

Treinamento de Machine Learning e Deep Learning

Do Básico ao Avançado

Salomão Machado Mafalda¹

¹Universidade Federal do Acre
PAVIC

2023



Agenda

- 1 Perceptron
- 2 Adaline
- 3 Neurônio Sigmoidal
- 4 Funções de Ativação
- 5 Backpropagation
- 6 Redes Neurais Profundas
- 7 Funções de Custo



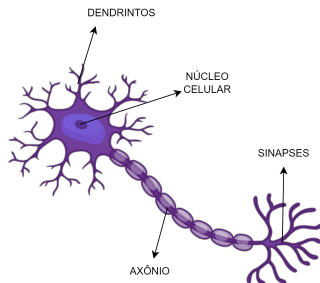


Figure: Neurônio humano

Perceptron

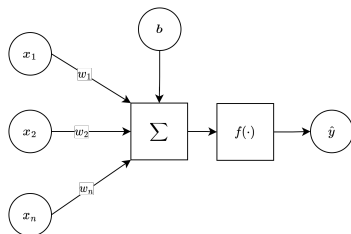
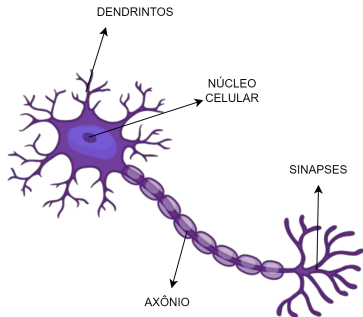


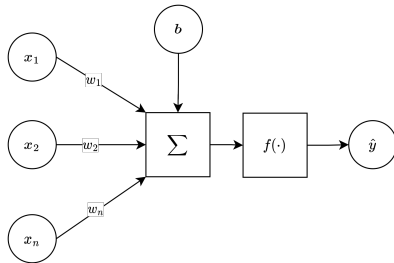
Figure: Neurônio Artificial



Perceptron



(a) Neurônio Humano



(b) Neurônio Artificial

Perceptron

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída \hat{y}

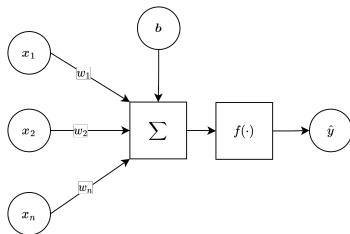


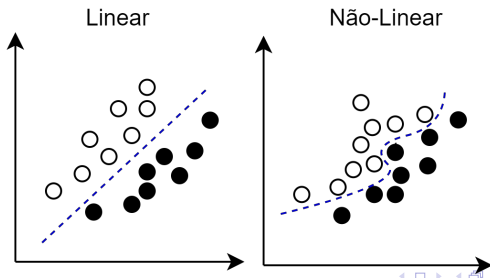
Figure: Neurônio Artificial

$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$



Perceptron

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída \hat{y}
- Classificador binário linear
- Pode ser usado para Regressão
- Perceptron Rule
- Aprendizado Online
 - Atualiza os pesos por amostra

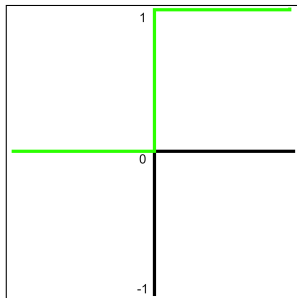


Perceptron

Função de ativação do perceptron

$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

- 0 se for negativo
- 1 se maior ou igual a 0



Perceptron Rule

O perceptron atualiza seus pesos utilizando a perceptron rule, não com o gradiente

Atualização dos pesos:

$$w_i = w_i + \lambda * (y_i - \hat{y}_i) * x_i$$

Atualização do *bias*:

$$b_i = b_i + \lambda * (y_i - \hat{y}_i)$$

Observação importante

“Quando a diferença $y_i - \hat{y}_i$ for 0 então não ocorrerá a atualização dos pesos”

Ponto de partida diferente

Com diferentes pontos de partida, o algoritmo encontra quase a mesma solução, embora com diferentes taxas de convergência.

▶ Caso 01

▶ Caso 02

Learning Rate - Taxa de aprendizagem

- $LR = 0.01$ a velocidade de convergência é muito lenta. Quando o cálculo se torna complicado, uma taxa de aprendizado muito baixa afetará a velocidade do algoritmo, mesmo nunca atingindo o destino.
- $LR = 0.5$, o algoritmo se aproxima do alvo muito rapidamente após várias iterações. No entanto, o algoritmo falha ao convergir porque o salto é muito grande, fazendo com que ele fique parado no destino.

