Perceptron funcional

Irving Montalvo

Aug 2, 2024

Perceptron funcional

Descrição

Um perceptron é uma unidade básica de uma rede neural artificial.

$$y = f(w \cdot x + b)$$

Componentes

1.- **Entrada** (x): Um vetor de características de entrada, $x = (x_1, x_2, \ldots, x_n)$.

2.- **Pesos** (w): Um vetor de pesos associado a cada entrada, $w = (w_1, w_2, \ldots, w_n)$.

3.- **Viés** (b): Um valor que permite ajustar a saída do perceptron, frequentemente chamado de bias.

4.- **Função de ativação** (f): Uma função que decide se o perceptron será ativado ou não.

4.1.- **Sigmoide**: Utilizada para mapear valores entre 0 e 1, adequada para problemas de classificação binária.

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

Esta função é útil para modelos onde se deseja uma probabilidade como saída.

```
--
sigmoide :: Double -> Double
sigmoide z = 1 / (1 + exp (-z))
--
```

4.2.- **tanh**: Mapeia valores entre -1 e 1, útil quando se deseja saídas centradas em torno de zero.

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

A tanh é frequentemente preferida à sigmoide porque sua saída é centrada em torno de zero, o que pode ajudar na convergência do modelo.

```haskell
--
tanh :: Double -> Double
tanh z = (exp z - exp (-z)) / (exp z + exp (-z))
--
```

Nota: É recomendável usar `Prelude.tanh`.

4.3. **ReLU**: Simples e eficiente, ideal para redes profundas.

$$f(z) = max(0, z)$$

ReLU é uma das funções de ativação mais populares devido à sua simplicidade e eficiência computacional, e é especialmente útil em redes profundas.

```haskell
--
relu :: Double -> Double
relu = max 0
--
```

4.4.- **Leaky ReLU**: Variante da ReLU que permite valores negativos pequenos.

$$f(z) = \begin{cases} z & \text{se } z > 0 \\ \alpha z & \text{senão.} \end{cases}$$

onde α é uma pequena constante.

```haskell
--
leakyRelu :: Double -> Double
leakyRelu z = if z > 0 then z else alpha * z
--
```

4.5.- **ELU**: Variante da ReLU que pode melhorar a robustez e os resultados da rede.

$$f(z) = \begin{cases} z & \text{se } z > 0 \\ \alpha(e^z - 1) & \text{senão.} \end{cases}$$

Onde α é um parâmetro que controla a inclinação para valores negativos.

```haskell
--
elu :: Double -> Double
elu z = if z > 0 then z else alpha * exp (z - 1)
--
```

4.6.- **Softmax**: Utilizada na camada de saída para problemas de classificação multi-classe.

$$f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Converte os valores da camada de saída em probabilidades.

Processo

1.- **Cálculo da soma ponderada**: Calcula-se uma soma ponderada das entradas e dos pesos, e adiciona-se o viés:

$$z_{w_i x_i} = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b$$

2.- **Aplicação da função de ativação**: A saída do perceptron é obtida aplicando-se uma função de ativação à soma ponderada:

$$\hat{y} = f(z_{w_i x_i})$$

3.- **Cálculo do Erro**: Compare a saída $\hat{y}$ com a saída desejada (ou rótulo) y.

$$Erro = y - \hat{y}$$

4.- **Atualização dos Pesos**: Atualize os pesos e o bias usando a regra de aprendizado.

$$w_i \leftarrow w_i + n \cdot (Erro) \cdot + x_i$$

$$b \leftarrow b + n \cdot (Erro)$$

5.- **Repetição**: Repita o processo para cada exemplo no conjunto de treinamento. O treinamento continua até que o erro global seja minimizado ou o número máximo de épocas (iterações) seja alcançado.

6.- **Convergência**: Verifique se a saída do perceptron está correta para todos os exemplos de treinamento ou se os pesos convergiram para valores estáveis.

Conclusão

O perceptron é um modelo simples e pode resolver apenas problemas linearmente separáveis. Para problemas mais complexos, como o **xor**, são necessárias redes neurais multicamadas.

Documentação

- Wikiperida: Perceptron
- Deep Learning Book. Capítulo 6 – O Perceptron
- (EN) What is Perceptron. The Simplest Artificial neural network
- (ES) YouTube: Funciones de activación a detalle (Redes neuronales)
- (EN) Wikipedia: e (mathematical constant)