf{H}(0)=\mathbf{I}$.

Resposta:

Para demonstrar a relação iterativa para gerar a matriz Hessiana $H^{-1}(n)$ com base na informação do gradiente, vamos começas com a equação dada:

$$A^{-1}=B-BC(D+C^tBC)^{-1}C^tB$$

Agora, vamos associar os termos desta equação com os termos da expressão:

A^{-1} *corresponde a* $H^{-1}(n)$
B *corresponde a* $H^{-1}(n)$
C *e* D *serão associados com outras matrizes* G *e* G^t, *respectivamente*
E vamos definir A *como* $1+G^tH^{-1}(n-1)G$

Substituindo esse valores na equação temos:

$$H^{-1}(n)=H^{-1}(n-1)-H^{-1}(n-1)G(D+G^tH-1(n-1)G)^{-1}G^tH^{-1}(n-1)$$

Agora comparando esse equação:

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

Vemos que:

$D+G^t H^{-1}(n-1)G=\frac{1}{1+g^t H^{-1}(n-1)g(n)}$ e G é equivalente a *g(n)*

4-) Dada a função

$$E(w_1, w_2, w_3) = w_1^2 - w_2^2 + 2w_3^2 - 0.1w_1w_2 + 0.5w_1w_3 + 0.08w_2w_3 - 0.2w_1 + 0.3w_2 - 0.7w_3 +$$

a) represente a mesma na forma matricial,

$E(\mathbf{w}) = \mathbf{w}^t\mathbf{A}\mathbf{w} + \mathbf{b}^t\mathbf{w} + a$, onde $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3]^t$, onde $a \in \mathbf{R}, \mathbf{b} \in \mathbf{R}^3$ e $\mathbf{A} \in \mathbf{R}^{3\times3}$.

Para representar a função E(w_1, w_2, w_3) na forma matricial, podemos utilizar a notação matricial e vetorial da seguinte forma.

E(w) = w^T Aw + b^Tw + a

Onde:

w = [w_1, w_2, w_3]^T é o vetor de variáveis,
A é uma matriz 3x3,
b é um vetor 3x1, a é um escalar.

$$A = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \frac{\partial^2 E}{\partial w_1 \partial w_3} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \frac{\partial^2 E}{\partial w_2 \partial w_3} \\ \frac{\partial^2 E}{\partial w_3 \partial w_1} & \frac{\partial^2 E}{\partial w_3 \partial w_2} & \frac{\partial^2 E}{\partial w_3^2} \end{bmatrix}$$

1. Primeira derivada parcial em relação a w_1:

$\frac{\partial E}{\partial w_1}=2w_1-0.1w_2+0.5w_3-0.2$

2. Segunda derivada parcial em relação a w_2:

$\frac{\partial^2 E}{\partial w_1^2}=2$

e assim por diante....

$\frac{\partial E}{\partial w_2}=-2w_2-0.1w_1+0.08w_3+0.3$

$\frac{\partial^2 E}{\partial w_2^2}=-2$

$\frac{\partial E}{\partial w_3}=4w_3+0.5w_1+0.08w_2-0.7$

$$\frac{\partial^2 E}{\partial w_3{}^2}=4$$

$$\frac{\partial^2 E}{\partial w_1 \partial w_3}=-0.05$$
$$\frac{\partial^2 E}{\partial w_2 \partial w_3}=0.25$$

$$E(w)=[w_1 w_2 w_3]\begin{bmatrix}2 & -0.1 & 0.5\\ -0.1 & -2 & 0.08\\ 0.5 & 0.08 & 4\end{bmatrix}+\begin{bmatrix}w_1\\ w_2\\ w_3\end{bmatrix}+[-0.2\ 0.3\ 0.4]\begin{bmatrix}w_1\\ w_2\\ w_3\end{bmatrix}+0.4$$

b) determine o seu gradiente, $\mathbf{g}(\mathbf{w})=\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$

Para a função
E = [w_1, w_2, w_3], o gradiente g(w) é dado por:

$$g(w)=\begin{bmatrix}\frac{\partial E}{\partial w_1}\\ \frac{\partial E}{\partial w_2}\\ \frac{\partial E}{\partial w_3}\end{bmatrix}$$

$$\frac{\partial E}{\partial w_1}=2w_1-0.1w_2+0.5w_3-0.2$$
$$\frac{\partial E}{\partial w_2}=-2w_2-0.1w_1+0.08w_3+0.3$$
$$\frac{\partial E}{\partial w_3}=4w_3+0.5w_1+0.08w_2-0.7$$

$$g(w)=\begin{bmatrix}2w_1-0.1w_2+0.5w_3-0.2\\ -2w_2-0.1w_1+0.08w_3+0.3\\ 4w_3+0.5w_1+0.08w_2-0.7\end{bmatrix}$$

c) determine a sua matriz Hessiana, $\mathbf{H}(\mathbf{w})=\frac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}^2}$

$$A=\begin{bmatrix}\frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \frac{\partial^2 E}{\partial w_1 \partial w_3}\\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \frac{\partial^2 E}{\partial w_2 \partial w_3}\\ \frac{\partial^2 E}{\partial w_3 \partial w_1} & \frac{\partial^2 E}{\partial w_3 \partial w_2} & \frac{\partial^2 E}{\partial w_3^2}\end{bmatrix}$$

$$\frac{\partial^2 E}{\partial w_1{}^2}=2 \qquad \frac{\partial^2 E}{\partial w_1 \partial w_3}=0.5 \qquad \frac{\partial^2 E}{\partial w_1 \partial w_2}=-0.1$$

$$\frac{\partial^2 E}{\partial w_2 \partial w_1} = -0.1 \qquad \frac{\partial^2 E}{\partial w_2{}^2} = -2 \qquad \frac{\partial^2 E}{\partial w_3 \partial w_2} = 0.5 \qquad \frac{\partial^2 E}{\partial w_3{}^2} = 4$$

$$\frac{\partial^2 E}{\partial w_2 \partial w_3} = 0.08 \qquad \frac{\partial^2 E}{\partial w_3 \partial w_2} = 0.08$$

d) verifique se esta função é estritamente convexa

Para verificar se a função E = [w_1, w_2, w_3] é estritamente conveza, podemos verificar se a matriz Hessiana H é definida positiva em todos os pontos do domínio da função.
Para determinar se a matriz Hessiana é definida positiva em todos os pontos, podemos usar o critério das menores principais: todos os determinantes das submatrizes principais devem der positivos.

1. Determinante da submatriz principal de ordem 1:
$det[2] = 2$

2. Determinante da submatriz principal de ordem 2.
$$det\begin{bmatrix} 2 & -0.1 \\ -0.1 & -2 \end{bmatrix} = -3.99$$
3. Determinante da submatriz principal de ordem 3
$$det\begin{bmatrix} 2 & -0.1 & 0.5 \\ -0.1 & -2 & 0.08 \\ 0.5 & 0.08 & 4 \end{bmatrix} = -16.09$$

Como temos determinantes das submatrizes principais de H com valores negativos, podemos concluir que a matriz Hessiana H não é definida positiva em todos os pontos do domínio da função. Portanto, a função não é estritamente convexa.

e) Considerando que em um determinado ponto **w o** gradiente da função E(**w**) é igual zero,isto é, **g(w)=0. Isto não é suficiente para indicar que este ponto corresponde a um ponto de mínimo, máximo ou um ponto de sela. Com base** nos autovalores da matriz Hessiana $\mathbf{H(w)} = \frac{\partial^2 E(\mathbf{w})}{\partial \mathbf{w}^2}$ apresente a condição para este ponto ser: (i) um ponto de mínimo, (ii) um ponto de máximo (iii) um ponto de sela da função.

