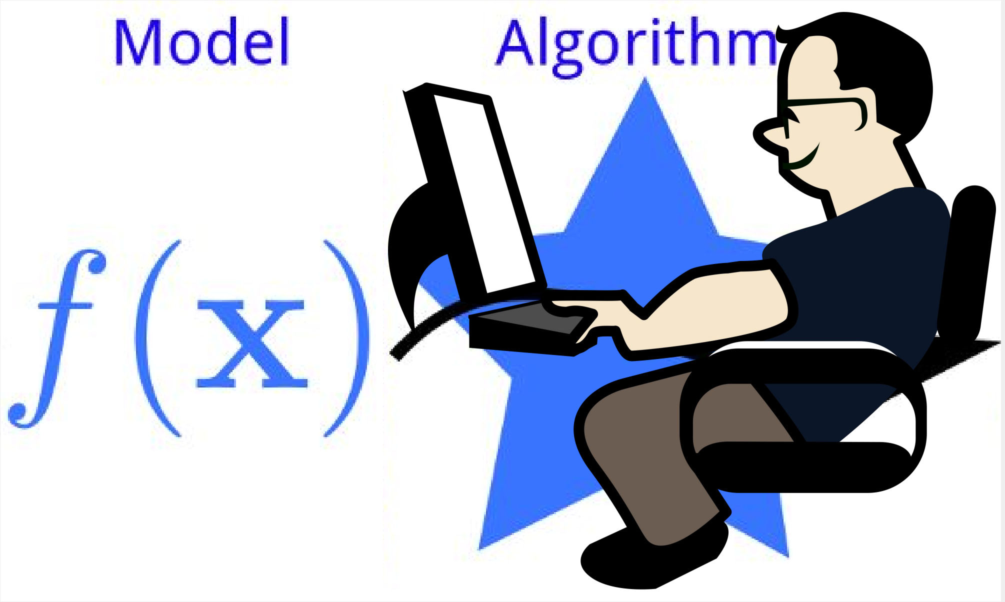
JAI 6 - Deep Learning Teoria e Prática

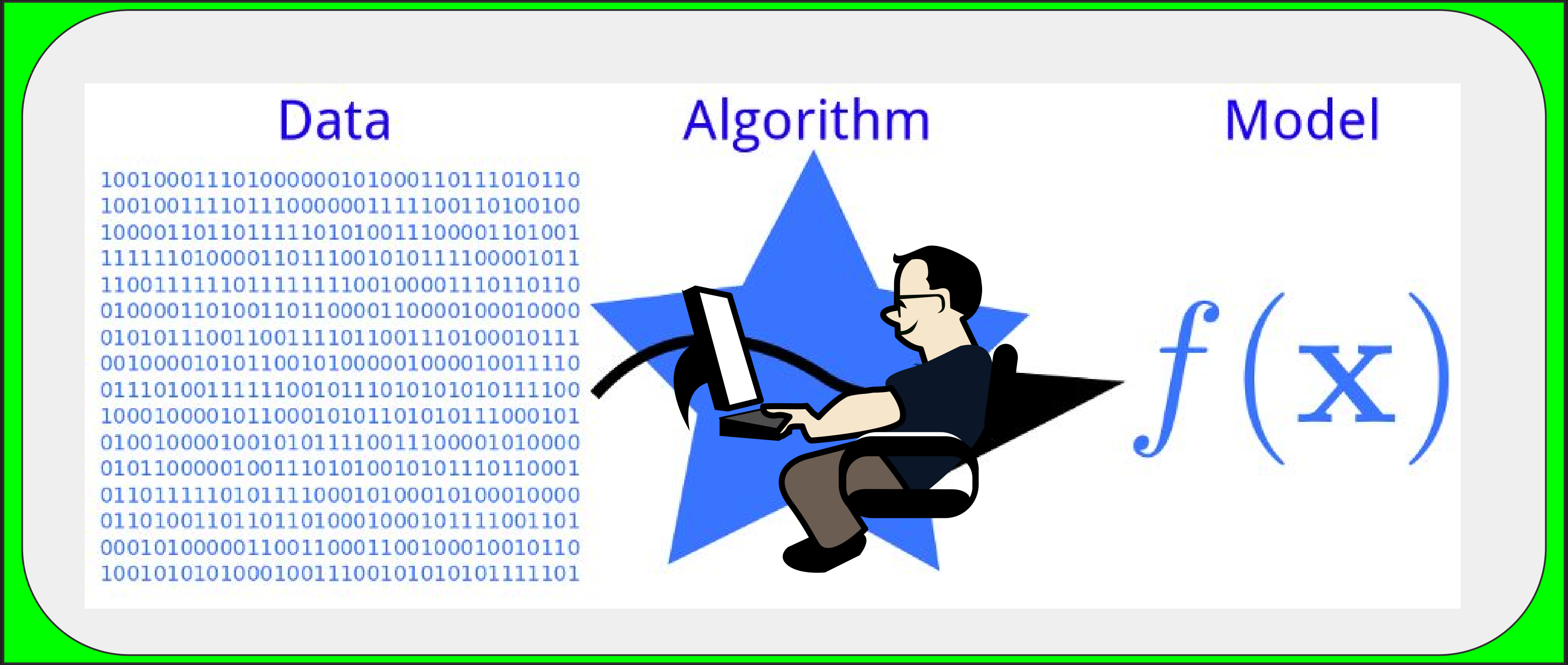
Esteban Clua e Cristina Nader Vasconcelos
Universidade Federal Fluminense

Fundamentos

Computação baseada em modelos

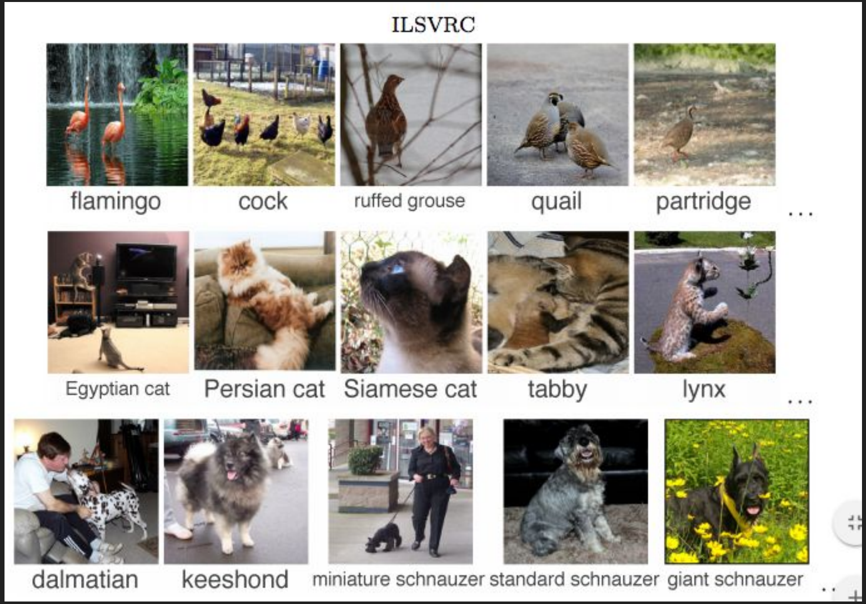


Computação baseada em aprendizado



Ingrediente chave: dados

O aprendizado profundo se beneficia com o aumento do número de exemplos coletados para a formação da base de treinamento



Aprendizado de máquina: categorias

Aprendizado supervisionado:

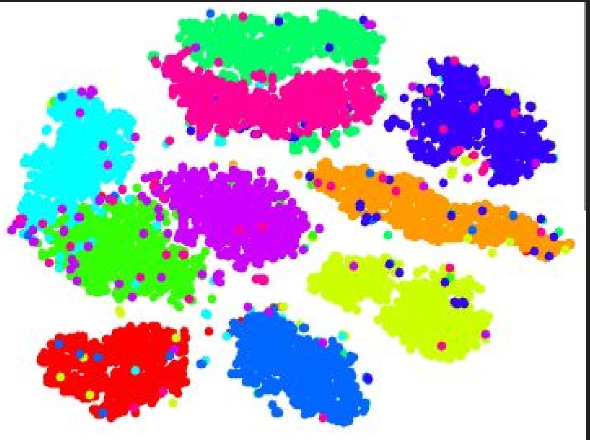
- são conhecidos ***pares*** associando ***entradas*** às respectivas ***respostas desejadas***;
- buscam aprender uma ***função que mapeia*** as entradas nas saídas desejadas, para ser aplicada a novas entradas.

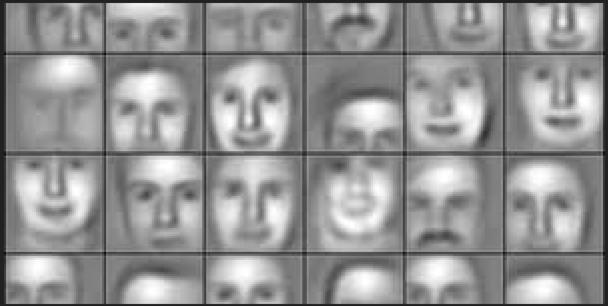
gatos ***cachorros***

Aprendizado de máquina: categorias

Aprendizado não-supervisionado:

- ***não são fornecidas saídas desejadas***, mas se deseja encontrar ou fazer inferências sobre ***padrões inerentes*** ao comportamento das amostras de entrada.





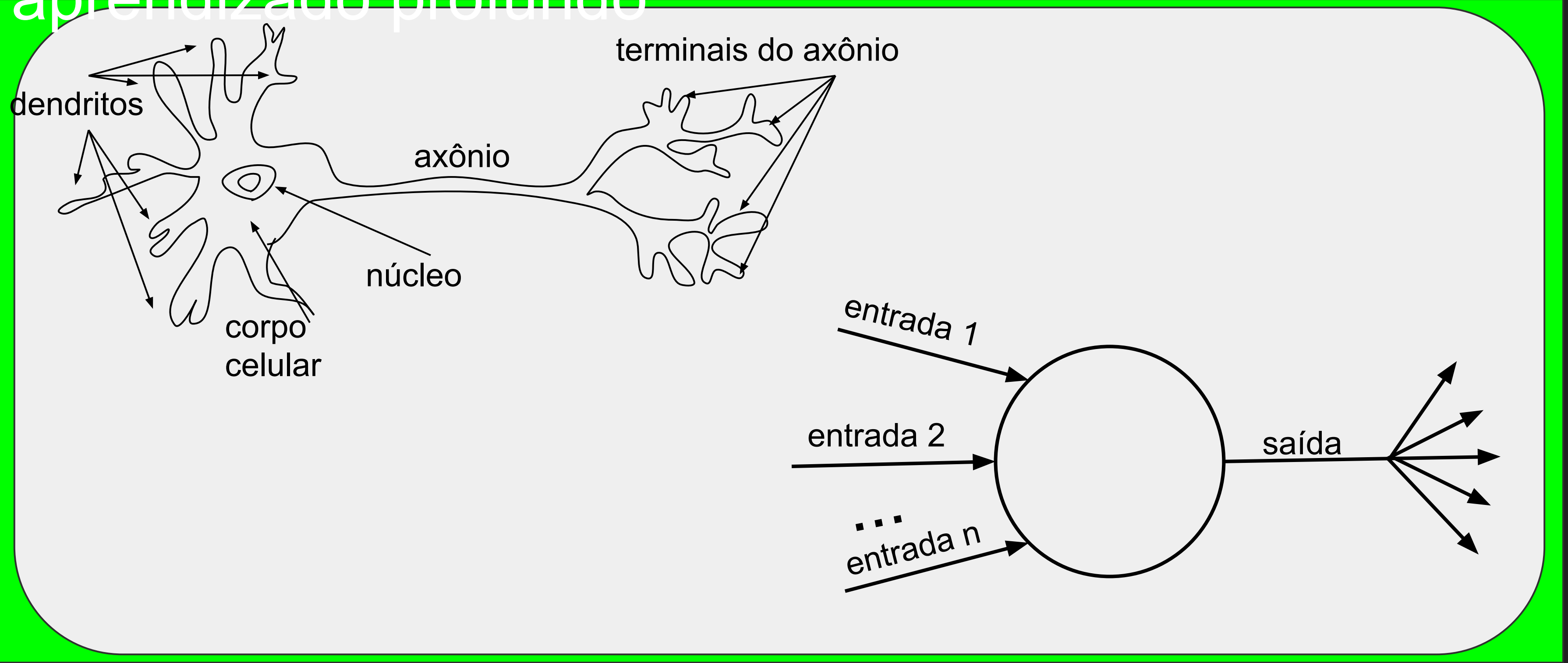
[Lee et al.- 2009]

Aprendizado de máquina: categorias

Aprendizado por reforço:

- agente interage com um ***ambiente*** para tomar ***decisões***. Ao praticar por tentativa e erro, recebe ***recompensas*** ou punições como ***feedback*** pelas decisões tomadas, alterando seu ***estado***. Com o passar do tempo, passa a acumular conhecimento com a ***experiência*** passada, para tomar decisões futuras.

