lista 02 - escrita

Tags: #fleeting #literature #permanent
Description:
Theme:

ID: 20250813131453

LISTA – II RNA e LLM

LISTA – II **RNA e LLM**

1. Aplique o algoritmo de aprendizagem do perceptron para o caso do operador lógico OU.

As entradas são dadas por x_0, x_1 e x_2, com pesos w_0 (bias), w_1 e w_2, respectivamente, sendo que y é a saída que deverá, de acordo com as entradas, ser igual ao valor desejado d mostrado na tabela abaixo.

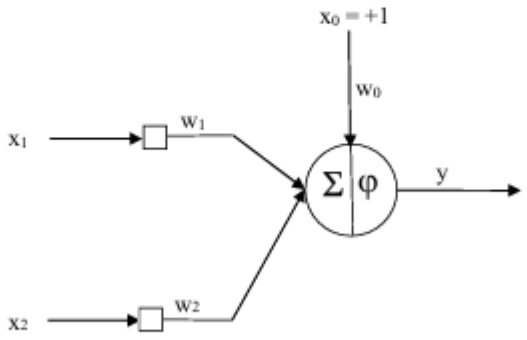
x_0	x_1	x_2	$x_1 \vee x_2$	d
1	1	1	1	1
1	1	0	1	1
1	0	1	1	1
1	0	0	0	0

A rede neural para realizar essa tarefa de aprendizagem pode ser representada como mostrado na Figura–1.
Serão atribuídos os seguintes valores iniciais para os pesos e para a taxa de aprendizagem:

$$w_0 = 0, w_1 = 0, w_2 = 0 \text{ e } \eta = 0,5 .$$

Será utilizada a função de ativação (ou função de transferência)

$$\varphi(v) = \begin{cases} 1, & se \quad v > 0 \\ 0, & se \quad v \leq 0 \end{cases}$$



Figura–1 Rede neural perceptron para aprendizagem do operador lógico OU.

Após a obtenção dos resultados, construa uma sequência de comandos em Python para realizar esse treinamento.

Nome: Vinícius de Oliveira Cardoso Rondon

- Inputs: $x_0 = 1$, x_1, x_0

- Dataset:

i	x_0	x_1	x_2	d
1	1	1	1	1
2	1	1	0	1
3	1	0	1	1
4	1	0	0	0

- Pesos e taxa de aprendizado:

$$w_0 = w_1 = w_2 = 0, \quad \eta = 0{,}5$$

- Função de ativação:

(Degrau unit.) $\varphi(v) = \begin{cases} 1, & v > 0, \\ 0, & v \leq 0, \end{cases}$ com $\boxed{v = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2 = w \cdot v}$

$$\begin{cases} \vec{x}^{(i)} = [x_0, x_1, x_2] \quad \text{com } x_0 = 1 \\ \vec{w} = [w_0, w_1, w_2]. \\ y^{(i)} = \varphi(v^{(i)}) \end{cases} \Rightarrow \begin{cases} v^{(i)} = w^T \cdot x^{(i)} = w_0 \cdot x_0^{(i)} + w_1 \cdot x_1^{(i)} + w_2 \cdot x_2^{(i)}, \\ y^{(i)} = \varphi(v^{(i)}). \end{cases}$$

- Regra de aprendizado:

$$\boxed{\vec{w} + \eta \cdot (d^{(i)} - y^{(i)}) \cdot x^{(i)} \rightarrow \vec{w'}}$$

- se $d^{(i)} = 1$ e $y^{(i)} = 0$ (falso negativo), então w' será atualizado;
- se $d^{(i)} = 0$ e $y^{(i)} = 0$ (falso positivo), então w' será atualizado;
- se $d^{(i)} = y^{(i)}$, w' não muda.