Para determinar se um ponto crítico w^* (onde o gradiente da função é zero, $g(w^*) = 0$) corresponde a um ponto de mínimo, máximo ou ponto de sela da função, podemos utilizar os autovalores da matriz Hessiana $H(w^*)$.

Se todos os autovalores de $H(w^*)$ forem positivos, então w^* é um ponto de mínimo. Se todos os autovalores forem negativos, então w^* é um ponto de máximo. Se houver tanto autovalores positivos quanto negativos, então w^* é um ponto de sela.

Portanto, após encontrar o ponto $H(w^*)$ onde $g(w^*) = 0$, podemos calcular os autovalores da matriz Hessiana $H(w^*)$ e então determinar o tipo de ponto.

Se os autovalores de $H(w^*)$ forem λ_1, $\lambda_2 \lambda_3$, então:

1. Se λ_1, $\lambda_2 \lambda_3$, > 0 w^* é um ponto de mínimo.

2. Se λ_1, $\lambda_2 \lambda_3$, < 0 w^* é um ponto de máximo.

3. Se houver autovalores positivos e negativos, w^* é um ponto de sela.

> 5-) Apresente um estudo dos seguintes algoritmos de otimização: **Gradiente Estocástico (SGM), AdaGrad, RMSProp e Adam.** Estes métodos ou otimizadores são utilizados no processo de aprendizagem de redes neurais/deep learning.

1. Gradiente Estocástico (SGD):

O método do gradiente (ou método do máximo declive) é um método numérico usado em otimização. Para encontrar um mínimo (local) de uma função e usa um esquema iterativo, onde em cada passo se toma a direção (negativa) do gradiente, que corresponde à direção de declive máximo.

Começando com um vetor inicial X_0 visando alcançar um ponto de mínimo de F, consideramos a sucessão definida por X_0, X_1, X_3, …, onde a pesquisa linear é dada pela direção de descida d_n.

É utilizado para encontrar o mínimo de uma função de erro (ou função de perda, em inglês *loss function*. Um algoritmo de aprendizagem de máquina, normalmente se inicia em um ponto e a partir desse ponto é claculada o seu gradiente, de modo ue seja possível minimizar o erro.

Em resumo temos:

- O SGD é o algoritmo de otimização mais simples. Ele atualiza os pesos da rede neural movendo-se na direção oposta ao gradiente da função de perda em relação aos pesos.
- No entanto, o SGD pode sofrer de oscilações e lentidão para convergir, especialmente em regiões planas ou irregulares da função de perda.

2. AdaGrad (Adaptative Gradiente Algorithm):

A ideia por trás do Adagrad é usar taxas de aprendizado diferentes para cada parâmetro com base na iteração. A razão por trás da necessidade de taxas de aprendizado diferentes é que a taxa de aprendizado para parâmetros de recursos esparsos precisa ser maior em comparação com o parâmetro de recursos densos porque a frequência de ocorrência de recursos esparsos é menor.

Adagrad, proposto por Duchi et al. (2011) é um algoritmo que ajusta a taxa de aprendizagem de acordo com os a frequência dos parâmetros. A taxa de aprendizagem dos parâmetros mais frequentes é atualizada mais vezes, porém com valores menores; para os parâmetros menos frequentes, a taxa de aprendizagem é atualizada com valores e intervalos maiores. Essa característica o torna ideal para trabalhar com dados esparsos (Ruder, 2016). A atualização do vetor de parâmetros.

Adagrad é um algoritmo baseado na descida do gradiente que adapta a taxa de aprendizado η para os parâmetros, executando grandes atualizações para parâmetros infrequentes e atualizações menores para os parâmetros mais frequentes. Adagrad altera a taxa de aprendizado η a cada intervalo de tempo t para cada parâmetro θ_i se baseando nos gradientes anteriores calculados para este parâmetro de acordo com:

$$\theta_{t+1,j} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} g_{t,i}$$

Onde é uma matriz diagonal onde cada elemento i j é a soma dos quadrados dos gradientes com relação a θ_{ij} no intervalo t, g_{ij} corresponde ao gradiente da função custo no tempo t. $g_{ij} = \nabla_\theta J\theta_i$ é chamado de termo de suavização que serve para evitar a divisão por zero, sendo normalmente da ordem de 10^{-8}.

$$G_t = \begin{pmatrix} \sum_{T=0}^{t} (g_{x,1})^2 & \cdots & o \\ \vdots & \ddots & \vdots \\ o & \cdots & \sum_{T=0}^{t} (g_{x,i})^2 \end{pmatrix}$$

Onde $g_{x,i}$ é o gradiente da fução custo no tempo T com relação ao último valor de θ.

A principal vantagem do Adagrad é que não há necessidade de ajustar a taxa de aprendizagem manualmente, sendo o valor de 0.01 o mais encontrado e recomendado. Outra vantagem é que a taxa de aprendizagem é ajustada para cada parâmetro. A maior desvantagem é o acúmulo dos quadrados dos gradientes no denominador: como os valores da diagonal da matriz G_t se tornam muito grandes fazendo com que o modelo não consiga mais aprender, estagnando ou até reduzindo a acurácia para um número grande de iterações.

Em resumo, temos:

- O AdaGrad é uma extensão do SGD que adapta a taxa de aprendizado para cada parâmetro da rede com base na frequência de atualização desse parâmetro.
- Ele reduz a taxa de aprendizado para parâmetros que são atualizados com frequência alta e aumenta a taxa de aprendizado para parâmetros que são atualizados com menos frequência.
- Isso é útil para lidar com gradientes esparsos e convergir mais rapidamente em direções mais constantes da função de perda.
- No entanto, o AdaGrad pode sofrer de uma taxa de aprendizado que se torna muito pequena com o tempo, o que pode levar a uma parada prematura da convergência.

3. RMSProp (Root Mean Square Propagation):

É um algoritmo similar ao Adagrad, mas substitui a soma dos quadrados dos gradientes anteriores por uma média móvel E dos quadrados dos gradientes anteriores. Essa média móvel é definida como:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

Onde g é o gradiente de função custo, e o g_t é o gradiente da função de custo no tempo t e $E[g^2]_t$ é a média móvel dos quadrados dos gradientes no tempo t, logo temos:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_{t,i}$$

Onde η é a taxa de aprendizagem (com valor sugerido de 0.001). O uso médio móvel diminui o impacto da soma dos quadrados no denominador, evitando o problema da estagnação do Adagrad.