O perceptron: base das abordagens por aprendizado profundo



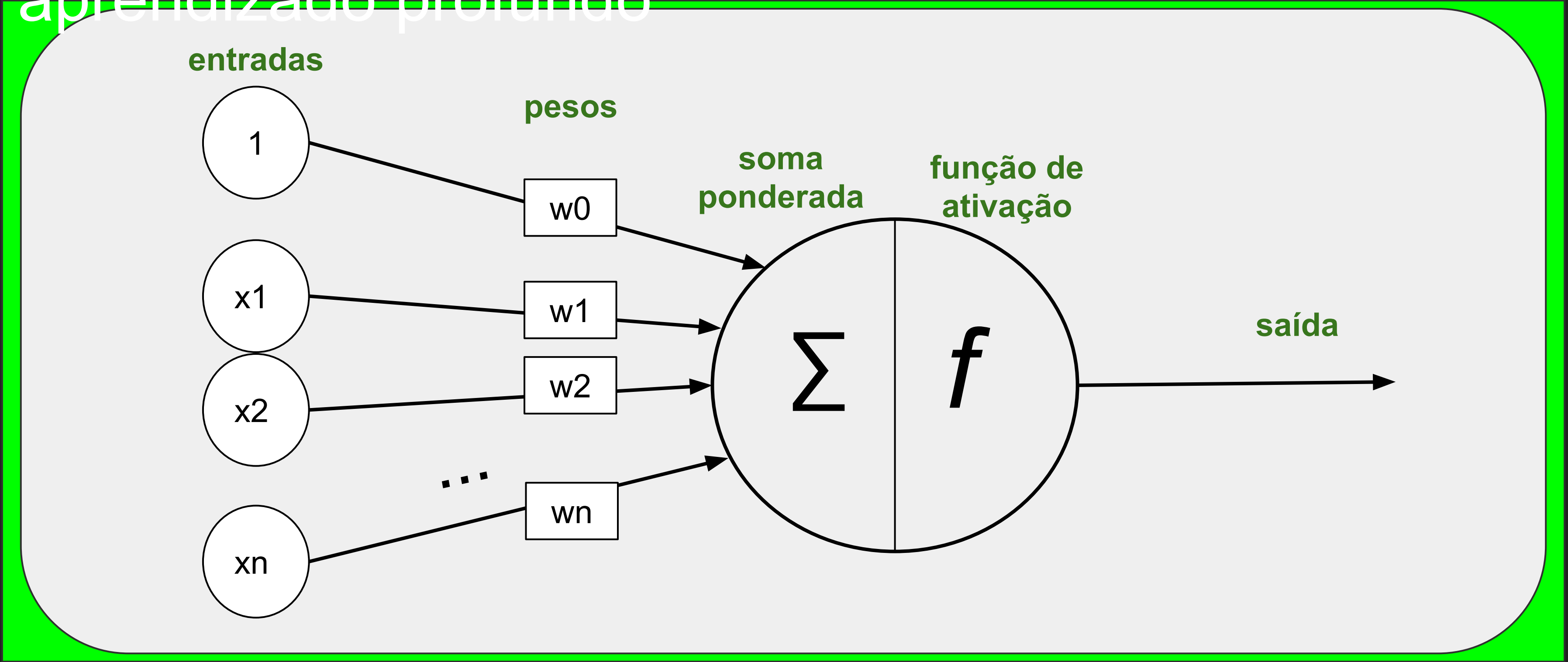
O perceptron de Rosenblatt (1957)

$$z = W \cdot X + b = \sum_{i=1}^{N} w_i x_i + b = w_1 x_1 + w_2 x_2 + \ldots + w_N x_N + b$$

Função de ativação desempenha o papel de aplicação de um limiar:

$$f(z) = \begin{cases} 1, & se\ W \cdot X + b > 0 \\ 0, & cc. \end{cases}$$

O perceptron: base das abordagens por aprendizado profundo



Treinando o perceptron

- Inicializa-se seus pesos (ex: valores entre -1 e 1) e bias b (ex: zero).

Repetidas vezes:

(i) processa-se um sinal de entrada X(t) obtendo ***o(t) = f(z(X(t)))*** usando:

$$z = W \cdot X = \sum_{i=0}^{N} w_i x_i = w_0 1 + w_1 x_1 + w_2 x_2 + \ldots + w_N x_N$$

$$f(z) = \begin{cases} 1, & se\ W \cdot X + b > 0 \\ 0, & cc. \end{cases}$$

Treinando o perceptron

- Inicializa-se seus pesos (ex: valores entre -1 e 1) e bias b (ex: zero).

Repetidas vezes:

(i) processa-se um sinal de entrada X(t) obtendo ***o(t) = f(z(X(t)))***

(ii) ajusta-se os ***parâmetros*** comparando-se o(t) e o resultado desejado Y(t) :

$$w(t+1) = W(t) + (Y - o(t))\ X$$

O perceptron de Rosenblatt: exemplo

T = 0

w0 = 1

w1= 0.2

w2 = -0.3

Pares de treinamento da função ***AND***:

0,0 -> 0
0,1 -> 0
1,0 -> 0
1,1 -> 1

O perceptron de Rosenblatt: exemplo

Pares de treinamento da função AND:

(i)

0,0 -> 0

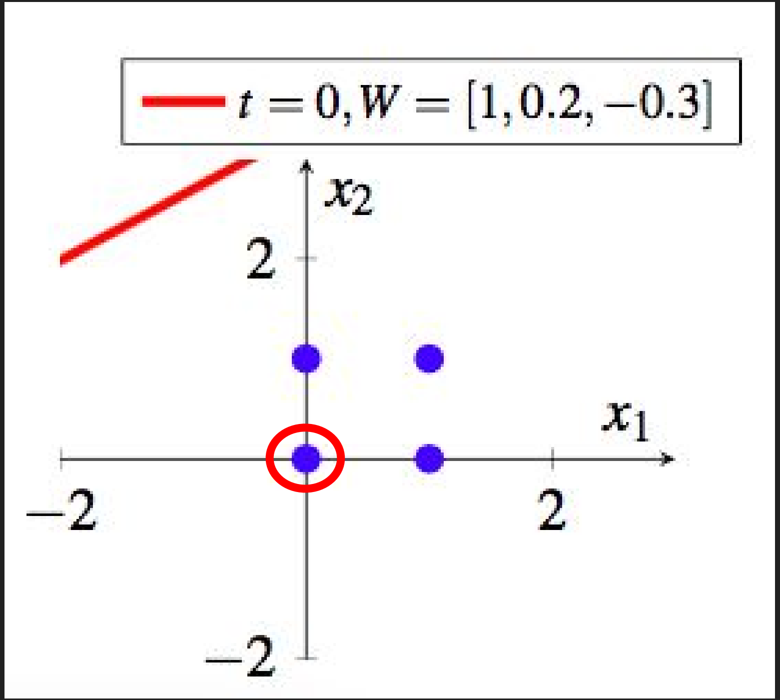
o(t=0) = f([1, 0.2, -0.3]. [1 0 0]) = 1

(ii)

w0 = 1 + (0-1)*1 = 0

w1= 0.2 + (0-1)*0 = 0.2

w2 = -0.3+ (0-1)*0= -0.3



O perceptron de Rosenblatt: exemplo

Pares de treinamento da função AND:

(i)

0,1 -> 0

o(t=1) = f(0*1+0.2*0-0.3*1)

= f(-0.3) = 0

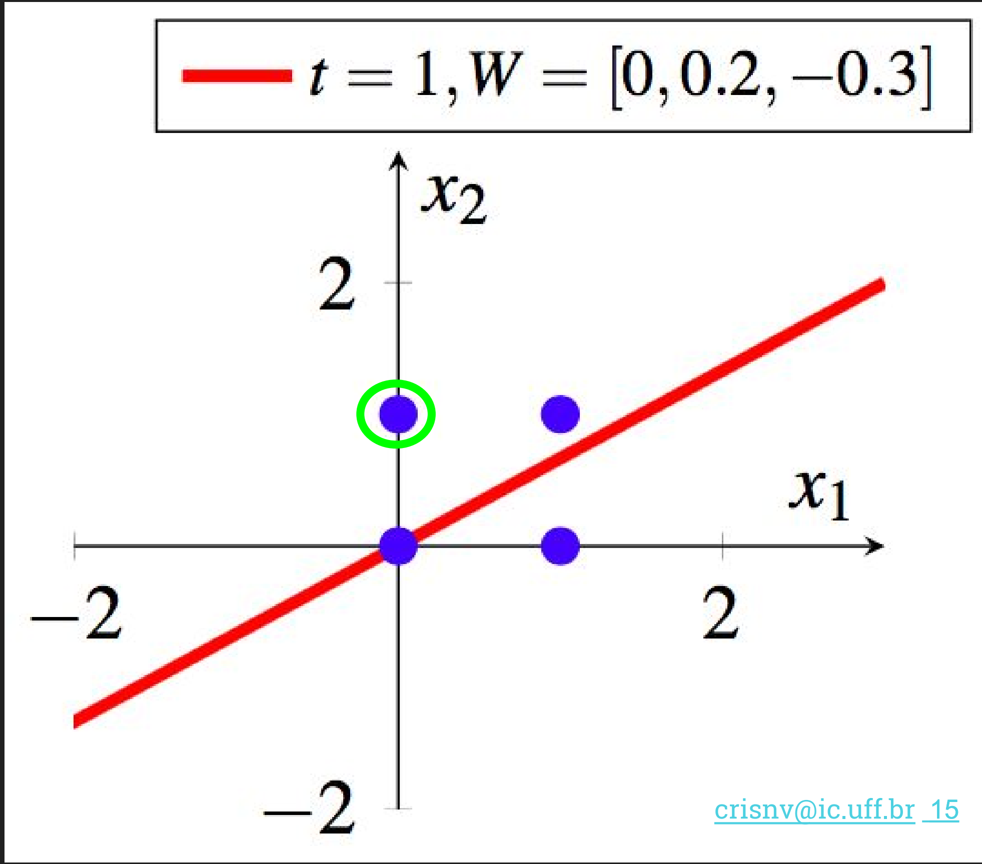
(ii)

w0 = 0 + (0-0)*1 = 0

w1= 0.2 + (0-0)*1 = 0.2

w2 = -0.3+ (0-0)*0= -0.3

T = 2



O perceptron de Rosenblatt: exemplo

Pares de treinamento da função AND:

(i)

1,0 -> 0

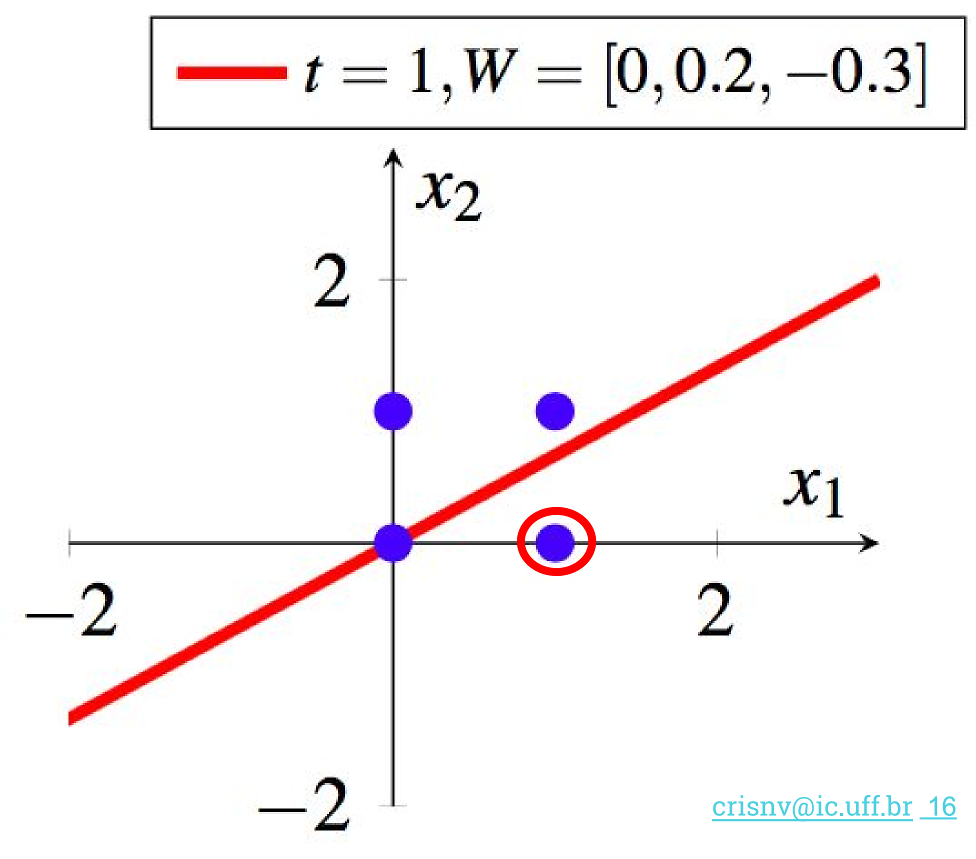
o(t=2) = f(0*1+0.2*1-0.3*0) = 1

(ii)

w0 = 0 + (0-1)*1   = -1

w1= 0.2 + (0-1)*1 = -.8

w2 = -0.3+ (0-1)*0= -0.3



O perceptron de Rosenblatt: exemplo

Pares de treinamento da função AND:

(i)

1,1 -> 1

o(t=3) = f(-1*1-0.8*1-0.2*1)
= 0

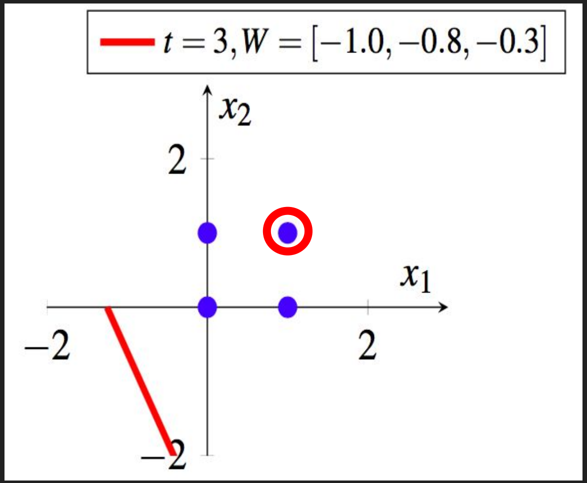
(ii)