▶ Caso 01

▶ Caso 02

Vamos ver na prática

Vamos praticar utilizando o notebook 00_perceptron



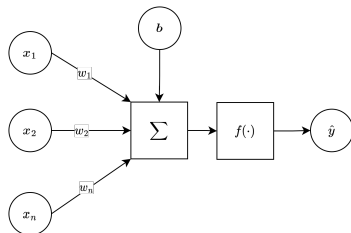


Figure: Neurônio Artificial

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída \hat{y}
- Classificador binário linear
- Pode ser usado para Regressão
- Sabe o quanto 'errou'
- Aplica-se o gradiente descendente
- Aprendizado Online

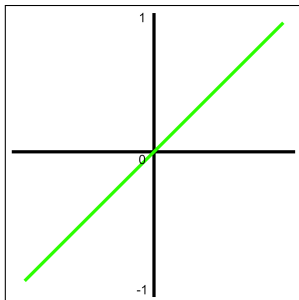
$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$



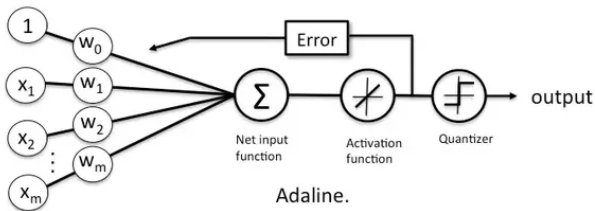
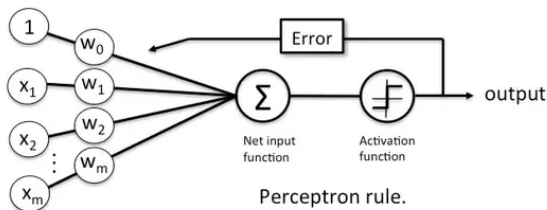
Função de ativação do Adaline

$$f(x) = x$$

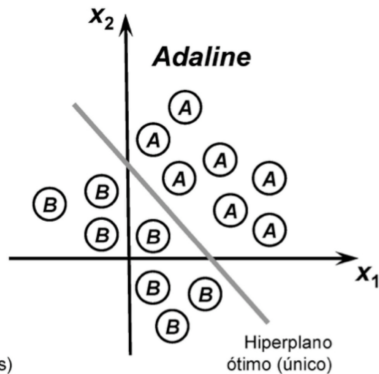
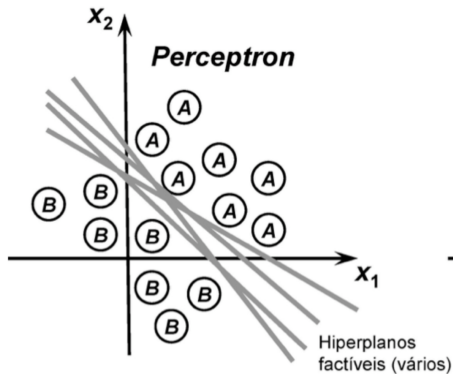
- Possibilita o cálculo da derivada



Adaline vs Perceptron



Adaline vs Perceptron



Como atualizar os pesos do Adaline

$$w_i = w_i - \lambda * (y_i - \hat{y}_i) * x_i$$

O erro predito será a saída predita menos a saída desejada multiplicados pela entrada (x)

$$J(w) = \frac{1}{2} \sum_i^N (y_i - \hat{y}_i)^2$$

$$\frac{\partial J}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_i^N (y_i - \hat{y}_i)^2 = \frac{1}{2} \sum_i^N \frac{\partial}{\partial w_i} (y_i - \hat{y}_i)^2$$

$$= \sum_i^N (y_i - \hat{y}_i) \frac{\partial}{\partial w_i} (y_i - \hat{y}_i) = \sum_i^N (y_i - \hat{y}_i) (x_i) \rightarrow \frac{\partial J}{\partial \vec{w}} = -(\vec{y} - \hat{\vec{y}}) \vec{x}$$

Vamos ver na prática

Vamos praticar utilizando o notebook 01_adaline



Neurônio Sigmoide

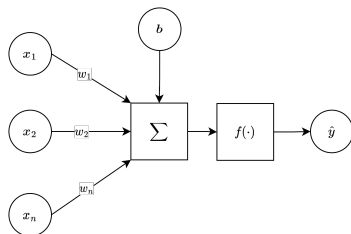


Figure: Neurônio Artificial

Neurônio Sigmoides

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída \hat{y}
- Custo: Entropia Cruzada
- Classificação binária não-linear
- Pequenas alterações nos parâmetros geram pequenas alterações nas saídas
- Sabe o quanto 'errou'
- Aplica-se o gradiente descendente

$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$

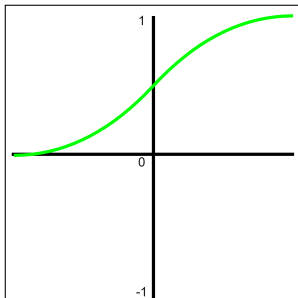


Neurônio Sigmoid

Função de ativação do Neurônio Sigmoid

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

- Possibilita o cálculo da derivada em todos pontos
- Aplicado em problemas de regressão logística



Entropia Cruzada

p_j é o valor a ser predito e t_j é o valor predito

$$L = -\frac{1}{N} \left[\sum_{j=1}^N [t_j \log(p_j) + (1 - t_j) \log(1 - p_j)] \right]$$



Neurônio Sigmoid

Vamos tomar:

p_j	t_j	Erro	L
0	0	$0 - 0 = 0$	0
0	1	$0 - 1 = -1$	∞
1	0	$1 - 0 = 1$	∞
1	1	$1 - 1 = 0$	0

Entropia Cruzada

Para entrada 0,0 e saída predita 0 e a saída desejada for 0

$$L = -\frac{1}{N} \left[\sum_{j=1}^N [0 \log(0) + (1 - 0) \log(1 - 0)] \right]$$