Em suma, temos:

- O RMSProp é uma melhoria do AdaGrad que tenta resolver o problema da diminuição excessiva da taxa de aprendizado ao longo do tempo.
- Em vez de acumular todos os gradientes passados para calcular a taxa de aprendizado adaptativa, o RMSProp calcula uma média móvel exponencial dos quadrados dos gradientes.
- Isso permite que o RMSProp mantenha uma taxa de aprendizado adaptativa eficaz ao longo do treinamento e lide melhor com problemas de diminuição excessiva da taxa de aprendizado.

4. Adam (Adaptive Moment Estimation):

Esse método utiliza as médias móveis dos gradientes e dos quadrados dos gradientes para atualizar os parâmetros. Essas médias são inicializadas como zero, e são calculadas como:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) . g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_1) . g^2_t$$

Onde m_t é o vetor das média dos gradientes no tempo t, v_t é o vetor das médias dos quadrados dos gradientes no tempo t, g_t é o gradiente da função de custo, β_1 é o peso atribuído à média dos gradientes (com valor sugerido de 0.9) e β_2 é o peso atrubuído à média dos quadrados dos quadrados dos gradientes (com valor sugerido de 0.999).

No entanto, esses valores de m_t e v_t são enviesados e tendem a mover a solução em direção ao zero, principalmente no início e quando os valores β_1 e β_2 são próximos de 1. A razão para isso é que os passos iniciais tendem a direcionar de m_t e v_t de m_{t-1} e v_{t-1} (inicalmente nulos) devido aos altos valores de β_1 e β_2. Isso torna o crescimento de m_t e v_t lento. Para evitar isso, esses valores são corrigidos de acordo com as equações:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Onde $m´_t$ é o vetor corrigido das médias dos gradientes, $v´_t$ é o vetor corrigido das médias dos quadrados dos gradientes. Atualização dos parâmetros é dada por:

$$\theta_{t+1} = \theta_{t,i} - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

Em resumo:

- O Adam combina os conceitos do RMSProp com o momentum, que ajuda a acelerar o treinamento, especialmente em regiões de superfície de erro curva.
- Ele calcula médias móveis exponenciais dos gradientes e dos quadrados dos gradientes, semelhante ao RMSProp, mas também incorpora uma técnica de momentum.
- O Adam é amplamente utilizado e geralmente é considerado uma escolha robusta para muitas tarefas de aprendizado profundo devido à sua eficácia geral e relativa facilidade de configuração.

Comparação:

- SGD é simples e computacionalmente eficiente, mas pode ser lento para convergir.
- AdaGrad e RMSProp adaptam a taxa de aprendizado, mas AdaGrad pode diminuir muito a taxa de aprendizado com o tempo, enquanto o RMSProp resolve esse problema.
- Adam combina a adaptatividade da taxa de aprendizado com momentum, geralmente proporcionando um desempenho robusto em uma variedade de cenários.

6-) O modelo de neurônio artificial de Mc-Culloch-Pitts faz uso da função de ativação para resposta do neurônio artificial. As função sigmoide (ou função logística) e a função tangente hiperbólica (tangsigmoide) são normalmente utilizadas nas redes neurais perceptrons de múltiplas camadas tradicionais (uma ou duas camadas ocultas - shallow network). A função ReLu (retificador linear) é normalmente utilizadas nas camadas ocultas das redes Deep Learning. Segue abaixo as expressões matemáticas de cada uma:

a-) $\varphi(v) = \frac{1}{1+\exp(-av)}$ (sigmoide); b-) $\varphi(v) = \frac{1-\exp(-av)}{1+\exp(-av)} = \tanh(\frac{av}{2})$ (tangente hiperbólica ou tangsigmoide) ; c-) $\varphi(v) = \max(0, v)$ (Re-Lu).

i) Faça uma análise comparativa de cada uma destas funções apresentando de forma gráfica a variação da função e da sua derivada com relação a v (potencial de ativação)

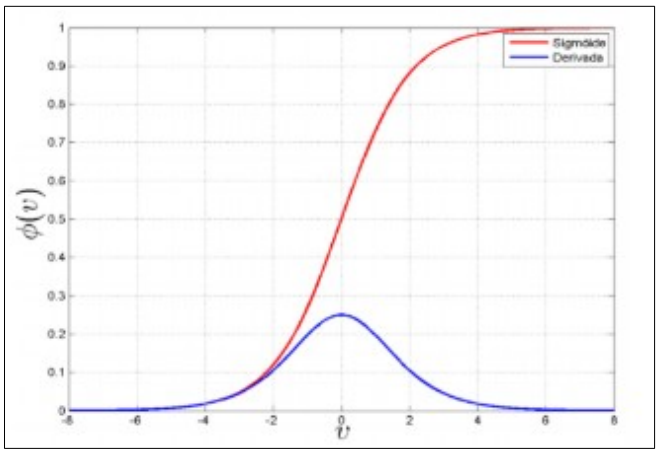
ii) Mostre que $\varphi'(v) = \frac{d\varphi(v)}{dv} = a\varphi(v)[1-\varphi(v)]$ para função sigmoide.

iii) Mostre que $\varphi'(v) = \frac{d\varphi(v)}{dv} = \frac{a}{2}[1-\varphi^2(v)]$ para função tangsigmoide.

1. Função Sigmoide (Logística):

A função sigmoide é dado por:

$$\sigma(v) = \frac{1}{1+e^{-v}}$$



E sua derivada em relação a v é:

$$\frac{d\sigma(v)}{dv} = \sigma(v)(1-\sigma(v))$$

Até pouco tempo atrás, a sigmoide era a função mais utilizada em Redes Neurais Artificiais (RNA's), por ser biologicamente mais plausível. Como neurônios biológicos funcionam de forma binária (ativado/não ativado), a função sigmoide é uma boa forma de modelar esse comportamento, já que assume valores apenas entre 0 e 1.

Contudo, observe que a derivada da função sigmoide é sempre menor que 1. Isso é um

fator problemático, pois causa o desvanecimento do produto dado pela regra da cadeia na propagação dos gradientes. Assim, não é mais recomendado utilizar a função sigmoide como não linearidade de ativação nas RNA's.

Vantagens:

- Produz saídas na faixa de [0, 1], o que é útil para problemas de classificação binária.
- É suave e diferenciável em todo o seu domínio, o que facilita o treinamento com algoritmos baseados em gradiente, como o gradiente descendente.

Desvantagens:

- Propenso a problemas de desaparecimento do gradiente, especialmente em redes muito profundas, onde a derivada da função sigmoide se torna muito próxima de zero. Isso pode levar a problemas de convergência lenta ou até mesmo estagnação durante o treinamento.
- Produz saídas não centradas em zero, o que pode resultar em desafios na convergência em determinados cenários.

