Treinamento de Machine Learning e Deep Learning

Do Básico ao Avançado

Salomão Machado Mafalda[1]

[1]Universidade Federal do Acre
PAVIC

2023

Agenda

Perceptron

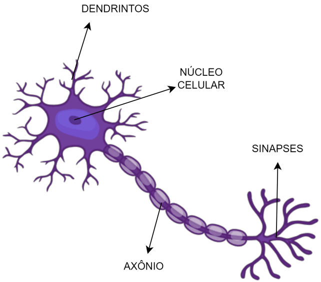


Figure: Neurônio humano

Perceptron

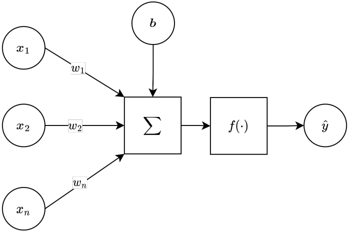
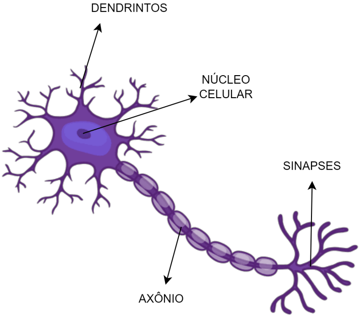
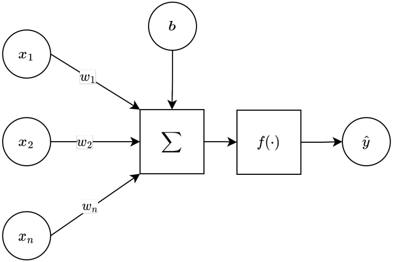


Figure: Neurônio Artificial

Perceptron



(a) Neurônio Humano



(b) Neurônio Artificial

Perceptron

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída ŷ

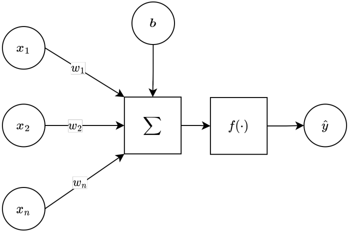
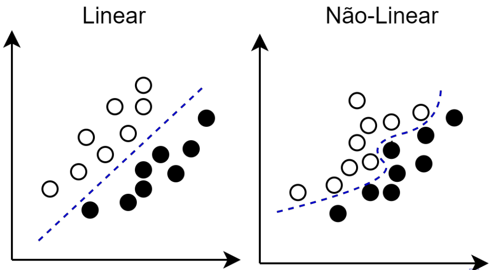


Figure: Neurônio Artificial

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

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

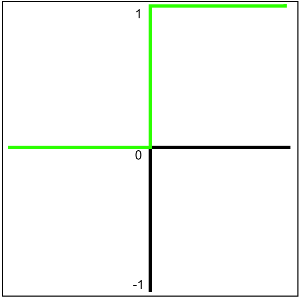


Perceptron

Função de ativação do perceptron

$$\begin{cases} 0 \ \textit{if} \ 0 > x \\ 1 \ \textit{if} \ x \geq 0 \end{cases}$$

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



Perceptron

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 yi - ŷi for 0 então não ocorrerá a atualização dos pesos"