Neurônio Sigmoid

Vamos tomar:

p_j	t_j	Erro	L
0	0	$0 - 0 = 0$	0
0	1	$0 - 1 = -1$	∞
1	0	$1 - 0 = 1$	∞
1	1	$1 - 1 = 0$	0

Entropia Cruzada

Para entrada 0, 1 e saída predita 0 e a saída desejada for 1

$$L = -\frac{1}{N} \left[\sum_{j=1}^N [0 \log(1) + (1 - 0) \log(1 - 1)] \right]$$



Neurônio Sigmoid

Vamos tomar:

p_j	t_j	Erro	L
0	0	$0 - 0 = 0$	0
0	1	$0 - 1 = -1$	∞
1	0	$1 - 0 = 1$	∞
1	1	$1 - 1 = 0$	0

Entropia Cruzada

Para entrada 1, 0 e saída predita 1 e a saída desejada for 0

$$L = -\frac{1}{N} \left[\sum_{j=1}^N [1 \log(0) + (1 - 1) \log(1 - 0)] \right]$$



Neurônio Sigmoid

Vamos tomar:

p_j	t_j	Erro	L
0	0	$0 - 0 = 0$	0
0	1	$0 - 1 = -1$	∞
1	0	$1 - 0 = 1$	∞
1	1	$1 - 1 = 0$	0

Entropia Cruzada

Para entrada 1, 1 e saída predita 1 e a saída desejada for 1

$$L = -\frac{1}{N} \left[\sum_{j=1}^N [1 \log(1) + (1 - 1) \log(1 - 1)] \right]$$



Neurônio Sigmoid

Vamos ver na prática

Vamos praticar utilizando o notebook 02_neuronio_sigmoid



Funções de Ativação

- Localizada a saída de cada neurônio
- Usada para mapear entradas em novas saídas
- Pode alterar o range ex: $[-100 \ 100]$ para $[1 \ 0]$

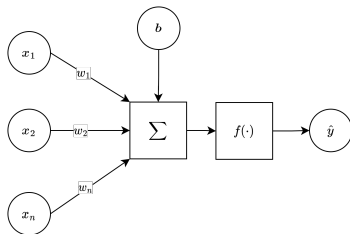


Figure: Neurônio Artificial

$$\hat{y} = f\left(\sum_i w_i x_i + b\right)$$



Linear

- $y \in [-\infty, +\infty]$
- Função de ativação simples
- Comumente usada em regressão
- Baixa complexidade
- Baixo poder de aprendizagem

Função:

$$f(x) = x$$

Derivada:

$$\frac{\partial y}{\partial x} = 1$$



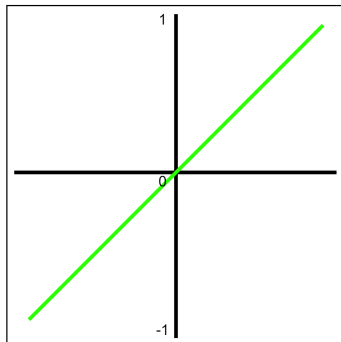
Atenção

Note este caso: Porque construir modelos apenas com Lineares?

input: “10 \rightarrow 100 \rightarrow 200 \rightarrow 10”

pesos: “10 \rightarrow 2 \rightarrow 00.5 = 10 \times 10 = 10”

Podemos substituir todos pesos por um só



Sigmoid

- $y \in [0, +1]$
- Regressão Logística
- Geralmente interpretada como probabilidade
- Saída não centrada em 0
- Satura os gradientes
- Não indicada para camadas ocultas
- Converge lentamente

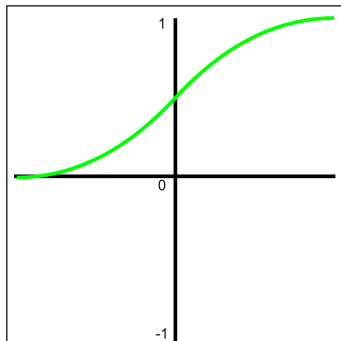
Função:

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

Derivada:

$$\frac{\partial y}{\partial x} = y(1 - y)$$

Funções de Ativação



Tanh

- $y \in [-1, +1]$
- Uma versão da Sigmoid
- Saída centrada em 0
- Satura os gradientes. um pouco menos que a Sigmoid
- Converge lentamente

Função:

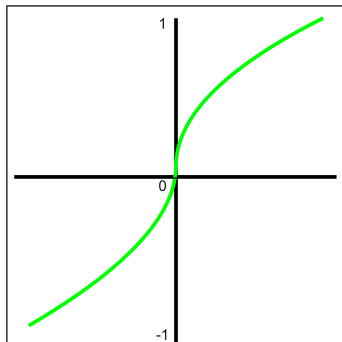
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Derivada:

$$\frac{\partial y}{\partial x} = 1 - y^2$$



Funções de Ativação



Relu

- $y \in [0, +\infty]$
- Não tem derivada para valores < 0
- Simples e Eficiente
- Evita a saturação dos gradientes
- Converge mais rápido
- Usada nas camadas escondidas
- Ela mata neurônio