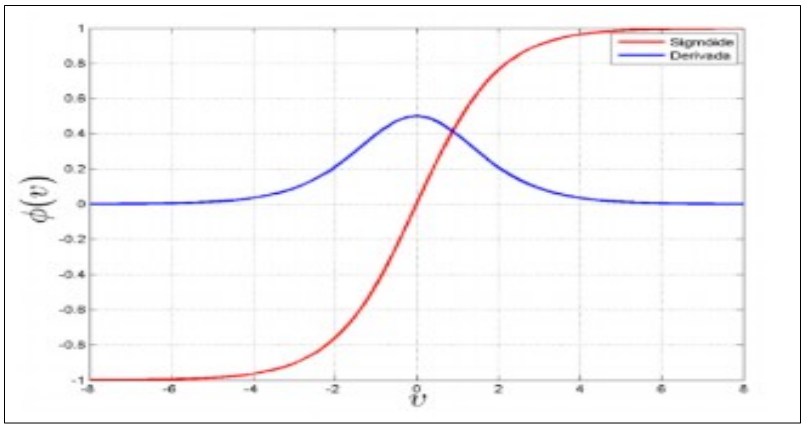
2. Função Tangente Hiperbólica (Tangsigmoide):

A tangente hiperbólica é dada por:

$$\tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}$$

E sua derivada em relação a v é:

$$\frac{d\tanh(v)}{dv} = 1 - \tanh^{2}(v)$$



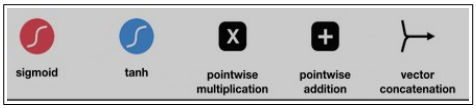
Similar a função sigmóide, a função tangente sigmoide (tanh), definida em 5.9, também tem um formato de 'S', as varia de -1 a 1, em vez de 0 a 1 como na sigmoide. A tanh se aproxima mais da identidade, sendo assim uma alternativa mais atraente do que a sigmoide para ser utilizada na ativação das camadas ocultas das RNA's. Pórem, assim como na sigmoide, apresenta o desvanecimento da propagação dos gradientes.

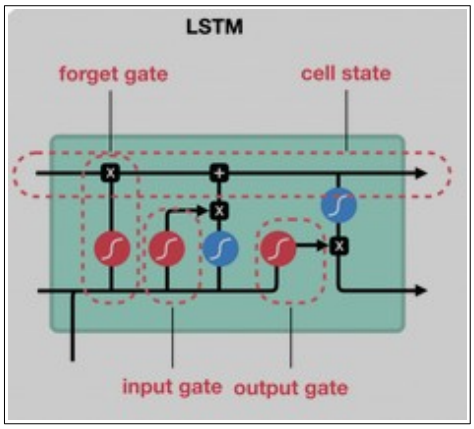
Vantagens:

- Produz saídas na faixa de [-1, 1], o que pode ser útil para problemas de classificação e regressão.
- Assim como a sigmoide, é suave e diferenciável em todo o seu domínio, facilitando o treinamento.
- As saídas são centradas em zero, o que pode facilitar o treinamento de redes neurais.

Desvantagens:

- Também pode sofrer com o problema do desaparecimento do gradiente em redes muito profundas, embora em menor grau do que a função sigmoide.
- A função tangente hiperbólica é mais computacionalmente intensiva do que a função sigmoide, o que pode tornar o treinamento mais lento.

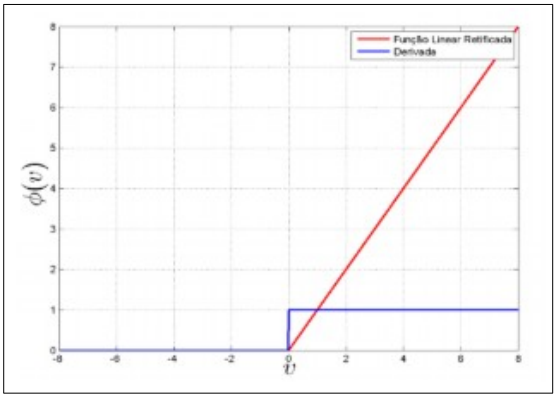




3. Função ReLU (Retificador Linear):

A funação ReLU é dado por:

$$\mathrm{ReLU}(v) = \max(0, v)$$



Sua derivada em relação a v é:

$$\mathrm{ReLU}'(v) = \begin{cases} 0, & \text{se } v < 0 \\ 1, & \text{se } v \geq 0 \end{cases}$$

A função de ativação linear retificada (ReLu), definida por , é extremamenteφ (v) = (0, v) parecida com a função identidade. A diferença é que a ReLu produz zero em metade de seu domínio. Como consequência, as derivadas se mantêm grandes enquanto a unidade estiver ativa.

Teoricamente, a derivada não está definida em zero, mas podemos implementá-la como

sendo 0 ou 1 sem maiores preocupações. Note que as derivadas são estáveis, sendo 1, quando x > 0 e zero caso contrário. Note também que a segunda derivada é zero em todo domínio. A ativação ReLu é mais eficiente do que as sigmodais.

Uma desvantagem da ativação ReLu é que unidade tendem a “morrer” durante o treinamento, um fenômeno que faz com que o neurônio passe a produzir apenas zeros. Isso ocorre quando a soma ponderada antes da aplicação de ReLu se torna negativa, fazendo com que a unidade produza zero. Nessa região, a derivada também é zero, fazendo com que os parâmetros w da unidade deixem de ser atualizados com gradiente descendente.

Vantagens:

- Computacionalmente eficiente devido à sua simplicidade. A função ReLU é implementada como uma operação de máximo, o que a torna mais rápida de calcular do que funções mais complexas como a sigmoide e a tangente hiperbólica.
- Ajuda a mitigar o problema do desaparecimento do gradiente em redes profundas, pois não satura para valores positivos de v, permitindo um treinamento mais rápido em muitos casos.
- Introduz esparsidade nas ativações, o que pode ajudar a reduzir o overfitting e melhorar a capacidade de generalização do modelo.

Desvantagens:

- Pode sofrer com o problema do "dying ReLU" ou "neurônios mortos", onde os neurônios com ativação zero não atualizam mais seus pesos durante o treinamento, levando-os a permanecer inativos para sempre. Isso pode acontecer para entradas negativas, já que a derivada da função ReLU é zero nessas regiões.
- Produz saídas não centradas em zero, o que pode levar a problemas de convergência em algumas situações, especialmente durante a fase inicial do treinamento.

7-) As funções de saída das redes neurais dependem do modelo que se busca gerar com a rede. Faça uma análise sobre a escolhas destas funções considerando os seguintes problemas: (i) Classificação de padrões com duas classes. (iii) Classificação de padrões com múltiplas classes. (iii) Problema de regressão (aproximação de funções)

Resposta:

Para cada problema mencionado, a escolha das funções de saída em redes neurais é crucial para garantir que o modelo atenda aos requisitos específicos da tarefa.

(i) Classificação de padrões com duas classes:

Para problemas de classificação binária, em que há apenas duas classes distintas, uma escolha comum para a função de saída é a função sigmóide. A função sigmóide mapeia a saída da rede para o intervalo [0, 1], o que é útil para interpretar a saída como uma probabilidade de pertencer a uma classe ou à outra. A função sigmóide é dada pela fórmula:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Onde z é a entrada para a função de ativação da camada de saída.

(ii) Classificação de padrões com múltiplas classes:

Para problemas de classificação com mais de duas classes, é comum utilizar a função de ativação Softmax na camada de saída.