Perceptron

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

Perceptron

Vamos ver na prática

Vamos praticar utilizando o notebook 00_perceptron

Adaline

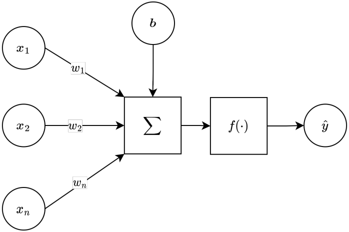


Figure: Neurônio Artificial

Adaline

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

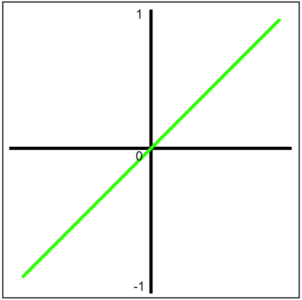
$$\hat{y} = f(\sum_i w_i x_i + b)$$

Adaline

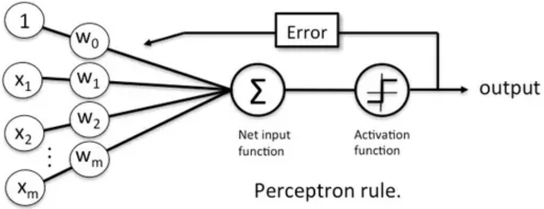
Função de ativação do Adaline

$$f(x) = x$$

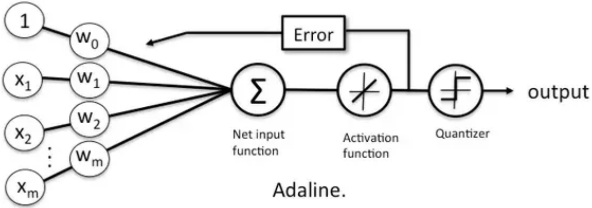
- Possibilita o cálculo da derivada



Adaline vs Perceptron

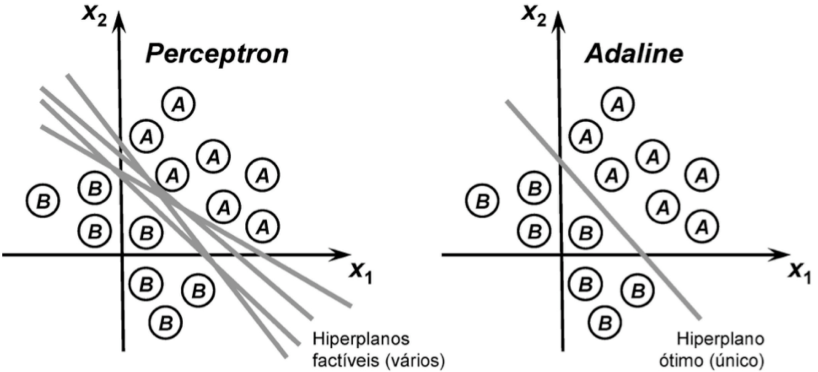


Perceptron rule.



Adaline.

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)

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

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

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

Vamos ver na prática

Vamos praticar utilizando o notebook 01_adaline

Neurônio Sigmoide

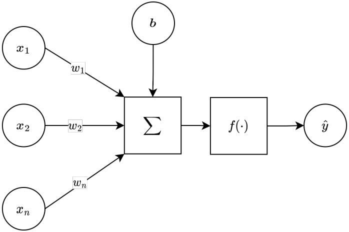


Figure: Neurônio Artificial

Neurônio Sigmoide

- Modelo mais básico de NN
- Um neurônio
- N entradas, Uma saída ŷ
- 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

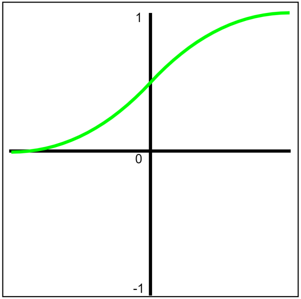
$$\hat{y} = f(\sum_i w_i x_i + b)$$

Neurônio Sigmoide

Função de ativação do Neurônio Sigmoide

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

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



Neurônio Sigmoide

Entropia Cruzada

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

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

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

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

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

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

Neurônio Sigmoide

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 Sigmoide

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 Sigmoide

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 $[1\ 0]$

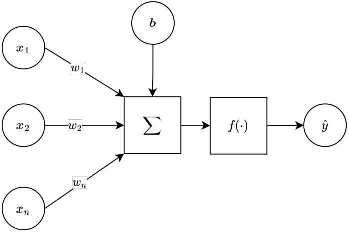


Figure: Neurônio Artificial

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

Funções de Ativação

Linear

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

Função:

$$f(x) = x$$

Derivada:

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

Funções de Ativação

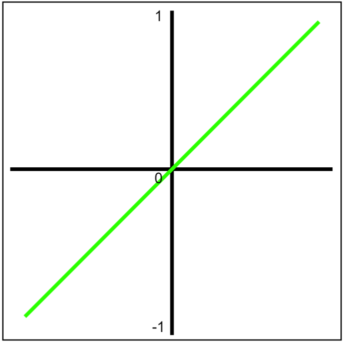
Atenção

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 x 10 = 10"

Podemos substituir todos pesos por um só



Funções de Ativação

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

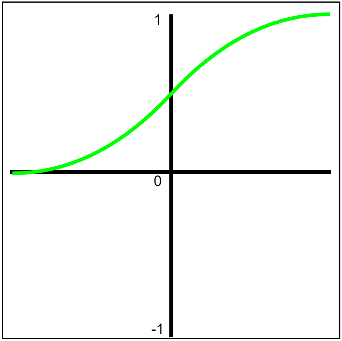
Função:

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

Derivada:

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

Funções de Ativação



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

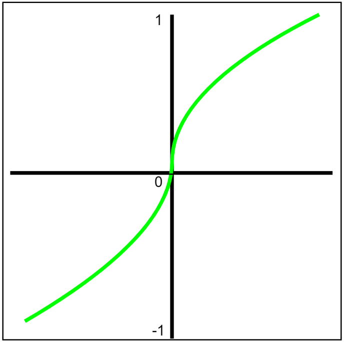
Função:

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

Derivada:

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

Funções de Ativação



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

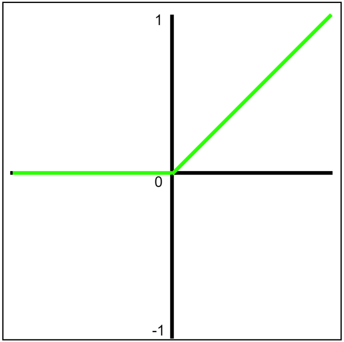
Função:

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

Derivada:

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

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

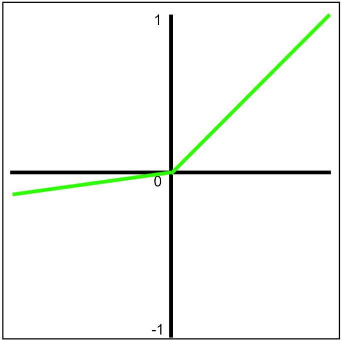
Função:

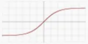
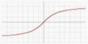
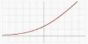
$$f(x) = \begin{cases} \alpha(e^x - 1), x \leq 0 \\ x, x > 0 \end{cases}$$

Derivada:

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

Funções de Ativação



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

Funções de Ativação

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

Funções de Ativação

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:

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

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

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

Funções de Ativação

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:

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

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

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

Funções de Ativação

Vamos ver na prática

Vamos praticar utilizando o notebook 03_funções de ativação

Backpropagation

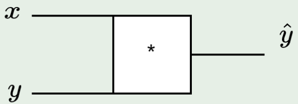
Para que usamos o *backpropagation*?

Utilizamos o Backpropagation no treinamento das redes neurais

Example

Vamos tomar uma função simples de multiplicação:

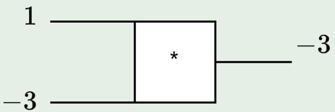
$$f(x, y) = x * y$$



Example

Vamos tomar uma função simples de multiplicação:

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



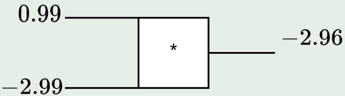
Com que força alterar as entradas desse circuito para de maneira "leve" alteremos a saída?

Backpropagation

Example

Vamos tomar uma função simples de multiplicação:

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

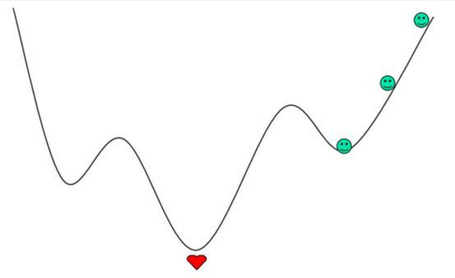


Backpropagation

Example

O Problema.....