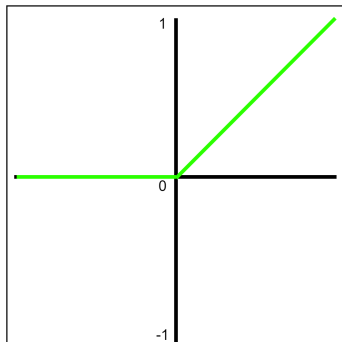
Função:

$$f(x) = \max(0, x)$$

Derivada:

$$\frac{\partial y}{\partial x} = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases}$$

Funções de Ativação



Leaky Relu

- $y \in [-\infty, +\infty]$
- Tem derivada para valores < 0
- Simples e Eficiente
- Evita a saturação dos gradientes
- Converge mais rápido
- Usada nas camadas escondidas
- Diminui as mortes de neurônios

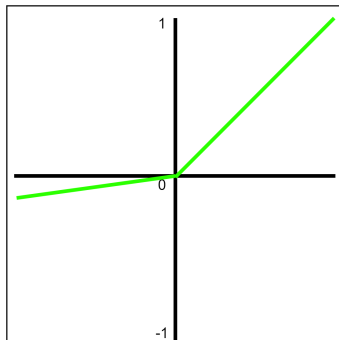
Função:

$$f(x) = \begin{cases} \alpha(e^x - 1), & x \leq 0 \\ x, & x > 0 \end{cases}$$









Derivada:

$$\frac{\partial y}{\partial x} = \begin{cases} \alpha, & x \leq 0 \\ 1, & x > 0 \end{cases}$$

Funções de Ativação



Funções de Ativação

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$



Qual função de ativação usar?

- Evitar Sigmoid nas camadas ocultas, boa na saída
- Tanh usada em modelos generativos
- Relu Muito boa nas camadas ocultas
- Linear não usar em camadas escondidas
- Leaky Relu raramente usadas



Softmax

- $y \in [0, 1]$ e $\sum_y = 1$
- Aplicada em dois ou mais neurônios. Pega saída e converte
- A saída é uma confiança
- Nunca nas camadas ocultas
- Multiclasses
- Saída One-hot Encode

Função:

$$S_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

Assim para cada k : $P^k = S_i^{[k]}$ Derivada:

$$\frac{\partial S}{\partial y} = p^k * (1 - p^k)$$

Softmax

Example

$[0.1, 1.3, 2.5] \rightarrow [0.07, 0.22, 0.72]$

$$\frac{e^{0.1}}{e^{0.1} + e^{0.3} + e^{2.5}}$$

Função:

$$S_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

Assim para cada k: $P^k = S_i^{[k]}$ Derivada:

$$\frac{\partial y}{\partial x} = \begin{cases} \alpha, x \leq 0 \\ 1, x > 0 \end{cases}$$

Vamos ver na prática

Vamos praticar utilizando o notebook 03_funções de ativação



Backpropagation

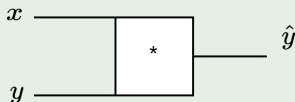
Para que usamos o *backpropagation*?

Utilizamos o Backpropagation no treinamento das redes neurais

Example

Vamos tomar uma função simples de multiplicação:

$$f(x, y) = x * y$$

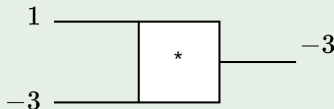


Backpropagation

Example

Vamos tomar uma função simples de multiplicação:

$$f(x, y) = x * y$$

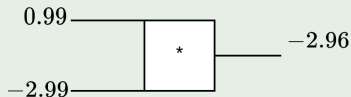


Com que força alterar as entradas desse circuito para de maneira “leve” alteremos a saída?

Example

Vamos tomar uma função simples de multiplicação:

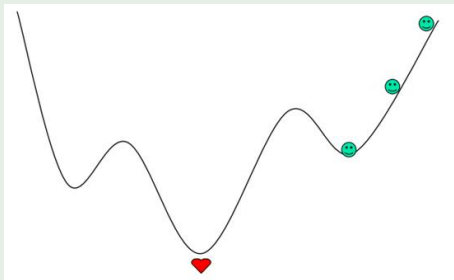
$$f(x, y) = x * y$$



Example

O Problema.....

- Para qual direção seguir?
- Com que velocidade seguir?
- Como sei se cheguei no local mais baixo?



Vamos ver na prática

Vamos praticar utilizando o notebook 03_backpropagation Será apresentado a busca aleatória e a busca aleatória local



Backpropagation

Como atualizamos os pesos?

Diante disso...

É possível encontrar a força de atualização dos pesos com **derivadas**

► Figura demonstrando

Qual a definição básica de derivada?

A derivada em relação à x pode ser definida como:

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Neste caso, o h tende a 0, ou seja, é um valor bem pequeno.



Backpropagation

Qual a definição básica de derivada parcial?

E se tivéssemos n argumentos (componentes)?

A derivada parcial de uma função com n argumentos $(x_1, x_2, x_3, \dots, x_n)$ é dada por:

$$\frac{\partial f}{\partial x_i}(x_1, \dots, x_n) = \lim_{h \rightarrow 0} \frac{f(x_1, \dots, x_i + h, x_n) - f(x_1, \dots, x_n)}{h}$$

Basta derivar cada elemento, ou cada componente da função.