A função Softmax é capaz de normalizar as saídas da rede em um vetor de probabilidades, onde cada elemento do vetor representa a probabilidade de pertencer a uma determinada classe. A função Softmax é definida como:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Onde z_i é a entrada para a função de ativação da i-ésima unidade de saída, e K é o número total de classes.

(iii) Problema de regressão (aproximação de funções):

Para problemas de regressão, em que o objetivo é prever um valor numérico contínuo, uma escolha comum para a função de saída é a função de identidade. A função de identidade simplesmente retorna a entrada sem aplicar qualquer transformação não linear. Isso é adequado para problemas de regressão, onde o objetivo é prever valores contínuos. Formalmente, a função de identidade é dada por:

$$f(x) = x$$

8-) O método da máxima verossimilhança (Max-Likelihood) é aplicado na determinação de parâmetros desconhecidos de um modelo de distribuição de probabilidades. O método é bastante utilizado para o desenvolvimento de algoritmos de aprendizagem de máquinas em particular em redes neurais. O problema considerado nesta questão envolve o modelo probabilístico de uma distribuição gaussiana multivariada . Determine os parâmetros estimados da distribuição gaussiana.

Resposta:

Para determinar os parâmetros estimados da distribuição gaussiana multivariada usando o métdo da máxima verossimilhança, vamos considerar que temos um conjunto de dados $\{x_1, x_2, \ldots, x_n\}$ onde x_i é um vetor de d dimensões.

A distribuição gaussiana multivariada é dada por:

$$f(x \vee \mu, \Sigma) = \left(\frac{1}{((2\pi)^{(d/2)}(\Sigma)^{(1/2)})}\right)\left(\exp\left(\frac{-1}{2}\right)(x-\mu)^T(\Sigma)^{(-1)}(x-\mu)\right)$$

Onde:

μ é o vetor de médias das d variáveis.

Σ é a matriz de covariância d x d.

A log-verossimilhança para o conjunto de dados é dada por:

$$\log L(\mu,\Sigma)=-(\frac{n}{2})\log(\Sigma)-(\frac{1}{2})\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{(-1)}(x_i-\mu)-(\frac{nd}{2})\log(2\pi)$$

Onde:

|Σ| é o determinante da matriz de covariância.

Para encontrar os parâmetros μ e Σ que maximizam a log-verossimilhança, deve diferenciar log L(μ, Σ) em relação a μ e Σ, e igualar as derivadas a zero.

A solução analítica para μ é a média dos dados:

$$\mu_{est}=\frac{1}{n}\sum_{i=1}^{n}X_i$$

Para Σ, a solução analítica á a matriz de covariância amostral:

$$\Sigma_{est}=\frac{1}{n}\sum_{i=n}^{n}(X_i-\mu_{est})(X_i-\mu_{est})^T$$

9-) Implementações Computacionais de Redes Neurais.

Para cada um dos problemas abaixo apresente a solução fazendo uso de redes. Apresente na solução a curva do erro de treinamento e o erro de validação:

9.1-) Defina a estrutura de uma rede perceptron de múltiplas camadas para aproximar a

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

Resposta:

Na implementação foi seguido alguns passos:

1. Preparação dos Dados: Gerar um conjunto de dados de treinamento e validação.

2. Construção do Modelo: Definido a arquitetura da rede neural.

3. Treino da rede neural usando os dados de treinamento.

4. Avaliação do desempenho da rede neural usando os dados de validação.

5. Visualização do Erro: Plote das curvas de erro de treinamento e validação, para entender o desempenho do modelo.

A função de entrada tem uma certa complexidade devido à presença de termos quadráticos, termos lineares e função logarítmica, o que torna a relação de entrada x e a saída não lineares, o que pode exigir que a rede neural capture relações não-lineares durante o aprendizado.

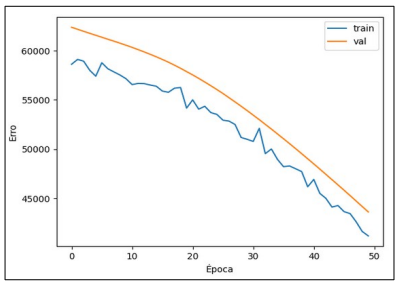
O código acima cria uma rede neural com duas camadas ocultas, cada uma com 64 neurônios. O modelo é compilado com orimizador Adam e a função de perda MSE (Erro Quadrático Médio). Os dados de entrada são gerados aleatoriamente em função do f(x) fornecido, assim como os dados de treinamento.

```python
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

# Definindo a função f(x)
def f(x):
    return 16 * x[:, 0]**2 + x[:, 0] * x[:, 1] + 8 * x[:, 1]**2 - x[:, 0] - x[:, 1] + np.log(1 + x[:, 0]**2 + x[:, 1]**2)

# Gerar dados
np.random.seed(0)
num_samples = 20000  # Dobramos o número de amostras para dividir igualmente entre treinamento e validação
x_data = np.random.rand(num_samples, 2) * 10 - 5
y_data = f(x_data)

# Dividir dados em treinamento e validação
x_train, x_val, y_train, y_val = train_test_split( *arrays: x_data, y_data, test_size=0.5, random_state=42)

# Construir modelo
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense( units: 64, activation='relu', input_shape=(2,)),  # Camada oculta com ativação ReLU e 64 neurônios
    #tf.keras.layers.Dense(8, activation='relu'),  # Outra camada oculta com ativação ReLU e 64 neurônios
    tf.keras.layers.Dense(1)  # Camada de saída
])
# Compilar modelo
model.compile(optimizer='adam', loss='mse')  # Compilando o modelo com otimizador Adam e função de perda MSE

# Treinar modelo
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_val, y_val), verbose=0)  # Treinamento do modelo

# Plotar curvas de erro
plt.plot( *args: history.history['loss'], label='train')  # Curva de erro de treinamento
plt.plot( *args: history.history['val_loss'], label='val')  # Curva de erro de validação
plt.xlabel('Época')  # Definindo o rótulo do eixo x
plt.ylabel('Erro')  # Definindo o rótulo do eixo y
plt.legend()  # Adicionando legenda ao gráfico
plt.show()  # Mostrando o gráfico
```

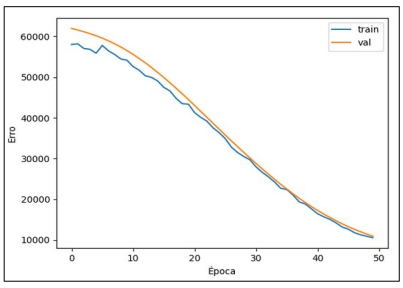
A arquitetura do modelo é a seguinte:

Camada de Entrada (2

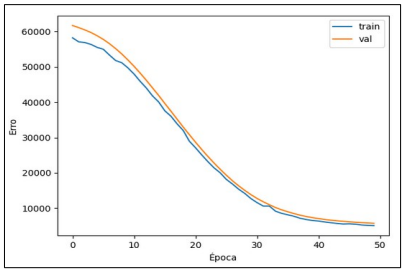
neurônios) -> Camada Oculta (64 neurônios, ReLU) -> Camada Oculta (64 neurônios, ReLU) -> Camada de Saída (1 neurônio)