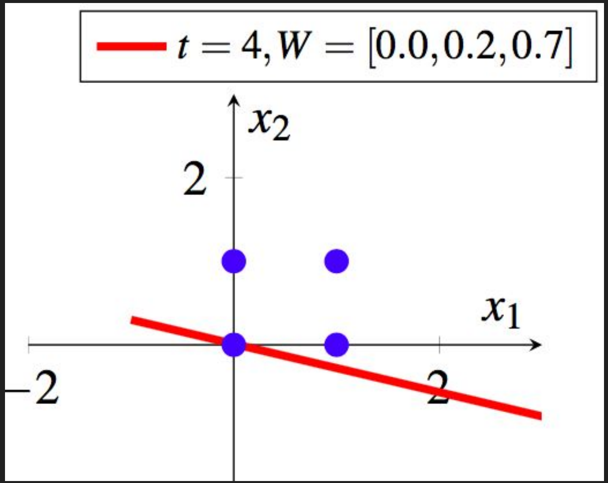
w0 = -1 + (1-0)*1 = 0

w1= -0.8 + (1-0)*1= 0.2

w2 = -0.3+ (1-0)*1= 0.7

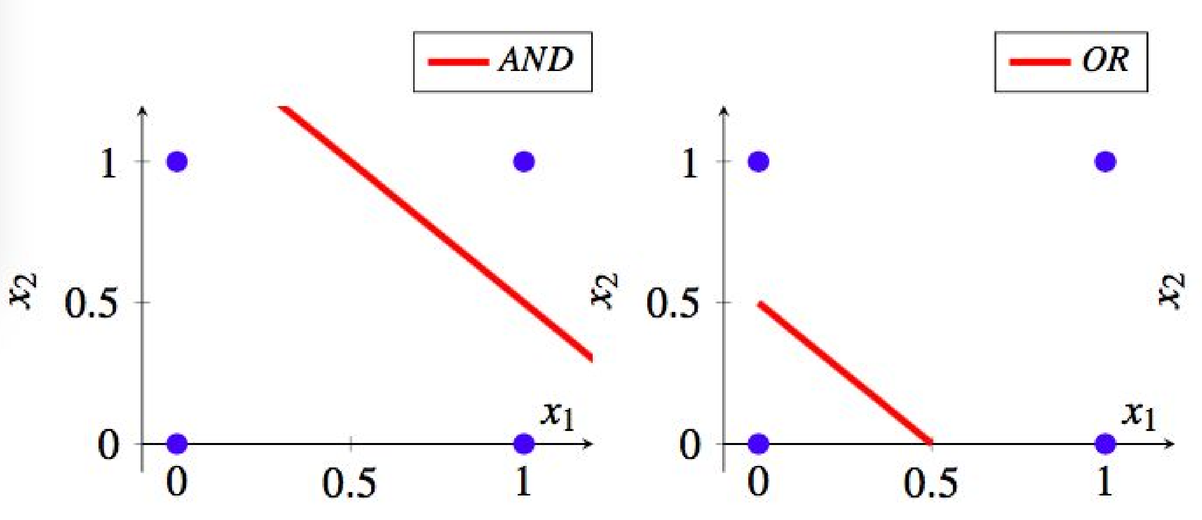
… e continua



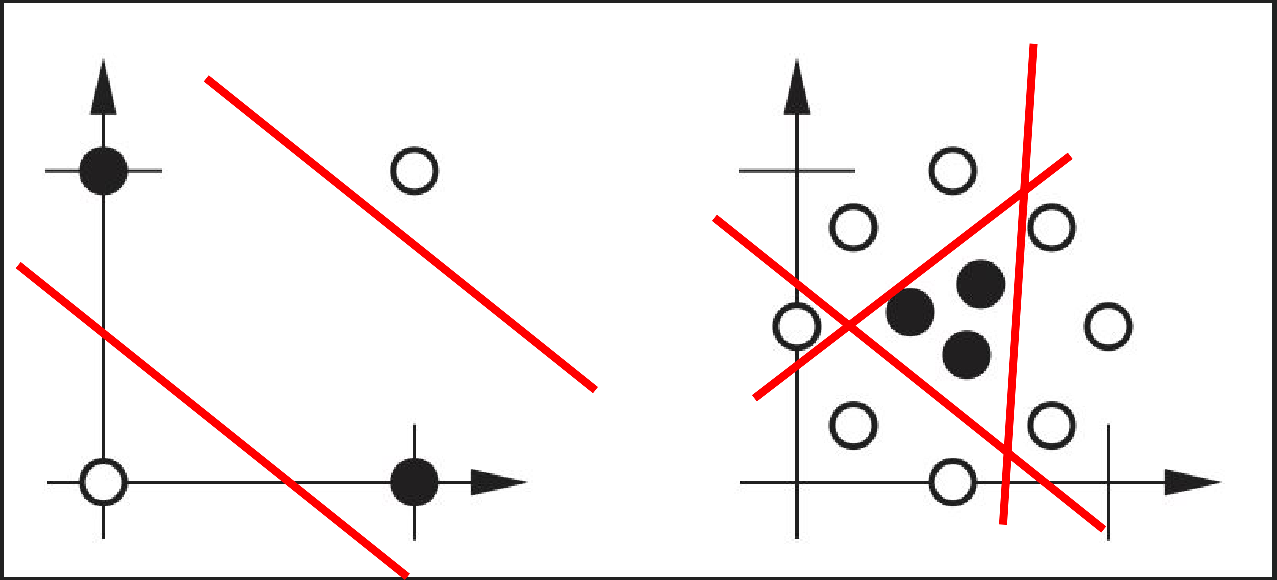


O perceptron de Rosenblatt

Encontra classificações linearmente separáveis



Problemas não separáveis linearmente

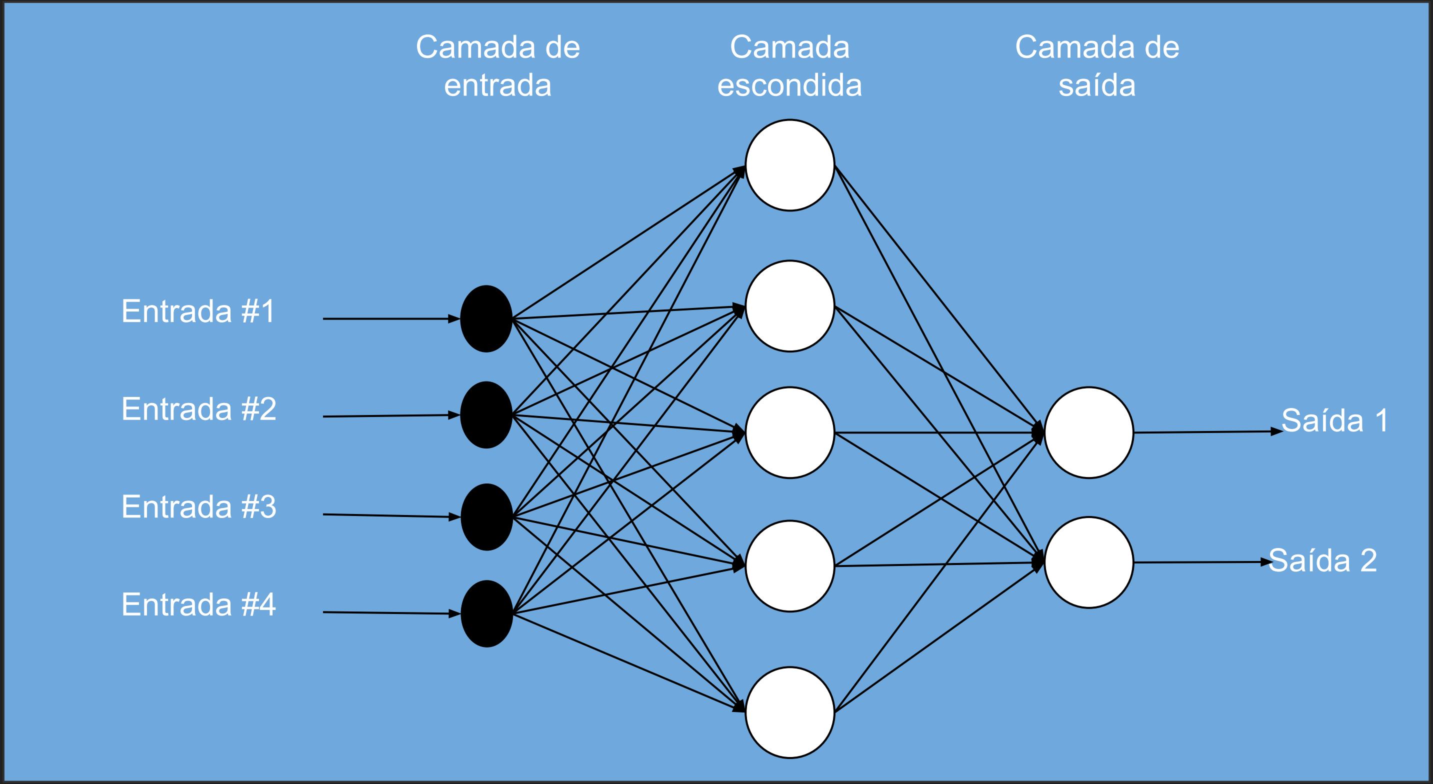


Redes Neurais: tipos de arquiteturas

Instituto de Computação

- as arquiteturas nas quais o ***sinal é transmitido*** em uma ***única direção*** e por este motivo são chamadas de **Redes Neurais Alimentadas para Frente** (do inglês **Feedforward Neural Networks**) ou ***MLP***.

Redes Neurais: arquitetura de múltiplas camadas



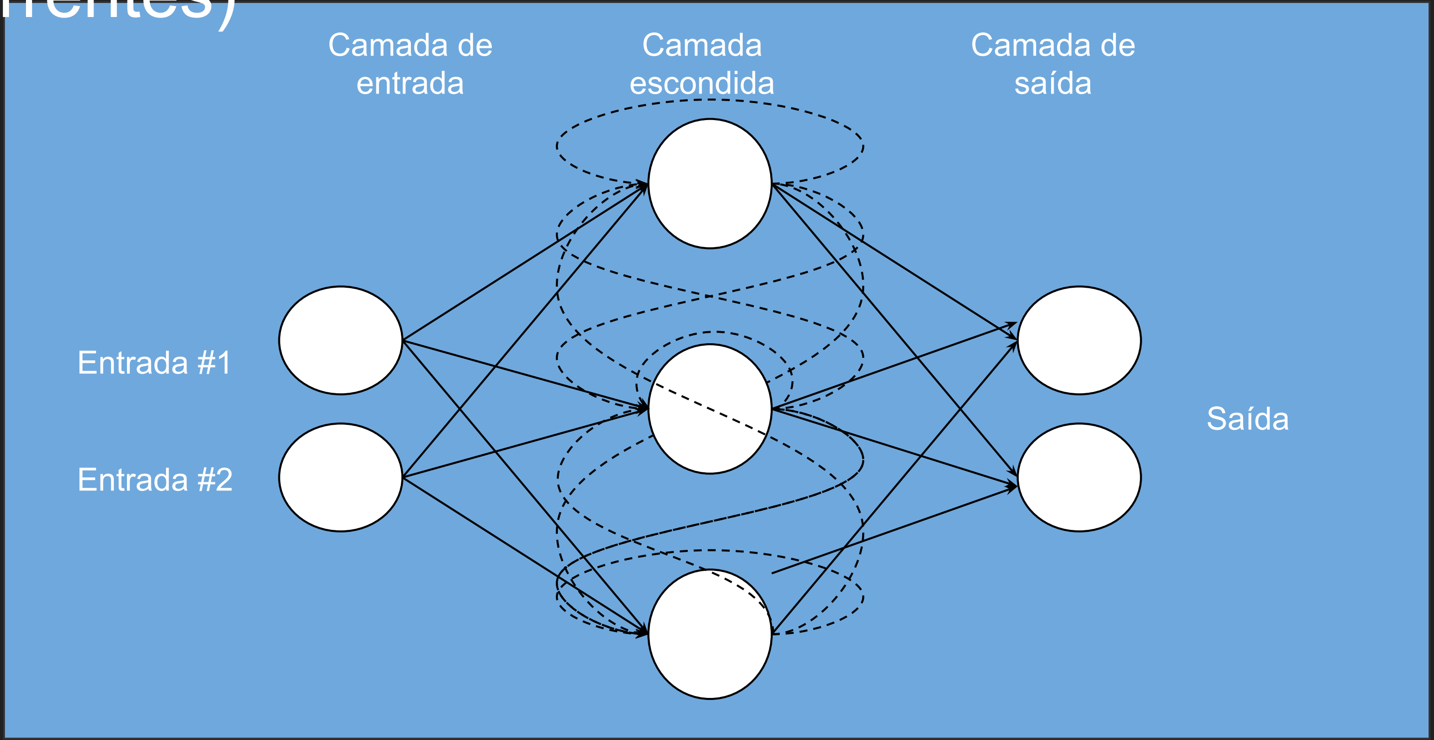
Redes Neurais: tipos de arquiteturas

- as arquiteturas nas quais o sinal é transmitido em uma única direção e por este motivo são chamadas de Redes Neurais Alimentadas para Frente (do inglês Feedforward Neural Networks) ou MLP.

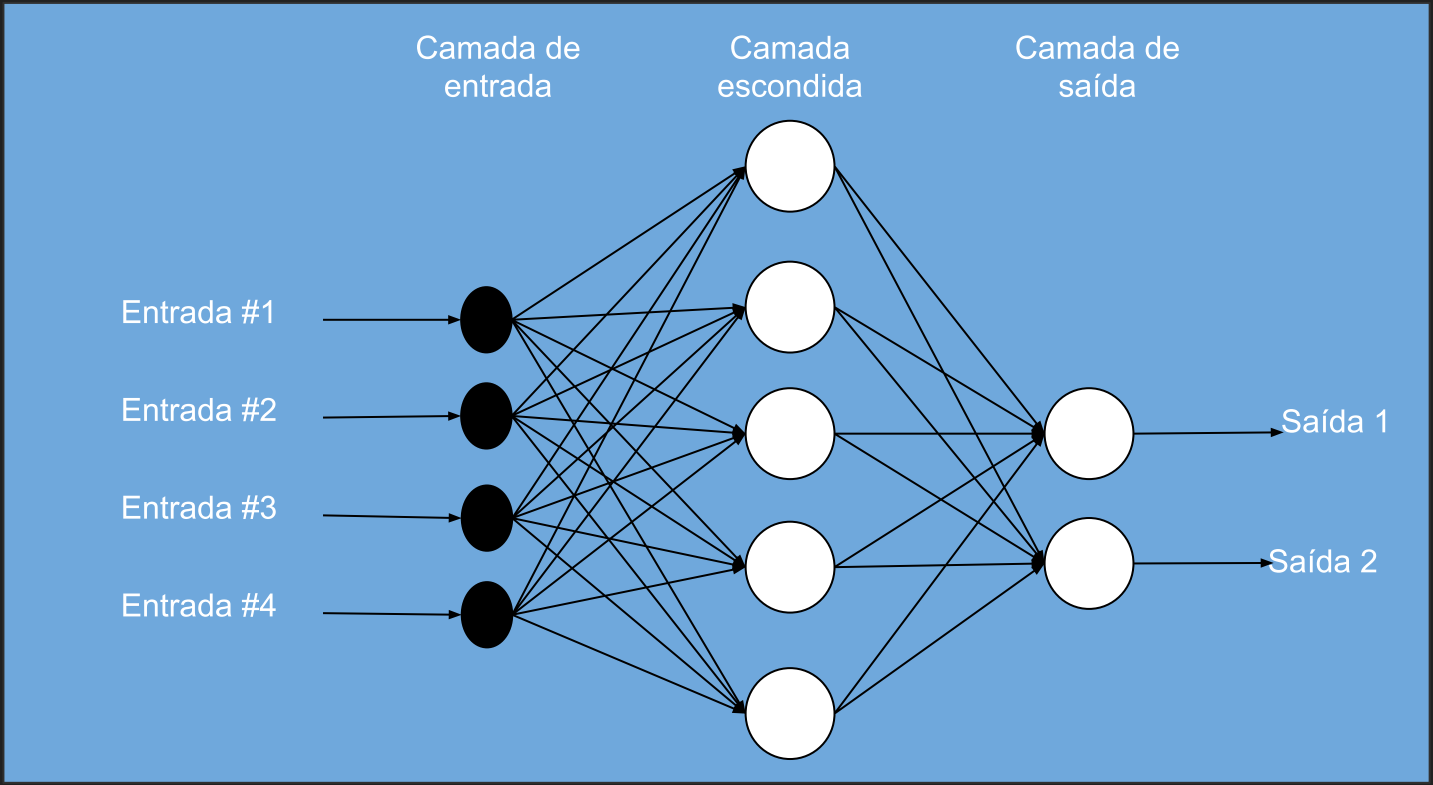
- as arquiteturas nas quais há neurônios cujas saídas ***realimentam*** a si próprios e a outros neurônios de maneira a criar ***ciclos***. Por esse motivo são chamadas **Redes Neurais de Retroalimentação ou ainda Redes Neurais Recorrentes** (do inglês **Recurrent Neural Networks**).

- Entre outras

Redes Neurais: arquitetura com retroalimentação (recorrentes)



Redes Neurais: arquitetura de múltiplas camadas



MLP: propagação

procedure MLPFORWARD(X, W)