Example

Derivada da função $f(x) = x^2$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

Backpropagation

Example

Derivada da função $f(x) = x^2$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$h \rightarrow 0$ tende a zero

$f(x+h)$ é p próprio $f(x)$ somado com um pequeno passo
– $f(x)$ que é a própria função

Dividido por h normalizando a derivação

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$\frac{(x+h)^2 - x^2}{h} \rightarrow \frac{\cancel{x^2} + 2xh + h^2 - \cancel{x^2}}{h} \rightarrow \frac{h(2x+h)}{\cancel{h}} \rightarrow 2x + \overset{0}{\cancel{h}} \rightarrow 2x$$

Example

Derivada da função $f(x) = x * y$

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

$$\frac{\partial f(x, y)}{\partial x} = \frac{(x+h)y - xy}{h} = \frac{\cancel{xy} + yh - \cancel{xy}}{h} = \frac{y\cancel{h}}{\cancel{h}} = y$$

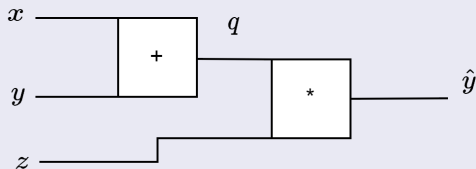
$$\frac{\partial f(x, y)}{\partial y} = \frac{x(y+h) - xy}{h} = \frac{\cancel{xy} + xh - \cancel{xy}}{h} = \frac{x\cancel{h}}{\cancel{h}} = x$$



Backpropagation

Mais de duas variáveis na função...

E se a função tiver 3 componentes? $f(x, y, z) = (x + y)z$



Neste caso, a melhor opção é decompor a função em subfunções, ou pequenos circuitos



Backpropagation

Mais de duas variáveis na função...

$$f(x, y, z) = (x + y)z$$

$$q(x, y) = x + y$$

$$f(q, z) = qz$$

Example

$$q(x, y) = x + y$$

$$\frac{\partial f(x, y)}{\partial x} = 1$$

$$\frac{\partial f(x, y)}{\partial y} = 1$$

$$f(q, z) = qz$$

$$\frac{\partial f(x, y)}{\partial q} = z$$

$$\frac{\partial f(x, y)}{\partial z} = q$$

Backpropagation

Mais de duas variáveis na função...

Se quisermos calcular a derivada em relação a uma entrada, utilizamos a regra da cadeia:

Ex: derivada em relação a x de f

- É a derivada de q em relação a f
- Derivada de x em relação a q

Example

$$\frac{\partial f(x, y)}{\partial x} = \frac{\partial f(x, y)}{\partial q} * \frac{\partial f(x, y)}{\partial x}$$

$$\frac{\partial f(x, y)}{\partial y} = \frac{\partial f(x, y)}{\partial q} * \frac{\partial f(x, y)}{\partial y}$$

$$\frac{\partial f(x, y)}{\partial z} = q$$

Backpropagation

Mais de duas variáveis na função...

Vamos verificar se dividindo a função principal em subfunções é válido.

Example

$$\begin{aligned}\frac{\partial f(x,y,z)}{\partial x} &= \lim_{h \rightarrow 0} \frac{f(x+h,y,z) - f(x,y,z)}{h} = \\ &= \frac{(x+h+y)*z - (x+y)*z}{h} = \\ &= \frac{\cancel{xz} + \cancel{hz} + \cancel{yz} - \cancel{xz} - \cancel{yz}}{h} = \\ &= \frac{hz}{h} = z\end{aligned}$$



Backpropagation

Mais de duas variáveis na função...

Vamos verificar se dividindo a função principal em subfunções é válido.

Example

$$\begin{aligned}\frac{\partial f(x,y,z)}{\partial y} &= \lim_{h \rightarrow 0} \frac{f(x,y+h,z) - f(x,y,z)}{h} = \\ &= \frac{(x+y+h)*z - (x+y)*z}{h} = \\ &= \frac{\cancel{xz} + \cancel{hz} + \cancel{yz} - \cancel{xz} - \cancel{yz}}{h} = \\ &= \frac{\cancel{hz}}{h} = z\end{aligned}$$



Backpropagation

Mais de duas variáveis na função...

Vamos verificar se dividindo a função principal em subfunções é válido.

Example

$$\begin{aligned}\frac{\partial f(x,y,z)}{\partial z} &= \lim_{h \rightarrow 0} \frac{f(x,y,z+h) - f(x,y,z)}{h} = \\ \frac{(x+y)*(z+h) - (x+y)*z}{h} &= \\ \frac{\cancel{xz} + \cancel{xh} + \cancel{yz} + yh - \cancel{xz} - \cancel{yz}}{h} &= \\ \frac{xh + yh}{h} &= \\ \frac{h(x+y)}{h} &= x + y\end{aligned}$$

Podemos notar que a saída é a porta de soma, ou seja. o próprio q



Vamos derivar o Neurônio Sigmoidal

- $y \in [0, +1]$
- Regressão Logística
- Geralmente interpretada como probabilidade
- Saída não centrada em 0
- Satura os gradientes
- Não indicada para camadas ocultas
- Converge lentamente

Função:

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



Vamos derivar o Neurônio Sigmoidal

- $y \in [0, +1]$
- Regressão Logística
- Geralmente interpretada como probabilidade
- Saída não centrada em 0
- Satura os gradientes
- Não indicada para camadas ocultas
- Converge lentamente

Função:

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

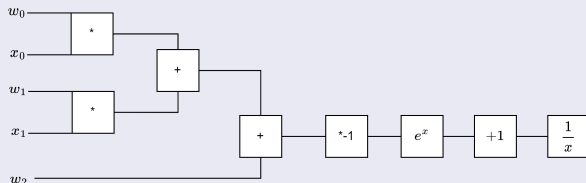


Vamos derivar o Neurônio Sigmoide

Função: $f(w, x) = \frac{1}{1 - e^{-x}}$

O w são os pesos e o x as entradas...

$$f(w, x) = \frac{1}{1 - e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



Example

Vamos praticar um pouquinho...

A derivada de $x = a * b$ é:

$$da = b * dx$$

Sempre teremos a derivada de dentro da multiplicando a derivada de quem está fora dx . Desta forma, podemos decompor qualquer função complexa.

$$\text{E } db = a * dx$$

$$x = a * b$$

$$da = b * dx$$

$$db = a * dx$$

$$x = a + b$$

$$da = 1 * dx$$

$$db = 1 * dx$$



Example

Vamos ver a derivada da função $x = a + b + c$

Primeiro vamos dividir por partes: $q = a + b$ e $x = q + c$

$$dq = 1 * dx$$

$$x = a + b + c$$

$$dc = 1 * dx$$

$$da = 1 * dx$$

$$db = 1 * dq$$

$$db = 1 * dx$$

$$da = 1 * dq$$

$$dc = 1 * dx$$

Intuitivamente pode-se perceber que a derivação da soma sempre será $1 * dx$



Backpropagation

Example

$$x = a * b + c$$

$$q = a * b dx$$

$$x = q + c$$

$$dq = 1 * dx$$

$$dc = 1 * dx$$

$$db = a * dq$$

$$da = b * dq$$

Example

$$x = a * a$$

$$da = 2a * dx$$

Example

$$x = a * a + b * b + c * c$$

$$da = 2a * dx$$

$$db = 2b * dx$$

$$dc = 2c * dx$$



Example

Vamos ver este exemplo mais complexo

$$x = ((a * b + c) * d)^2$$

$$x_1 = a * b + c$$

$$x_2 = x_1 * d$$

$$x = x_2 * x_2$$

$$dx_2 = 2x_2 * dx$$

$$dx_1 = d * dx_2$$

$$dd = x_1 * dx_2$$

$$da = b * dx_1$$

$$db = a * dx_1$$

$$dc = 1 * dx_1$$



Example

Vamos ver este exemplo de divisão

$$x = \frac{1}{a}$$

$$da = \left(-\frac{1}{a * a}\right) * dx_1$$



Example

$$x = \frac{a + b}{c + d}$$

$$x_1 = a + b$$

$$x_2 = c + d$$

$$x_3 = \frac{1}{x_2}$$

$$x = x_1 * x_3$$

$$dx_1 = x_3 * dx$$

$$dx_3 = x_1 * dx$$

$$dx_2 = -\frac{1}{x_2 * x_2} * dx_3$$

$$dc = 1 * dx_2$$

$$dd = 1 * dx_2$$

$$da = 1 * dx_1$$

$$db = 1 * dx_1$$



Example

$$x = \max(a, b)$$

$$da = x == a ? 1 * dx : 0$$

$$da = x == b ? 1 * dx : 0$$

Lembram da Relu?

$$x = \max(0, a)$$

$$da = a > 0 ? 1 * dx : 0$$



Backpropagation

Como fazer derivação de matrizes

O que vimos até agora foram números escalares...

Como derivar uma matriz?

Example

```
 $W = np.random.randn(5, 10)$ 
```

```
 $X = np.random.randn(3, 10)$ 
```

```
 $Y = X.dot(W^T)$ 
```

W são nossos pesos, X nossas entradas, Y a saída

10 é a dimensão da entrada, 3 Qtd de amostras, 5 Qtd de neurônios.

Anteriormente só tínhamos 1 neurônio

A multiplicação de uma matriz de $3 \times 10 * 10 \times 5$ gerará uma saída 3×5



Backpropagation

Example

Vamos imaginar que nosso $Y = WX$ Se a multiplicação de uma matriz de $3 \times 10 * 10 \times 5$ gerará uma saída 3×5 a derivação de dY deverá ter a dimensão de 3×5 . Assim, a derivada de uma matriz sempre terá o shape da matriz original

$dY = np.random.randn(*Y.shape)$

A derivada de W dado $Y = WX$ será o de dentro $*$ o de fora $X * dY$, assim:

$dW = dY^T.dot(X)$

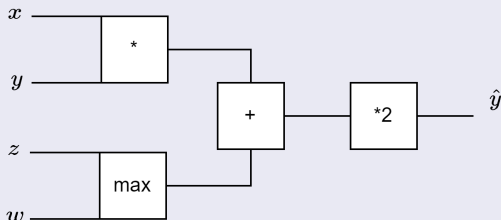
A derivada de X dado $Y = WX$ será o de dentro $*$ o de fora $W * dY$, assim:

$dY = dY.dot(W)$

A derivada de uma matriz deverá possuir o mesmo *shape* da matriz original, neste caso Y tem 3×5 e dY também

Resumo das derivadas

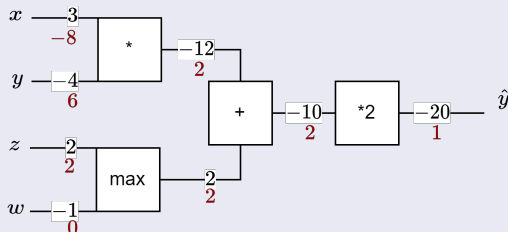
Vamos tomar este exemplo:



Backpropagation

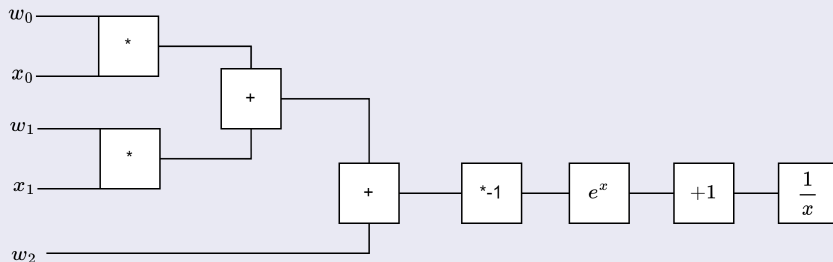
Resumo das derivadas

Vamos tomar este exemplo:



Derivando o neurônio Sigmoide

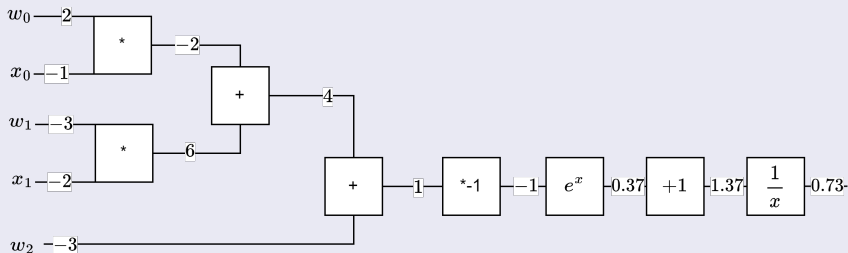
Agora que sabemos como derivar, vamos retornar ao neurônio Sigmoide:



Backpropagation

Derivando o neurônio Sigmoido

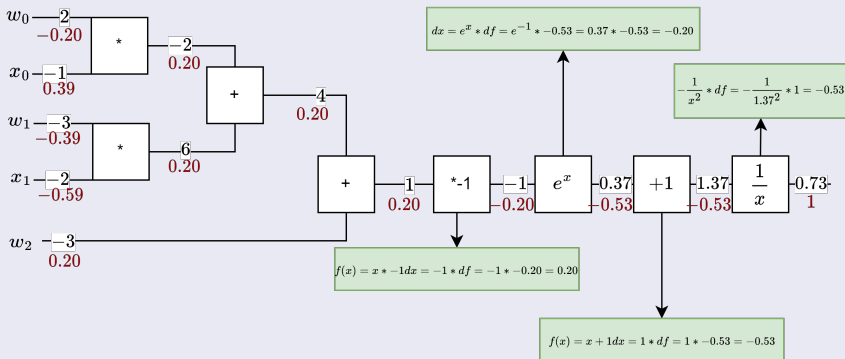
Vamos fazer o processo de *forward*.



Backpropagation

Derivando o neurônio Sigmoide

Após o *forward*, faremos agora a propagação reversa ou *backpropagation*.



Vamos ver na prática

Vamos praticar utilizando o notebook 04_backpropagation



Dimensão das matrizes

Até o momento vimos que um neurônio pode ser dado pela equação: $y = f(xw^T + b)$. Sendo f uma função de ativação. A função de ativação não altera o *shape* dos dados.

A princípio tivemos $x = [1 \times D_{in}]$ *shape*. Onde 1 era a quantidade de amostras e D_{in} o dimensões da amostra, por exemplo, a porta OR que recebe duas entradas (x_1, x_2) , assim $D_{in} = 2$.

O mesmo ocorre para a saída, $y = [1 \times D_{out}]$ *shape*. Onde 1 era a quantidade de amostras e D_{out} a dimensão de saída, em todos casos visto até agora igual a 1. Mas podemos ter quantas saídas forem necessárias. Mas qual a dimensão do *bias* e dos nossos pesos?



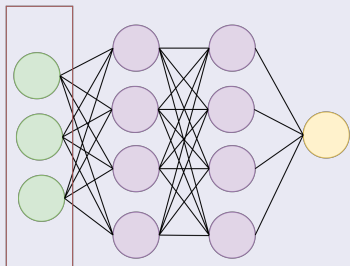
Dimensão das matrizes

Assim...