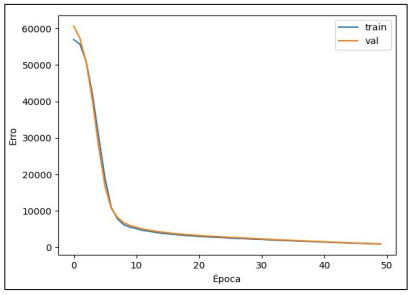
1 camada – 8 neurônios



1 camada – 16 neurônios

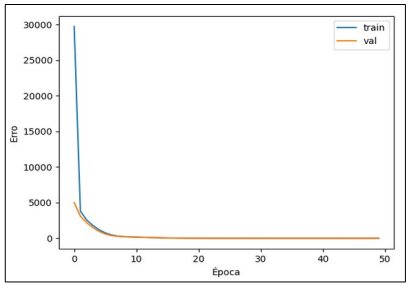


1 camada – 64 neurônios



2 camadas – 64 neurônios

2000 dados



2 camadas – 64 neurônios

20000 dados

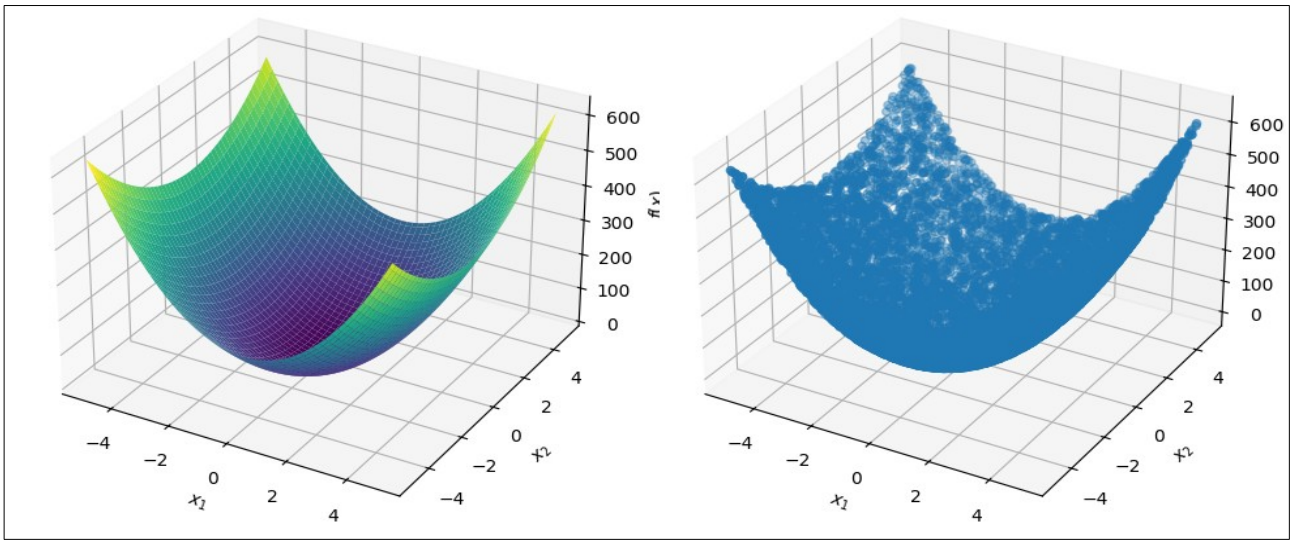


Gráfico gerado no programa

9.2-) Considere o problema das espirais. Sendo a espiral 1 uma classe e a espiral 2 outra classe. Gere as curvas das espirais usando as seguintes equações:

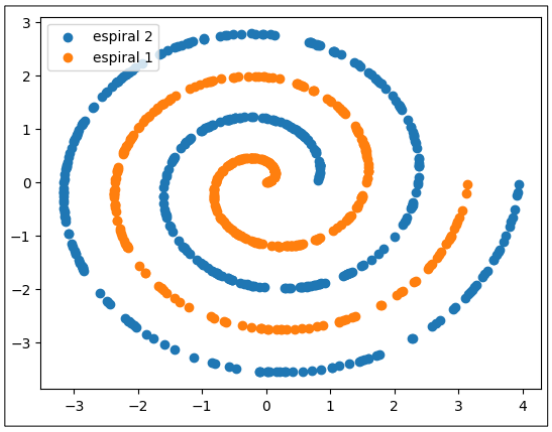
para espiral 1 $x = \frac{\theta}{4}\cos\theta \qquad y = \frac{\theta}{4}\operatorname{sen}\theta \qquad \theta \geq 0$

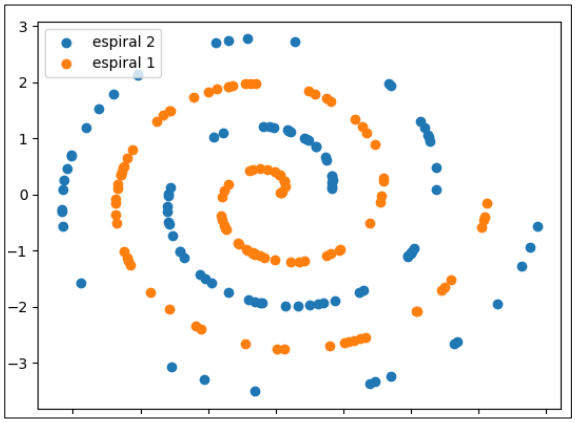
para espiral 2 $x = (\frac{\theta}{4} + 0.8)\cos\theta \qquad y = (\frac{\theta}{4} + 0.8)\operatorname{sen}\theta \qquad \theta \geq 0$

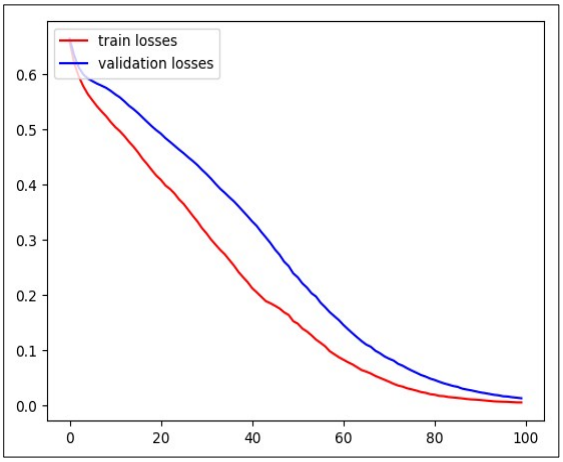
Solucione este problema utilizando uma rede perceptron de múltiplas camadas. Gere a partir das equações os dados para treinamento e teste. Determine a matriz de confusão.