$\quad x(0) \leftarrow X.$

$\quad c \leftarrow 1.$

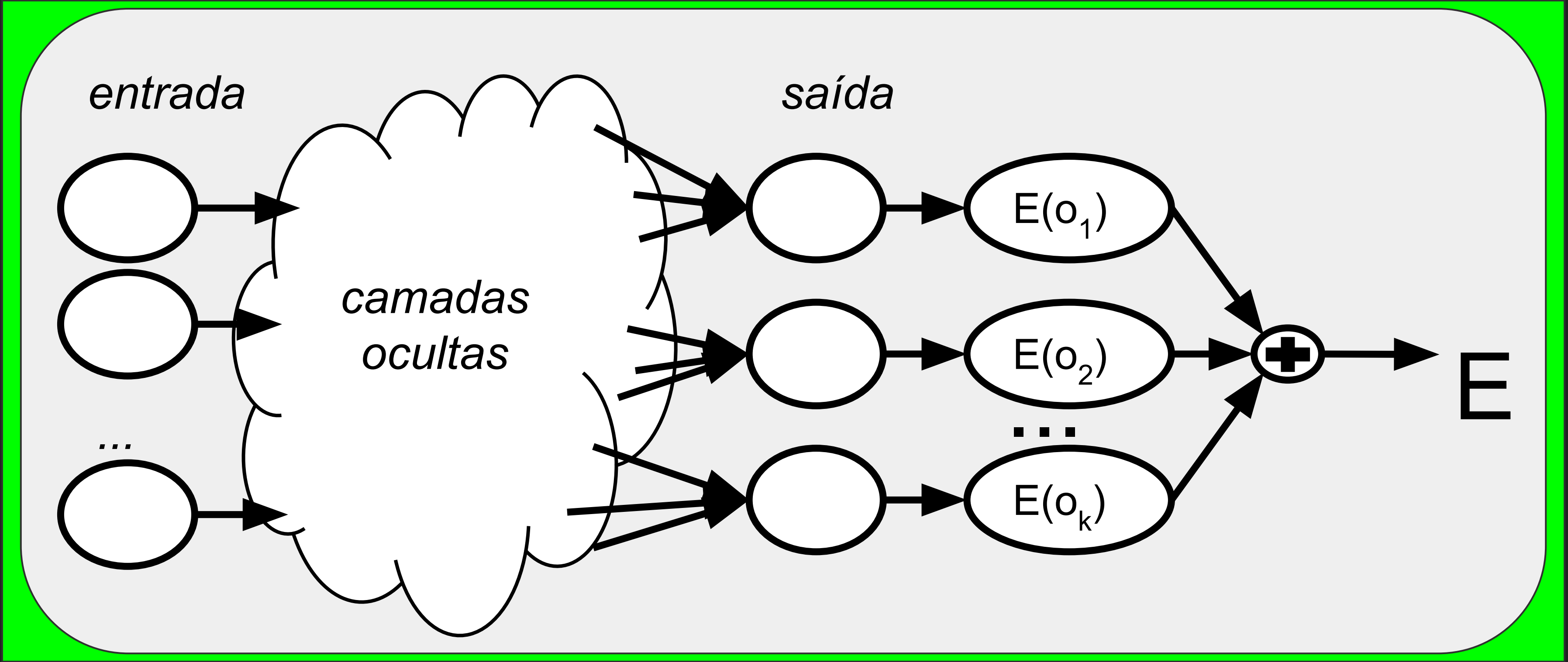
$\quad$ **for** $c <= L+1$ **do**

$\qquad z(c) \leftarrow W(c-1)x(c-1)$

$\qquad x(c) \leftarrow f(z(c))$

$\quad$ **return** $o \leftarrow x(L+1)$

Cálculo do erro



Métrica de erro:

Mean squared error (MSE):

$$\mathcal{L}(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Sejam:
m : o número de amostras de treinamento;
x(i): i-ésima amostra de treinamento;
y(i): saída desejada para a amostra i;
Θ o vetor com os parâmetros que se deseja otimizar;
o(i): saída produzida pela rede;

Cross entropy:

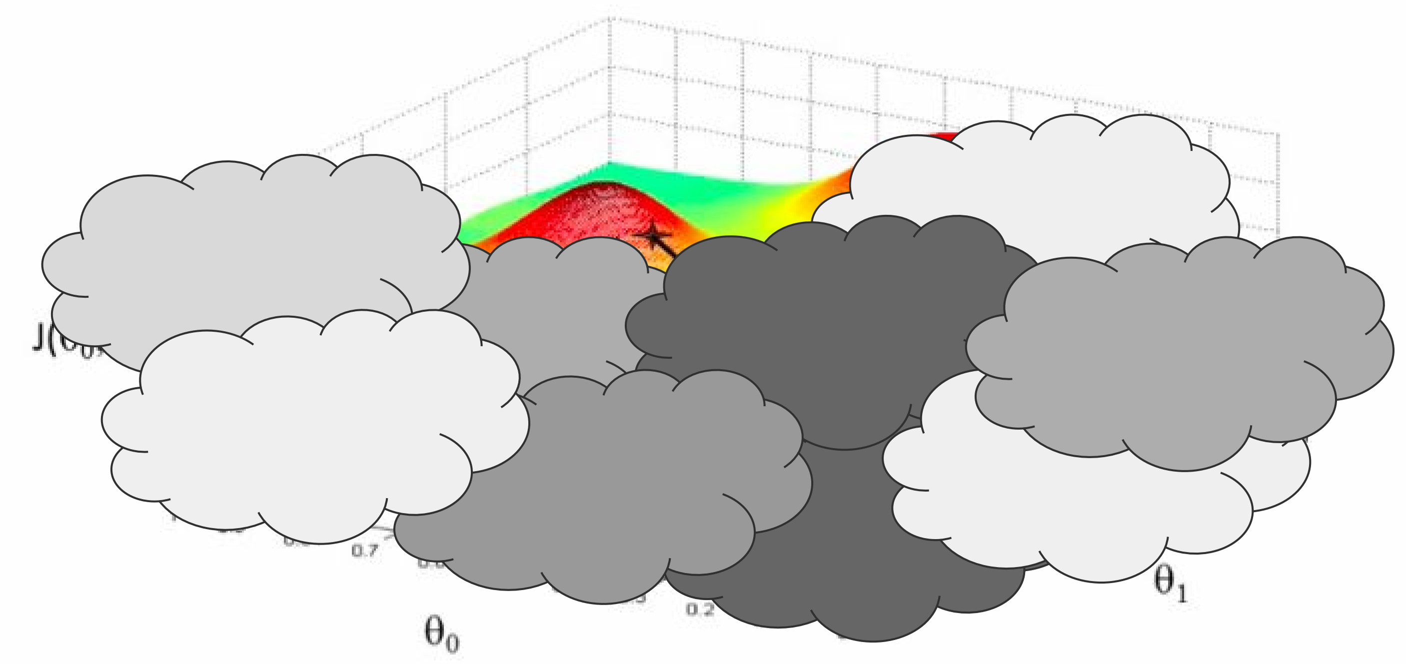
$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right]$$

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\boxed{\sum_{j=1}^{m} y_{ij}\log(p_{ij})}$$

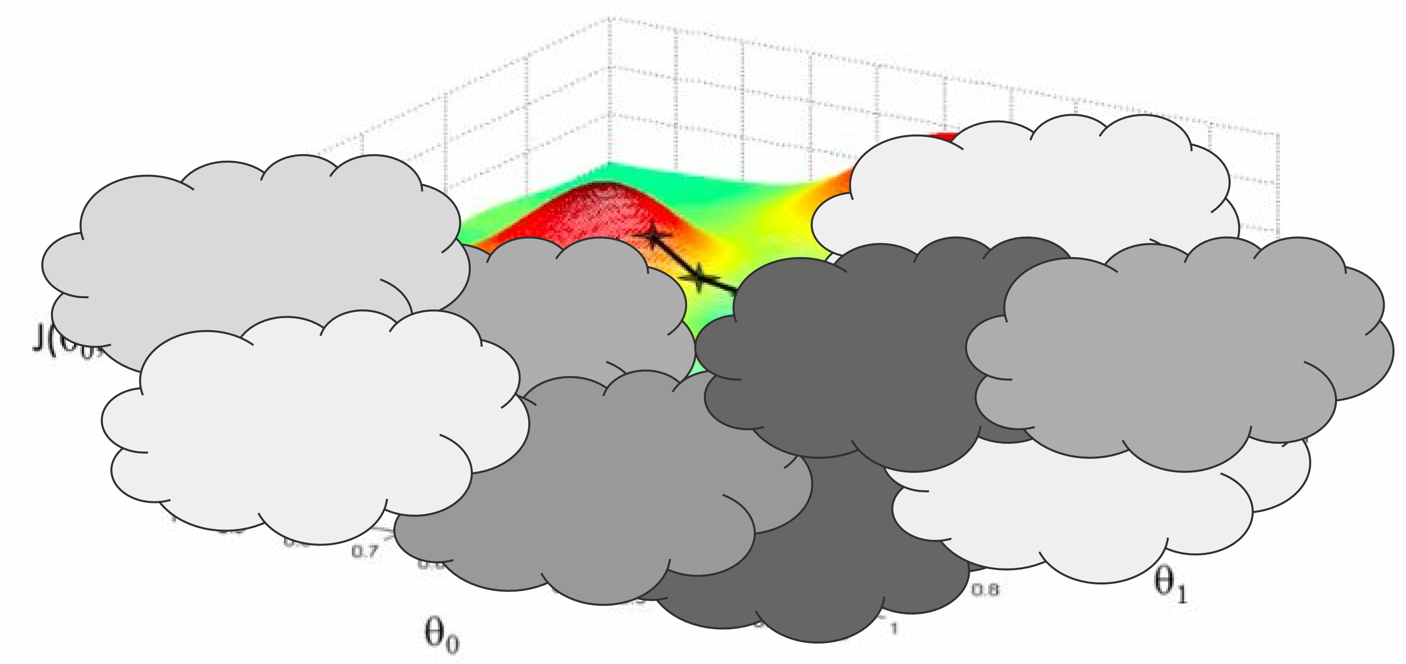
$$\log(p_{ij})$$

Sejam:
p : distribuição real
q: distribuição estimada
i: índice da amostra de treinamen
j: índice das classes;