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



Backpropagation

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:

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

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 $(x_1, x_2, x_3, ..., x_n)$ é dada por:

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

Basta derivar cada elemento, ou cada componente da função.

Example

Derivada da função $f(x) = x^2$

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

Backpropagation

Example

Derivada da função $f(x) = x^2$

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

$h \to 0$ 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

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

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

Backpropagation

Example

Derivada da função $f(x) = x * y$

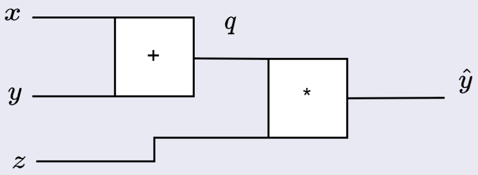
$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

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

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

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

$$q(x, y) = x + y \qquad f(q, z) = qz$$

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

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

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

$$\frac{\partial f(x,y,z)}{\partial x} = \lim_{h \to 0} \frac{f(x+h,y,z)-f(x,y,z)}{h} =$$

$$\frac{(x+h+y)*z-(x+y)*z}{h} =$$

$$\frac{\cancel{xz}+hz+\cancel{yz}-\cancel{xz}-\cancel{yz}}{h} =$$

$$\frac{\cancel{h}z}{\cancel{h}} = z$$

Mais de duas variáveis na função...

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

Example

$$\frac{\partial f(x,y,z)}{\partial y} = \lim_{h \to 0} \frac{f(x,y+h,z) - f(x,y,z)}{h} =$$

$$\frac{(x+y+h)*z-(x+y)*z}{h} =$$

$$\frac{\cancel{xz}+hz+\cancel{yz}-\cancel{xz}-\cancel{yz}}{h} =$$

$$\frac{\cancel{h}z}{\cancel{h}} = z$$

Mais de duas variáveis na função...

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

Example

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

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

Backpropagation

Vamos derivar o Neurônio Sigmoide

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

Função:

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

Backpropagation

Vamos derivar o Neurônio Sigmoide

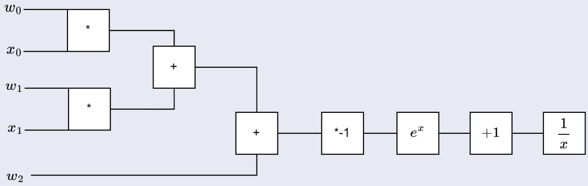
- $y \in [0, +1]$
- Regressão Logística
- Geralmente interpretada como probabilidade
- Saída não centrada em 0
- Satura os gradientes
- Não indicada para camadas ocultas
- Converge lentamente

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...
$f(w,x) = \frac{1}{1-e^{-(w_0x_0+w_1x_1+w_2)}}$



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.

E $db = a * dx$

$$x = a * b \qquad\qquad x = a + b$$

$$da = b * dx \qquad\qquad da = 1 * dx$$

$$db = a * dx \qquad\qquad db = 1 * dx$$

Backpropagation

Example

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

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

$$dq = 1 * dx \qquad x = a + b + c$$

$$dc = 1 * dx \qquad da = 1 * dx$$

$$db = 1 * dq \qquad db = 1 * dx$$

$$da = 1 * dq \qquad dc = 1 * dx$$

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

Backpropagation

Example

$$x = a * b + c$$

$$q = a * bdx$$

$$x = q + c$$

$$dq = 1 * dx$$

$$dc = 1 * dx$$

$$db = a * dq$$

$$da = b * dq$$

Example

$$x = a * a$$

$$da = 2a * dx$$

Example

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

$$da = 2a * dx$$

$$db = 2b * dx$$

$$dc = 2c * dx$$

Backpropagation

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

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

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

Backpropagation

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 $3x10 * 10x5$ gerará uma saída $3x5$

Example

Vamos imaginar que nosso $Y = WX$ Se a multiplicação de uma matriz de $3x10 * 10x5$ gerará uma saída $3x5$ a derivação de dY deverá ter a dimensão de $3x5$. 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:

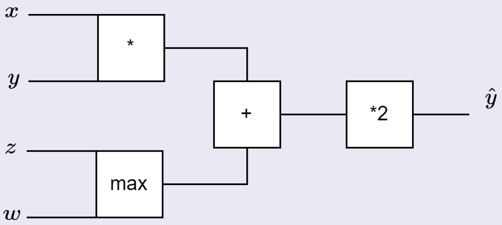
$dY = dY.dot(W)$

A derivada de uma matriz deverá possuir o mesmo *shape* da matriz original, neste caso Y tem $3x5$ e dY também

Backpropagation

Resumo das derivadas

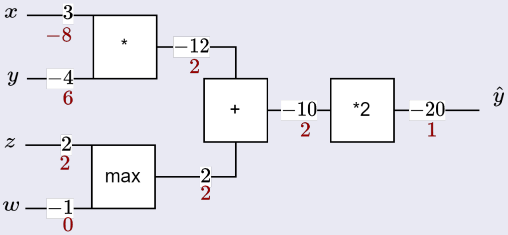
Vamos tomar este exemplo:



Backpropagation

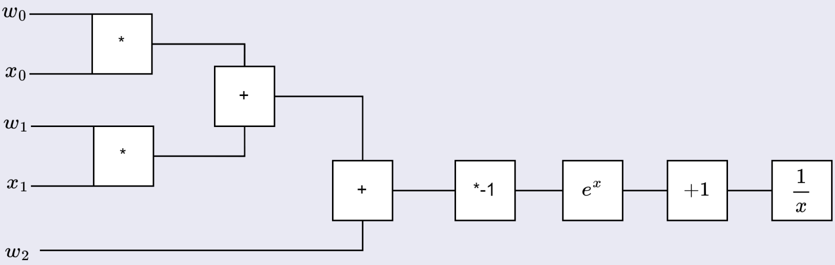
Resumo das derivadas

Vamos tomar este exemplo:



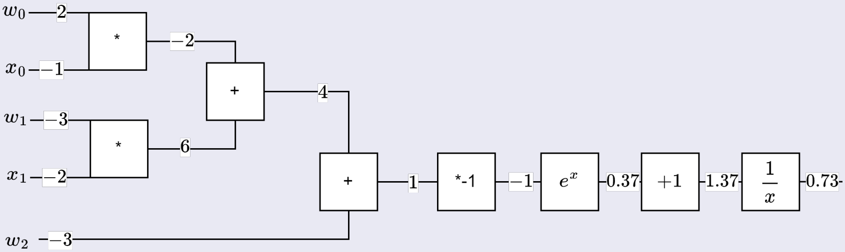
Derivando o neurônio Sigmoide

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



Derivando o neurônio Sigmoide

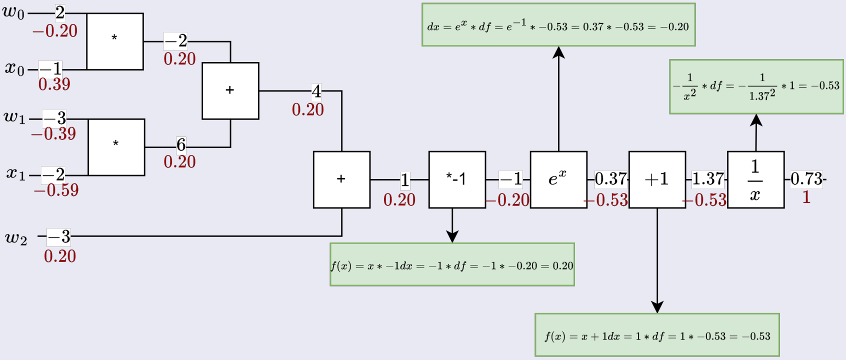
Vamos fazer o processo de *forward*.



Backpropagation

Derivando o neurônio Sigmoide

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



Funções de Ativação

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 $x = [1xD_{in}]$ *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, $y = [1xD_{out}]$ *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...

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

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".