- $x = [1 \times D_{in}]$
- $y = [1 \times D_{out}]$
- $bias = [1 \times D_{out}]$
- $w^T = [D_{in} \times D_{out}]$
- $[1 \times D_{out}] = [1 \times D_{in}] * [D_{in} \times D_{out}] + [1 \times D_{out}]$



Dimensão das matrizes

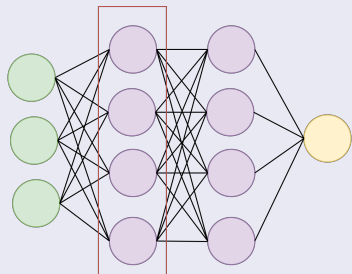


$[1 \times D_{in}]$	$[D_{in} \times D_{out}]$	$[1 \times D_{out}]$	$[1 \times D_{out}]$
1X3	3x4	1x4	1x4

A multiplicação das matrizes $1 \times 3 * 3 \times 4$ nos retornará uma matriz 1×4 , que deve ser a mesma dimensão do *bias*.

Cada neurônio possui um bias.

Dimensão das matrizes

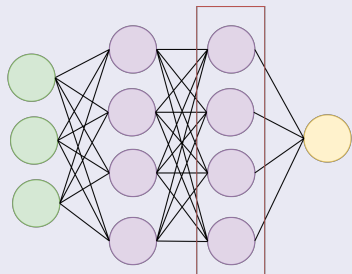


$[1 \times D_{in}]$	$[D_{in} \times D_{out}]$	$[1 \times D_{out}]$	$[1 \times D_{out}]$
1×3	3×4	1×4	1×4
1×4	4×4	1×4	1×4

A multiplicação das matrizes $1 \times 3 * 4 \times 4$ nos retornará uma matriz 1×4 , que deve ser a mesma dimensão do *bias*.

Cada neurônio possui um *bias*.

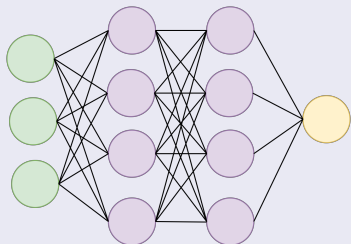
Dimensão das matrizes



$[1 \times D_{in}]$	$[D_{in} \times D_{out}]$	$[1 \times D_{out}]$	$[1 \times D_{out}]$
1×3	3×4	1×4	1×4
1×4	4×4	1×4	1×4
1×4	4×1	1×1	1×1

A multiplicação das matrizes $1 \times 1 * 4 \times 1$ nos retornará uma matriz 1×1 , que deve ser a mesma dimensão do *bias*.

Dimensão das matrizes



Quantos parâmetros treináveis esta rede possui?

$[1 \times D_{in}]$	$[D_{in} \times D_{out}]$	$[1 \times D_{out}]$	$[1 \times D_{out}]$
1X3	$3 \times 4 = 12$	$1 \times 4 = 4$	1x4
1X4	$4 \times 4 = 16$	$1 \times 4 = 4$	1x4
1X4	$4 \times 1 = 4$	$1 \times 1 = 1$	1x1

$$12 + 4 + 16 + 4 + 4 + 1 = 41 \text{ Parâmetros}$$



MAE- mean of absolute errors

$$J = \frac{1}{N} \sum |y_i - \hat{y}_i|$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{1}{N} \begin{cases} +1 & \text{se } \hat{y} > y \\ -1 & \text{se } \hat{y} < y \end{cases}$$

MSE- mean of squared errors

$$J = \frac{1}{2N} \sum |y_i - \hat{y}_i|^2$$

$$\frac{\partial J}{\partial \hat{y}} = -(y - \hat{y}) \frac{1}{N}$$



Sigmoid Cross Entropy

$$J = \frac{1}{N} \sum y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i)$$

$$\frac{\partial J}{\partial \hat{y}} = \frac{-(y - \hat{y})}{\hat{y}(1 - \hat{y})} \frac{1}{N}$$



One hot Encode

One hot Encode

Antes de falar sobre classificação de classes, vamos entender a seguinte situação:

y	One-hot
1	1 0 0 0
2	0 1 0 0
3	0 0 1 0
4	0 0 0 1

y	One-hot
3	0 0 1 0
2	0 1 0 0
4	0 0 0 1
1	1 0 0 0



Softmax

Função:

$$S_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

Assim para cada k : $p^k = S_i^{[k]}$ Derivada:

$$\frac{\partial S}{\partial y} = p^k * (1 - p^k)$$

Atenção

A Softmax não é uma função de custo.



Funções de Custo - Classificação Multiclasse

Softmax

$$S_i = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

Assim para cada k: $p^k = S_i^{[k]}$

Derivada:

$$\frac{\partial S}{\partial y} = p^k * (1 - p^k)$$

Neg. Log-likelihood

$$J = \frac{1}{N} \sum -\ln(p_i^k)$$

$$\frac{\partial J}{\partial p^k} = -\frac{1}{p^k}$$

Derivada:

$$\frac{\partial J}{\partial \hat{y}} = -(1 - p^k) = -(y - \hat{y}) \frac{1}{N}$$

Atenção

A Neg. Log-likelihood será o somatório do logarítmico da negação para cada elemento da Softmax pelo total de elementos.

Highlighting text

In this slide, some important text will be **highlighted** because it's important. Please, don't abuse it.

Remark

Sample text

Important theorem

Sample text in red box

Examples

Sample text in green box. The title of the block is “Examples”.