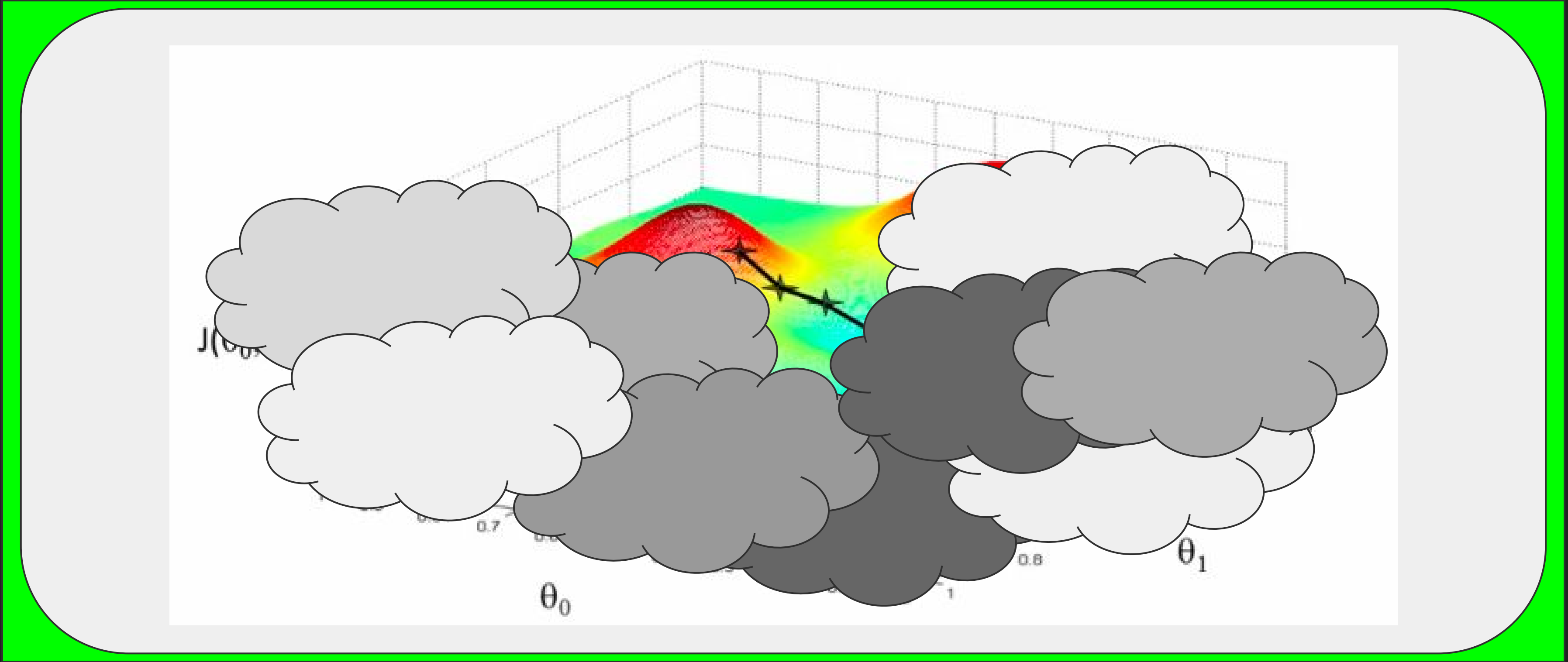
Otimização: Gradiente descendente



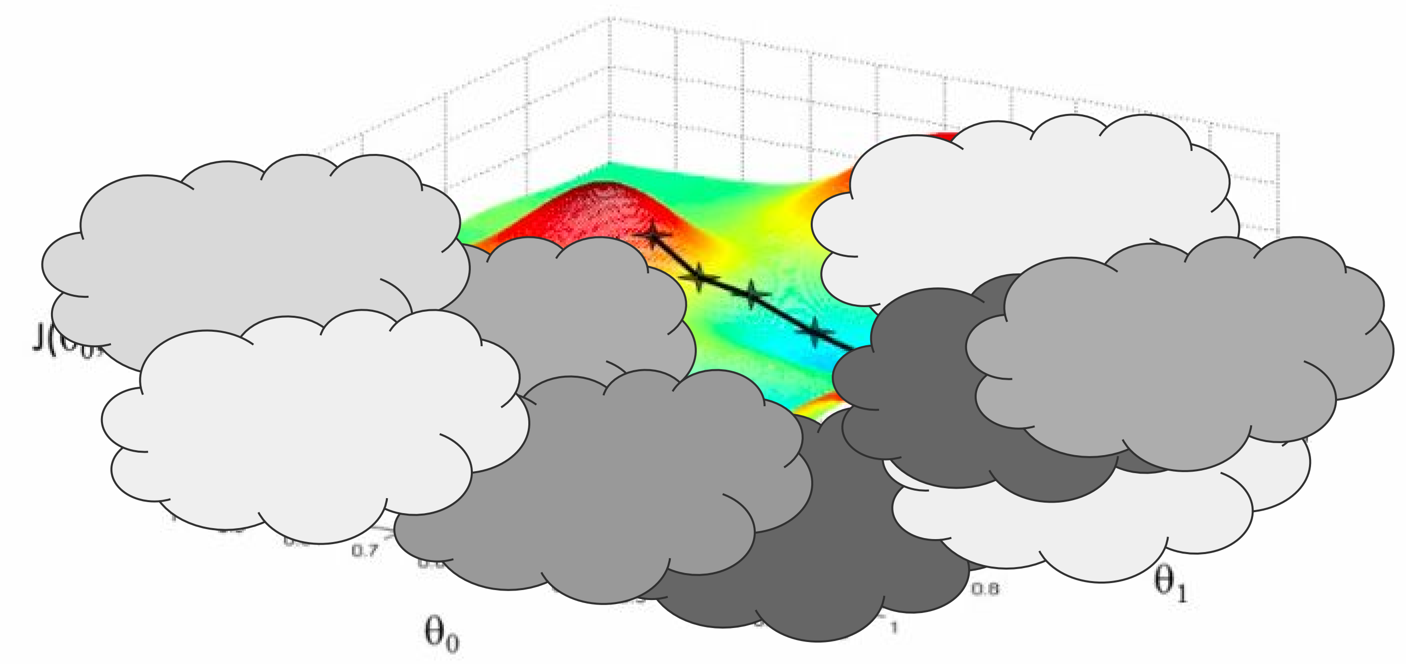
Otimização: Gradiente descendente



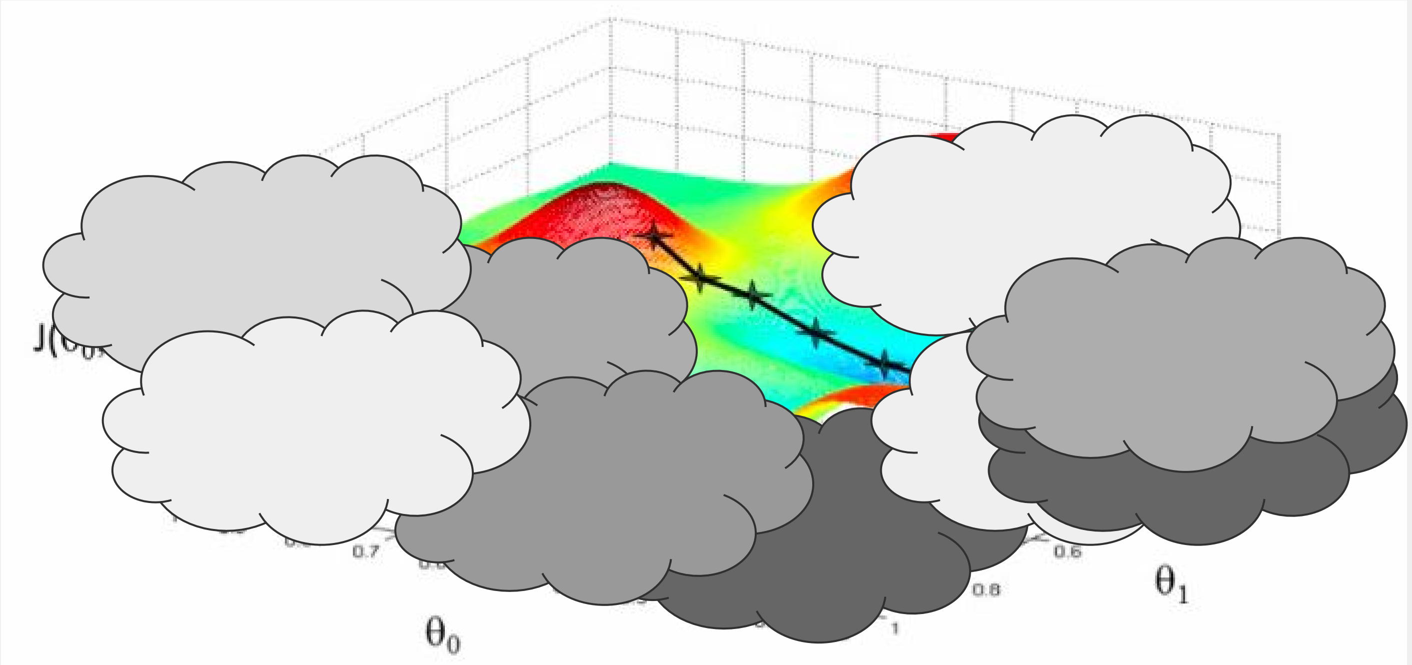
Otimização: Gradiente descendente



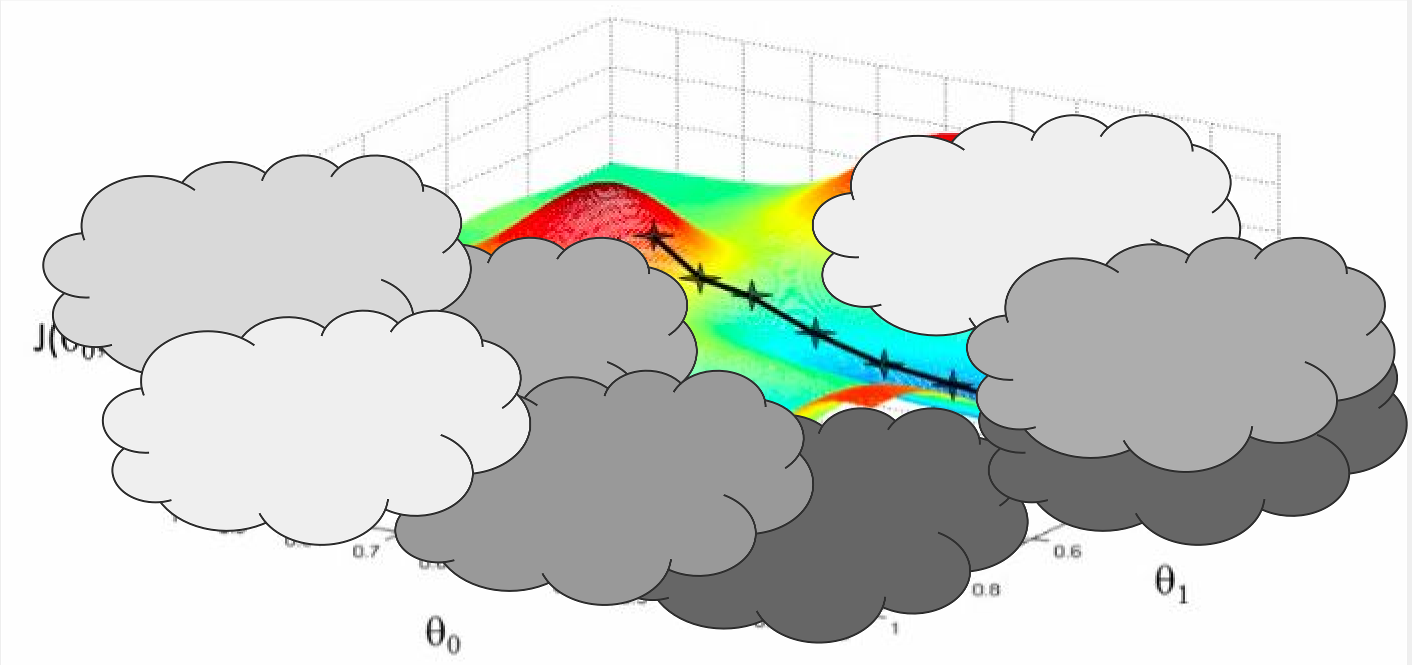
Otimização: Gradiente descendente



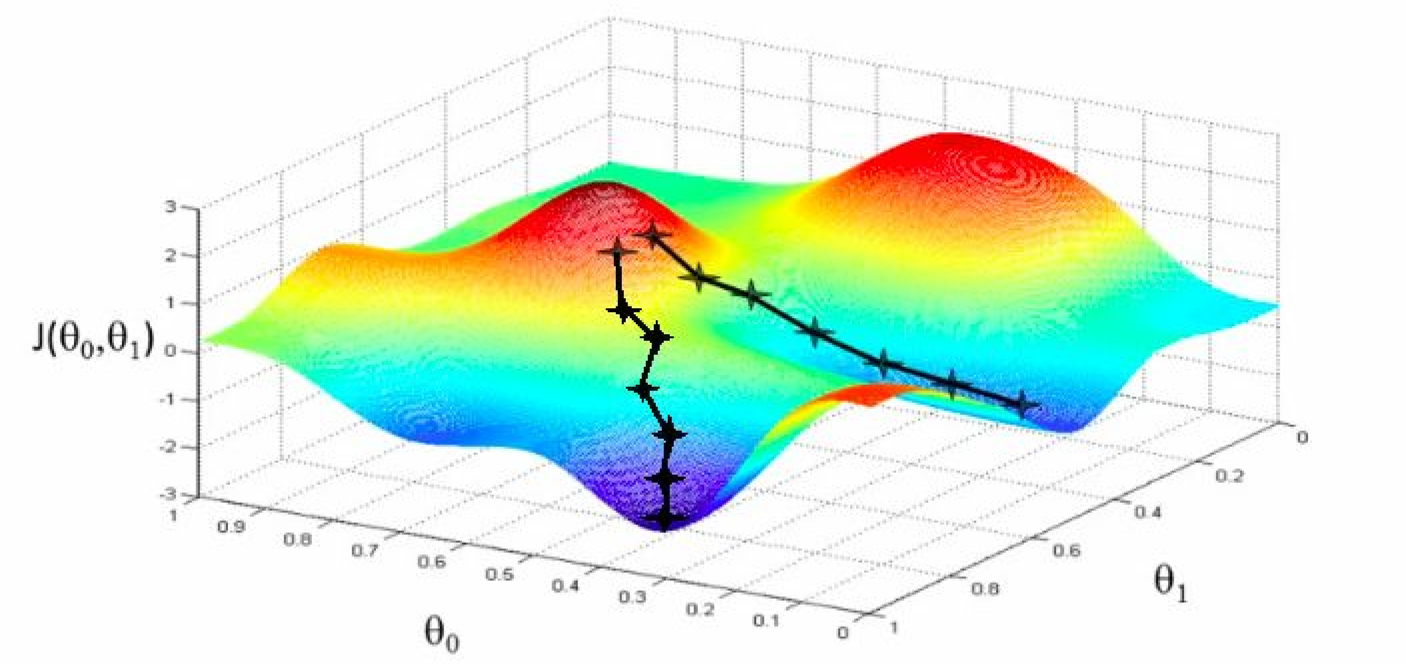
Otimização: Gradiente descendente



Otimização: Gradiente descendente



Otimização: Gradiente descendente



Otimização: Gradiente descendente

Queremos modificar os parâmetros para diminuir o erro!