Nome: Vinicius de Lemos Cardoso Rocadom

Resolução

$y = \varphi(v)$, $v = w^T.x$, $w + \eta \underbrace{(d-y)}_{erro} x \rightarrow w'$, $\eta = 0{,}5$, $w^{(0)} = [0,0,0]$

(1) $w^{(0)} = [0,0,0]$

□ $i = 1 \rightarrow \vec{x} = [1,1,1]$, $d = 1$:

$v = 0.1 + 0.1 + 0.1 = 0 \Rightarrow y = 0$.

Erro $= d - y = 1$

$w' = [0,0,0] + 0{,}5 \cdot 1 \cdot [1,1,1] =$

$\boxed{w' = [0{,}5, 0{,}5, 0{,}5]}$

□ $i = 2 \rightarrow \vec{x} = [1,1,0]$, $d = 1$:

$v = 0{,}5.1 + 0{,}5.1 + 0{,}5.0 = 1 \Rightarrow y = 1$.

Erro $= d - y = 0$

Sem alteração

□ $i = 3 \rightarrow \vec{x} = [1,0,1]$, $d = 1$:

$v = 0{,}5.1 + 0{,}5.0 + 0{,}5.1 = 1 \Rightarrow y = 1$.

Erro $= 0$

Sem alteração

□ $i = 4 \Rightarrow \vec{x} = [1,0,0]$, $d = 0$:

$v = 0{,}5.1 + 0{,}5.0 + 0{,}5.0 = 0{,}5 \Rightarrow y = 1$.

Erro $= -1$

$w' = [0{,}5, 0{,}5, 0{,}5] + 0{,}5 \cdot -1 \cdot [1,0,0] =$

$\boxed{w' = [0, 0{,}5, 0{,}5]}$

Nome: Vinicius de Santos Cardoso Random

② $w^{(1)} = [0, 0{,}5, 0{,}5]$

□ $i = 1 \rightarrow \vec{x} = [1, 1, 1]$, $d = 1$:

$v = 0 \cdot 1 + 0{,}5 \cdot 1 + 0{,}5 \cdot 1 = 1 \Rightarrow y = 1$

Erro = 0

$\boxed{\text{Sem alteração}}$

□ $i = 2 \rightarrow \vec{x} = [1, 1, 0]$, $d = 1$:

$v = 0 \cdot 1 + 0{,}5 \cdot 1 + 0{,}5 \cdot 0 = 0{,}5 \Rightarrow y = 1$

Erro = 0

$\boxed{\text{Sem alteração}}$

□ $i = 3 \rightarrow \vec{x} = [1, 0, 1]$, $d = 1$:

$v = 0{,}1 + 0{,}5 \cdot 0 + 0{,}5 \cdot 1 = 0{,}5 \Rightarrow y = 1$

Erro = 0

$\boxed{\text{Sem alteração}}$

□ $i = 4 \rightarrow \vec{x} = [1, 0, 0]$, $d = 0$:

$v = 0{,}1 + 0{,}5 \cdot 0 + 0{,}5 \cdot 0 = 0 \Rightarrow y = 0$

Erro = 0

$\boxed{\text{Sem alteração}}$

Resultado: $w_{final} = [0, 0{,}5, 0{,}5]$

Sequencia em python

```python
class Perceptron:
    def __init__(self, epochs=20, learning_rate=0.01):
        self.epochs = epochs
        self.learning_rate = learning_rate
```

```python
        self.W = None

    def perceptron_fit(self, X_train, y_train):
        X = X_train
        y = y_train
        n_samples, n_features = X.shape


        self.W = np.random.randn(n_features, 1)

        for epoch in range(self.epochs):
            for i in range(n_samples):
                x = X[i].reshape(-1, 1)
                u = np.dot(x.T, self.W)
                y_pred = np.heaviside(u, 1)
                error = y[i] - y_pred
                self.W += self.learning_rate * error * x

        return self

    def predict(self, X):

        if len(X.shape) == 1:
            X = X.reshape(1, -1)

        # Calcula a predição
        u = np.dot(X, self.W)
        y_pred = np.heaviside(u, 0)
        return y_pred.flatten()
```