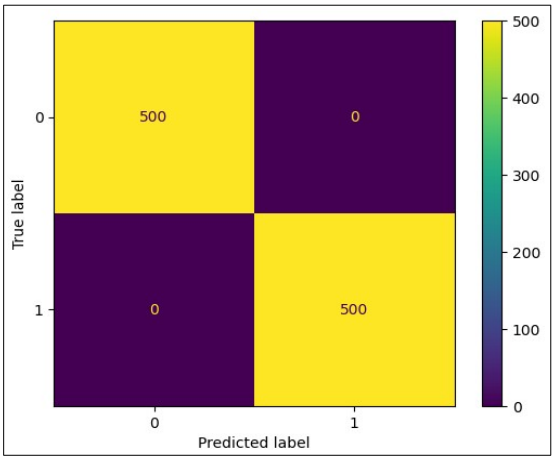
```python
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Funções para gerar os pontos das espirais
def generate_spiral1(theta):
    x = (theta / 4) * np.cos(theta)
    y = (theta / 4) * np.sin(theta)
    return x, y

def generate_spiral2(theta):
    x = ((theta / 4) + 0.8) * np.cos(theta)
    y = ((theta / 4) + 0.8) * np.sin(theta)
    return x, y

# Gerar dados para treinamento e teste
n_points = 1000
theta_train = np.linspace( start: 0, 6*np.pi, n_points)
theta_test = np.linspace( start: 0, 6*np.pi, n_points)

X_train = np.zeros((2*n_points, 2))
y_train = np.zeros(2*n_points)
X_train[:n_points, 0], X_train[:n_points, 1] = generate_spiral1(theta_train)
X_train[n_points:, 0], X_train[n_points:, 1] = generate_spiral2(theta_train)

y_train[n_points:] = 1  # Classe 1 para a espiral 2

# Embaralhar os dados de treinamento
shuffle_index = np.random.permutation(2*n_points)
X_train_shuffled = X_train[shuffle_index]
y_train_shuffled = y_train[shuffle_index]

# Treinar o modelo MLP
mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000)
mlp.fit(X_train_shuffled, y_train_shuffled)

# Gerar dados de teste
X_test = np.zeros((2*n_points, 2))
y_test = np.zeros(2*n_points)

X_test[:n_points, 0], X_test[:n_points, 1] = generate_spiral1(theta_test)
X_test[n_points:, 0], X_test[n_points:, 1] = generate_spiral2(theta_test)

y_test[n_points:] = 1  # Classe 1 para a espiral 2
```

```python
# Fazer previsões no conjunto de teste
y_pred = mlp.predict(X_test)

# Calcular a matriz de confusão
conf_matrix = confusion_matrix(y_test, y_pred)
print("Matriz de Confusão:")
print(conf_matrix)

# Plotar matriz de confusão como mapa de calor
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="viridis", cbar=False,
            xticklabels=['Espiral 1 (pred)', 'Espiral 2 (pred)'],
            yticklabels=['Espiral 1 (true)', 'Espiral 2 (true)'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Matriz de Confusão')
plt.show()

# Plotar os pontos e a fronteira de decisão
plt.figure(figsize=(10, 5))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Classificação das Espirais')
plt.show()
```