Seja:

- Θ o vetor com os parâmetros que se deseja otimizar;
- L a função de erro;
- $0 < \alpha <= 1$, é a taxa de aprendizado (learning rate) ;

Algoritmo gradiente descendente:

$$\Theta(t+1) = \Theta(t) - \alpha \; d/d\Theta \; L(\Theta_n)$$

Minibatch Stochastic Gradient Descent Training

Input: Function $f(\mathbf{x};\theta)$ parameterized with parameters θ.
Input: Training set of inputs $\mathbf{x_1},\ldots,\mathbf{x_n}$ and outputs $\mathbf{y_1},\ldots,\mathbf{y_n}$.
Input: Loss function L.
while stopping criteria not met **do**
    Sample a minibatch of m examples $\{(\mathbf{x_1},\mathbf{y_1}),\ldots,(\mathbf{x_m},\mathbf{y_m})\}$
    $\hat{\mathbf{g}} \leftarrow 0$
    for $i = 1$ to m **do**
        Compute the loss $L(f(\mathbf{x_i};\theta),\mathbf{y_i})$
        $\hat{\mathbf{g}} \leftarrow \hat{\mathbf{g}} +$ gradients of $\frac{1}{m}L(f(\mathbf{x_i};\theta),\mathbf{y_i})$ w.r.t θ
    $\theta \leftarrow \theta + \eta_k\hat{\mathbf{g}}$
return θ

Gradiente descendente e MLP

$$\nabla E(t) = \left\langle \frac{\partial E(t)}{\partial w_{1,1}(t)}, \frac{\partial E(t)}{\partial w_{1,2}(t)}, \frac{\partial E(t)}{\partial w_{1,\dots}}, \frac{\partial E(t)}{\partial w_{2,1}(t)}, \frac{\partial E(t)}{\partial w_{2,2}(t)}, \frac{\partial E(t)}{\partial w_{2,\dots}}, \cdots, \frac{\partial E(t)}{\partial w_{n,1}(t)}, \frac{\partial E(t)}{\partial w_{n,2}(t)}, \frac{\partial E(t)}{\partial w_{n,\dots}}, \right\rangle$$

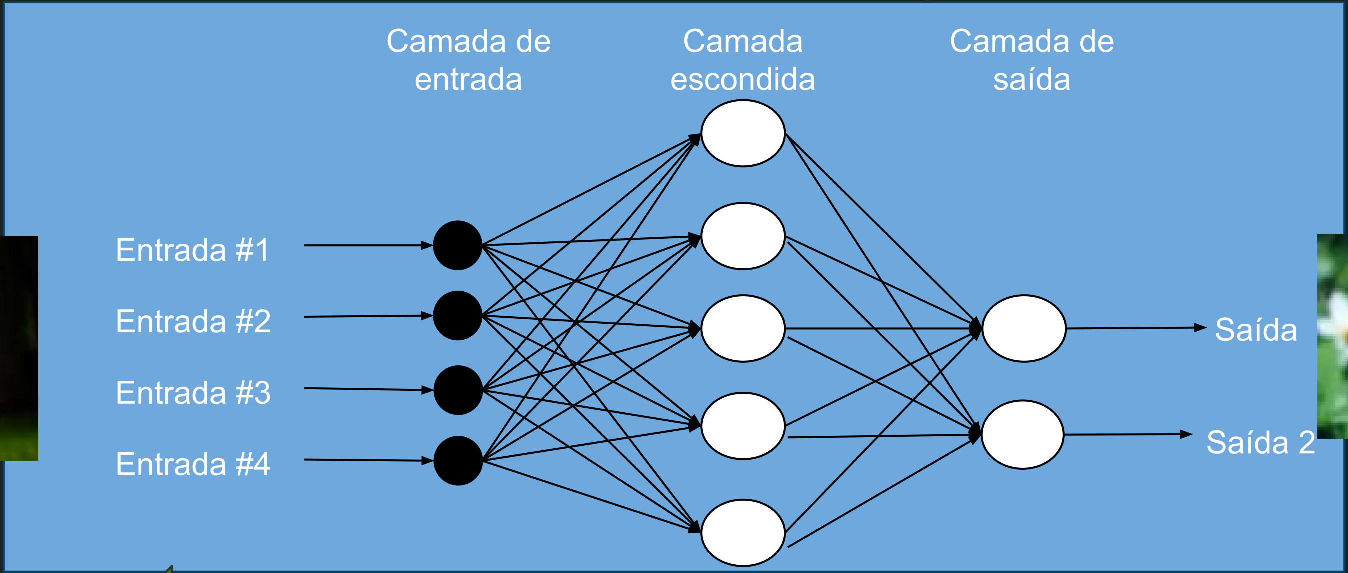
$$W(t+1) = W(t) - \alpha \nabla E(t)$$

$$w_{ji}(t+1) = w_{ji}(t) - \alpha \frac{\partial E(t)}{\partial w_{ji}(t)}$$

Treinamento de MLPs: Back propagation

Instituto de Computação

Forward propagation

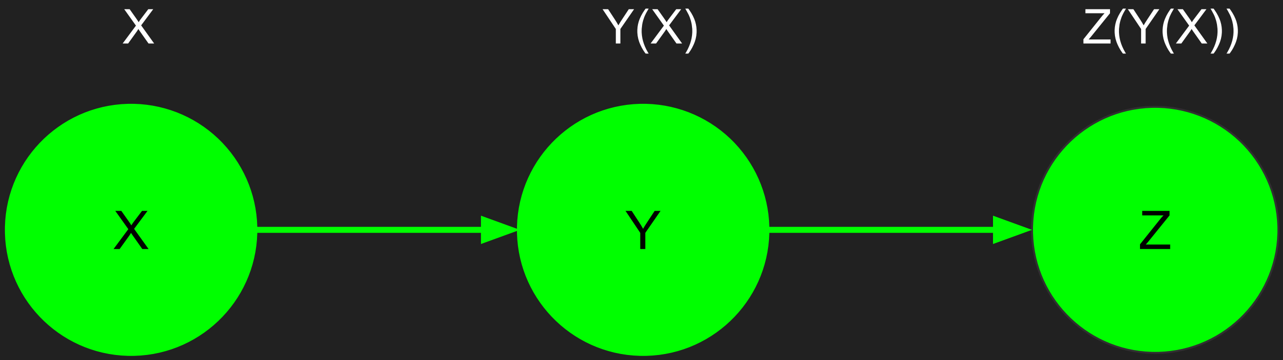


ERRO

Backward propagation

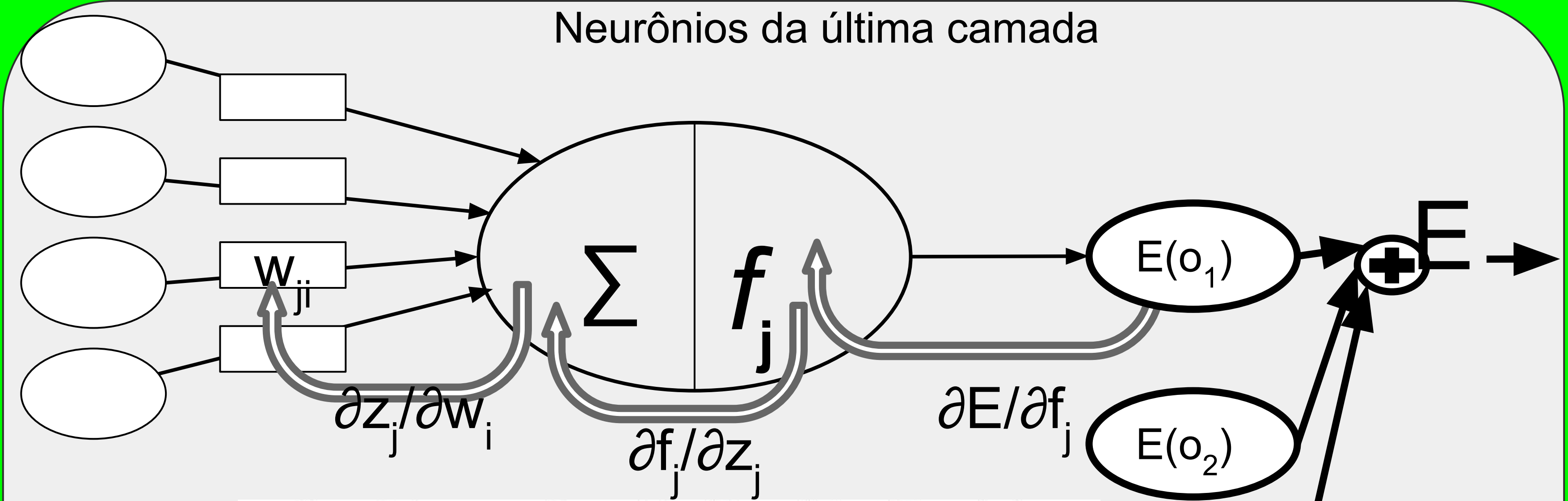
Treinamento de MLPs: Back propagation

Regra da cadeia aplicada a sequência de processamentos:



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

Treinamento de MLPs: Back propagation



$$\frac{\partial E(t)}{\partial w_{ji}(t)} = \frac{\partial E(t)}{\partial f_j(t)} \frac{\partial f_j(t)}{\partial z_j(t)} \frac{\partial z_j(t)}{\partial w_{ji}(t)}$$

Treinamento de MLPs: Back propagation

$$W(t+1) = W(t) - \alpha \nabla E(t)$$

$$\frac{\partial E(t)}{\partial w_{ji}(t)} = \boxed{\frac{\partial E(t)}{\partial f_j(t)} \frac{\partial f_j(t)}{\partial z_j(t)}} \boxed{\frac{\partial z_j(t)}{\partial w_{ji}(t)}}$$