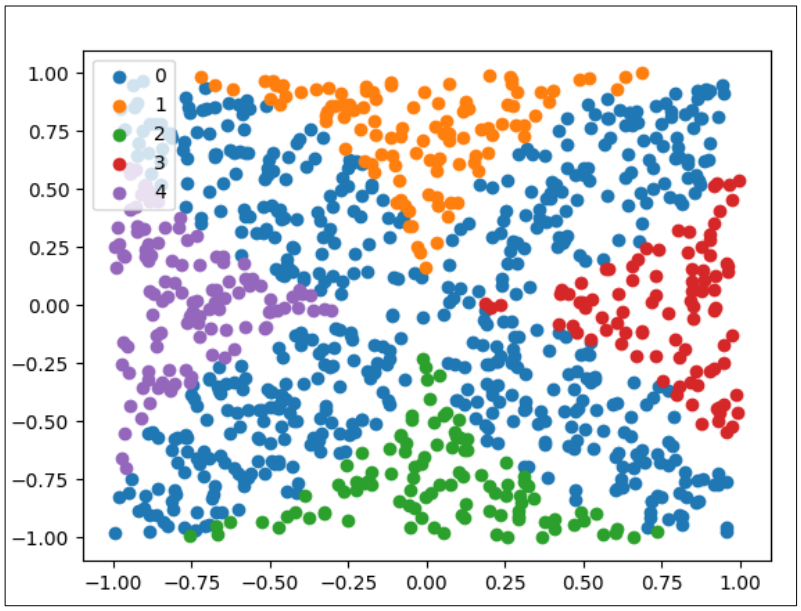


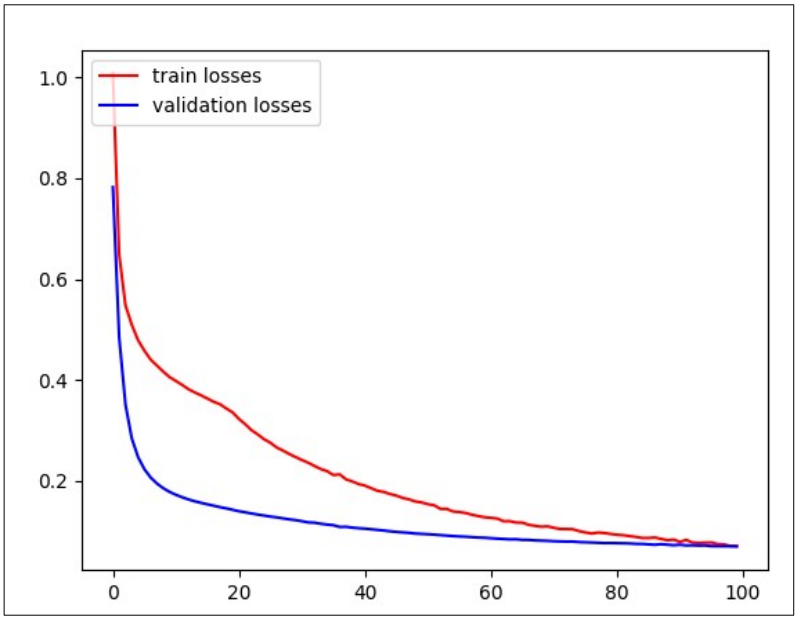


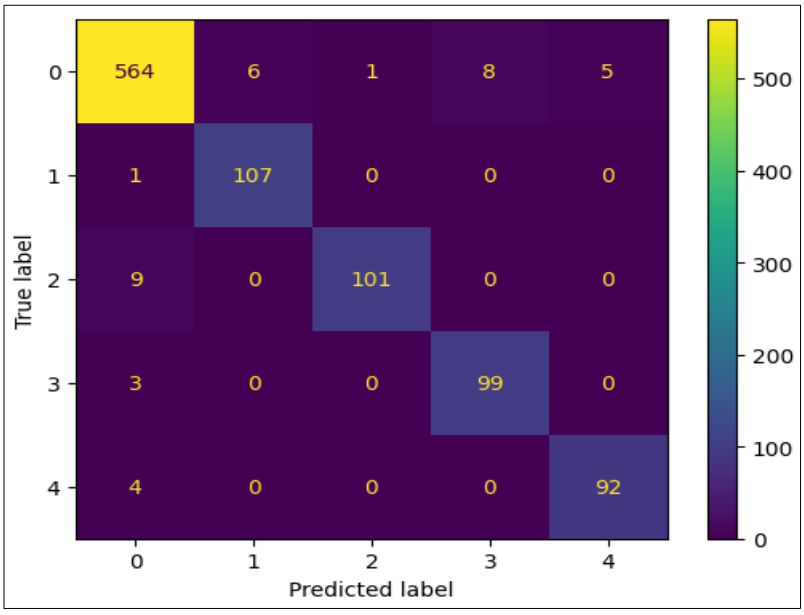
9.3-) Considere o problema de classificação de padrões bidimensionais constituído neste caso de 5 padrões. A distribuição dos padrões tem como base um quadrado centrado na origem interceptando os eixos nos pontos +1 e -1 de cada eixo. Os pontos +1 e -1 de cada eixo são centros de quatro semicírculos que se interceptam no interior do quadrado originando uma classe e as outras quatro classes nas regiões de não interseção. Após gerar aleatoriamente dados que venham formar estas distribuições de dados, selecione um conjunto de treinamento e um conjunto de validação. Treine uma rede perceptron para classificar os padrões associados a cada uma das classes. Verifique o desempenho do classificador usando o conjunto de validação e calculando a matriz de confusão.

```python
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn
from torch import optim
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from numpy.random import shuffle

data_size = 1_000
data_x = np.random.uniform(low=-1, high=1, size=(data_size, 2))

data_y = np.zeros(data_size, dtype=int)
for i in range(data_size):
    count = 0
    if np.linalg.norm(np.array([0,1]) - data_x[i]) < 1:
      count += 1
    if np.linalg.norm(np.array([0,-1]) - data_x[i]) < 1:
      count += 2
    if np.linalg.norm(np.array([1,0]) - data_x[i]) < 1:
      count += 4
    if np.linalg.norm(np.array([-1,0]) - data_x[i]) < 1:
      count += 8
    if count == 1:
      data_y[i] = 1
    if count == 2:
      data_y[i] = 2
    if count == 4:
      data_y[i] = 3
    if count == 8:
      data_y[i] = 4

# Criando um índice aleatório
random_index = torch.randperm(data_size)

# Embaralhando os dados de acordo com o índice aleatório
data_x = data_x[random_index]
data_y = data_y[random_index]

data_x = torch.Tensor(data_x)
data_y = torch.Tensor(data_y).long()

print(data_x.shape, data_y.shape)

plt.scatter(data_x[data_y==0][:,0], data_x[data_y==0][:,1], label='0')
plt.scatter(data_x[data_y==1][:,0], data_x[data_y==1][:,1], label='1')
plt.scatter(data_x[data_y==2][:,0], data_x[data_y==2][:,1], label='2')
plt.scatter(data_x[data_y==3][:,0], data_x[data_y==3][:,1], label='3')
plt.scatter(data_x[data_y==4][:,0], data_x[data_y==4][:,1], label='4')
```

```
plt.legend(loc="upper left")
plt.show()

X_train, X_test, y_train, y_test = train_test_split( *arrays: data_x, data_y, train_size=0.8, stratify=data_y)
train_size = int(data_size*0.8)
test_size = int(data_size*0.2)

model = nn.Sequential(
    nn.Linear( in_features: 2,  out_features: 128),  # input layer
    nn.ReLU(),
    nn.Linear( in_features: 128,  out_features: 128),  # hidden layer 1
    nn.ReLU(),
    nn.Linear( in_features: 128,  out_features: 5),  # output layer
).to("cpu")

loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

n_epochs = 100
batch_size = 10

train_losses = []
val_losses = []

for epoch in range(n_epochs):
    for i in range(0, train_size, batch_size):
        Xbatch = X_train[i:i+batch_size]
        y_pred_train = model(Xbatch)
        ybatch = y_train[i:i+batch_size]
        loss = loss_fn(y_pred_train, ybatch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    train_losses.append(loss.item())
    with torch.no_grad():
      y_pred_test = model(X_test)
      val = loss_fn(y_pred_test, y_test)
      val_losses.append(val)
    print(f'Finished epoch {epoch+1}, latest training loss {loss}, validation loss {val}')

plt.plot( *args: train_losses, "-r", label="train losses")
plt.plot( *args: val_losses, "-b", label="validation losses")
plt.legend(loc="upper left")
plt.show()

x_data = torch.Tensor(np.concatenate((X_train, X_test)))

with torch.no_grad():
      y_pred_test = model(x_data)
      _, y_pred_test = torch.max(y_pred_test, dim=1)

plt.scatter(x_data[y_pred_test == 0,0], x_data[y_pred_test == 0,1], label="0")
plt.scatter(x_data[y_pred_test == 1,0], x_data[y_pred_test == 1,1], label="1")
plt.scatter(x_data[y_pred_test == 2,0], x_data[y_pred_test == 2,1], label="2")
plt.scatter(x_data[y_pred_test == 3,0], x_data[y_pred_test == 3,1], label="3")
plt.scatter(x_data[y_pred_test == 4,0], x_data[y_pred_test == 4,1], label="4")
plt.legend(loc="upper left")
plt.show()

x_data = torch.Tensor(np.concatenate((X_train, X_test)))
y_data = torch.Tensor(np.concatenate((y_train, y_test)))

plt.scatter(x_data[y_data == 0,0], x_data[y_data == 0,1], label="0")
plt.scatter(x_data[y_data == 1,0], x_data[y_data == 1,1], label="1")
plt.scatter(x_data[y_data == 2,0], x_data[y_data == 2,1], label="2")
plt.scatter(x_data[y_data == 3,0], x_data[y_data == 3,1], label="3")
plt.scatter(x_data[y_data == 4,0], x_data[y_data == 4,1], label="4")
plt.legend(loc="upper left")
plt.show()

true_labels = np.concatenate((y_train, y_test))
cm_display = ConfusionMatrixDisplay(confusion_matrix(true_labels, y_pred_test)).plot()
```


0
1
2
3
4
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


train losses
validation losses
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100


564
6
1
8
5
1
107
0
0
0
9
0
101
0
0
3
0
0
99
0
4
0
0
0
92
True label
Predicted label
0
1
2
3
4
500
400
300
200
100
0

10 - Trabalho:

Apresente um estudo com uma aplicação fazendo uso de uma rede percetron de múltiplas camadas. A aplicação pode ser na área de estudo que for de seu interesse.

A questão pode ser visualizada no endereço abaixo:

http://github.com/alessandropequeno/startPhayton/blob/main/deep_q_10.ipynb

REFERÊNCIAS

Bickel, P. J. and Doksum, K.A. (2015). Mathematical Statistics. Second edition. Chapmam and Hall.

Bussab, W.O. e Morettin, P.A. (2017). Estatística Básica, 9ª Edição. São Paulo: Saraiva.

Goodfellow, I., Bengio, Y. And Courville, A. (2016). Deep Learning. The MIT Press.

McKinney, W. (2018). Python para Análise de Dados. Novatec.