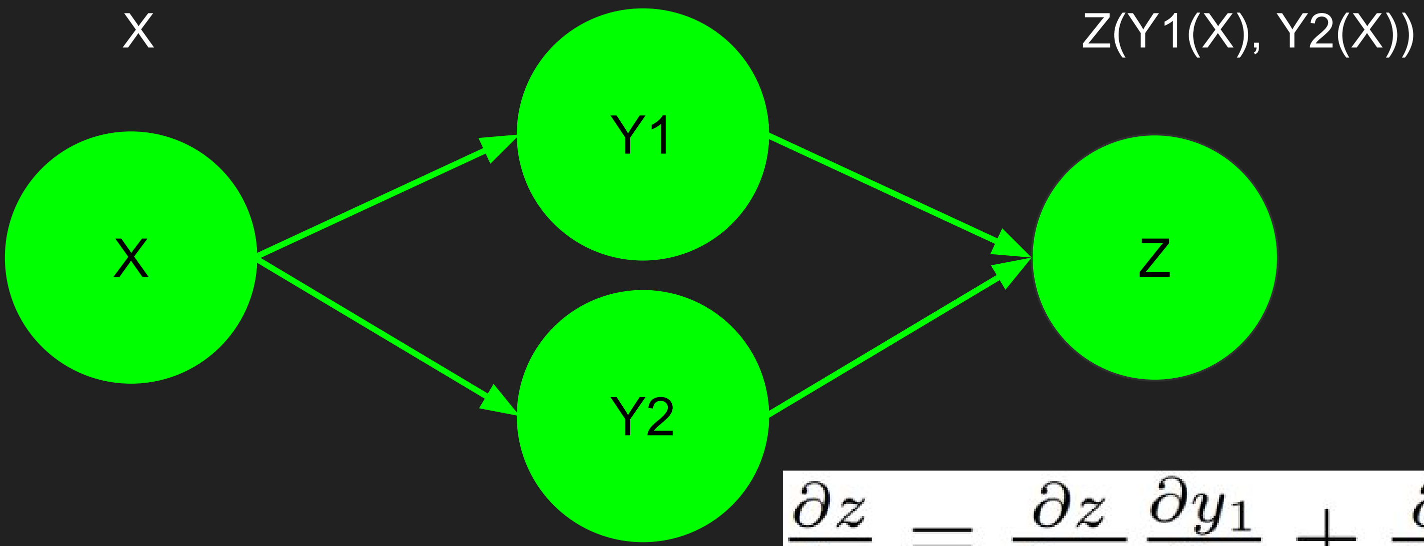
$$\boxed{\delta_j = \frac{\partial E(t)}{\partial z_j(t)}}$$

$$\frac{\partial E(t)}{\partial w_{ji}(t)} = \delta_j(t) \boxed{x_{ji}(t)}$$

$$W(t+1) = W(t) - \alpha \delta_j(t) * X_j$$

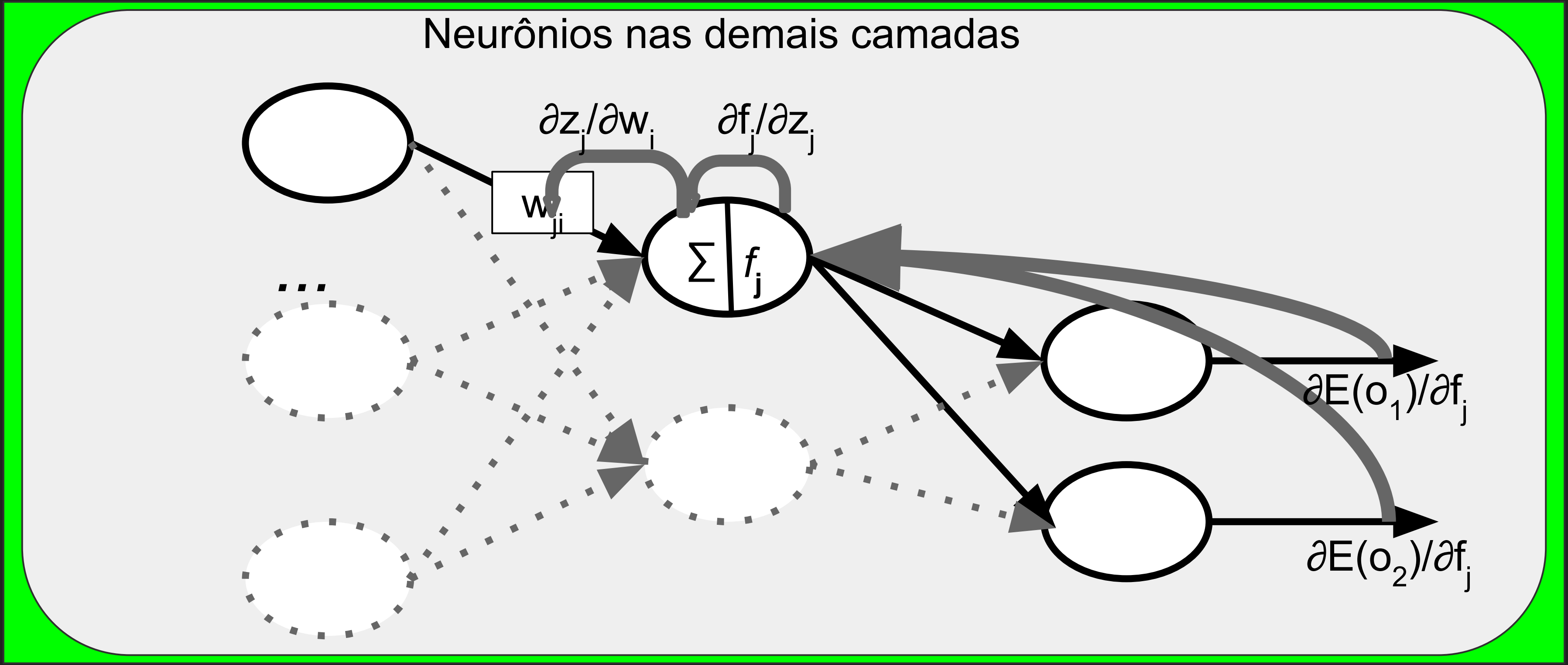
Treinamento de MLPs: Back propagation

Regra da cadeia aplicada a sequência de processamentos com 2 caminhos:



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Treinamento de MLPs: Back propagation



Treinamento de MLPs: Back propagation

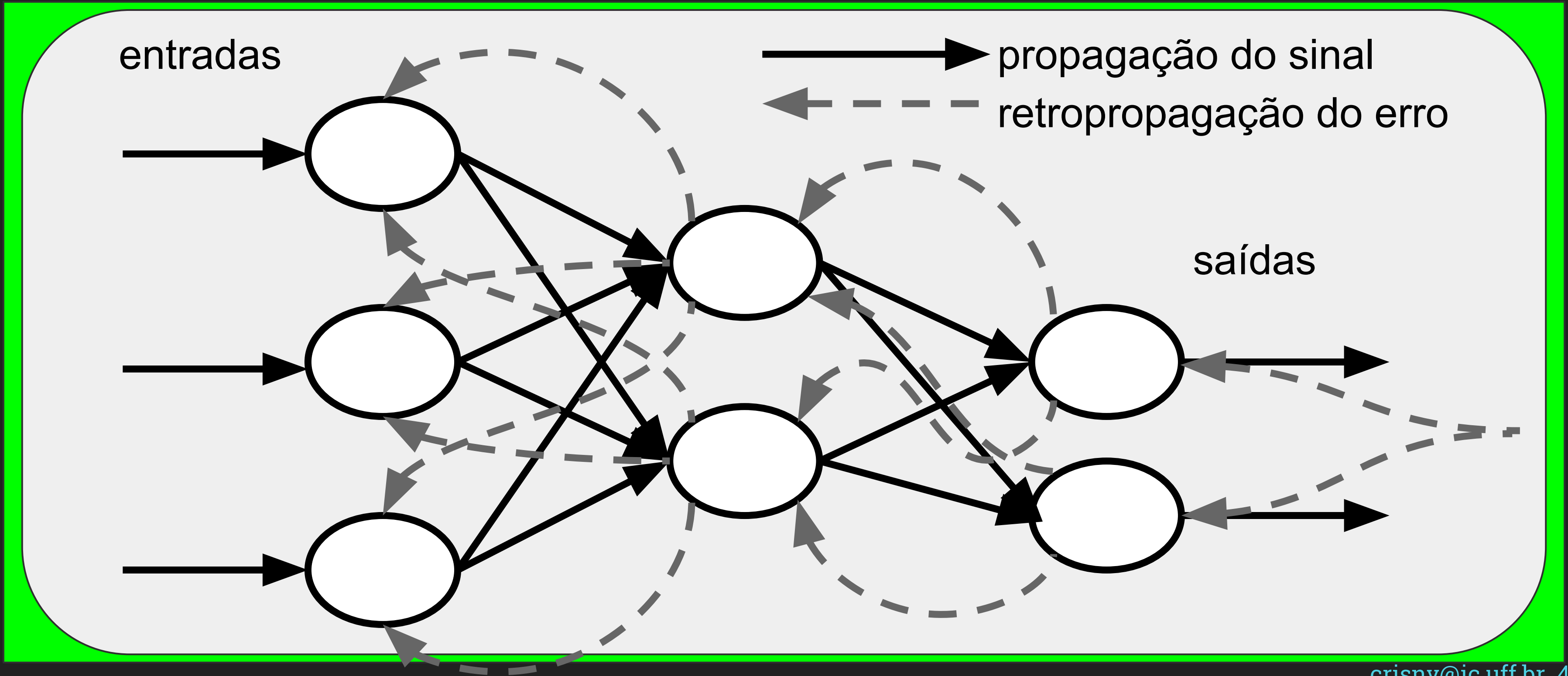
$$\frac{\partial E(t)}{\partial f_j(t)} = \sum_{k=1}^{n} \frac{\partial E(t)}{\partial x_{ki}(t)} = \sum_{k=1}^{n} \boxed{\frac{\partial E(t)}{\partial z_k(t)}} \boxed{\frac{\partial z_k(t)}{\partial x_{ki}(t)}} = \sum_{k=1}^{n} \frac{\partial E(t)}{\partial z_k(t)} \boxed{w_{ki}(t)} = \sum_{k=1}^{n} \delta_k(t) w_{ki}(t)$$

$$\delta_j(t) = \frac{\partial f_j(t)}{\partial z_j(t)} \sum_{k=1}^{n} \delta_k(t) w_{ki}(t)$$

$$\frac{\partial E(t)}{\partial w_{ji}(t)} = \delta_j(t) \boxed{x_{ji}(t)}$$

$$W(t+1) = W(t) - \alpha \delta_j(t) * X_j$$

Treinamento de MLPs: Back propagation



Treinamento de MLPs: Back propagation

procedure MLPBACK$(X, W0)$

  $dx(L+1) \leftarrow dE(x(L+1), y)/dx(L+1)$

  for c **from** $L+1$ **downto** 1 **do**

    $dz(c) \leftarrow df/dz(z(c)) \cdot dx(c)$

    $dx(c-1) \leftarrow W^{\top}(c-1)dz(c)$

    $dW(c-1) \leftarrow dz(c)x^{\top}(c-1)$

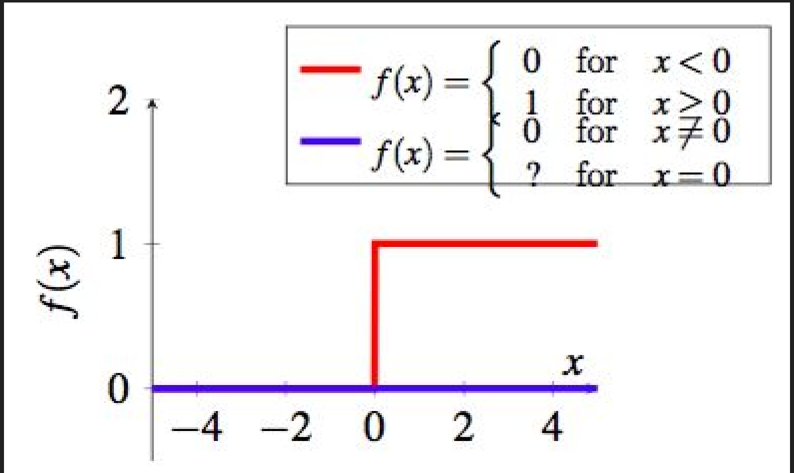
  return $dW[0 \cdots L]$

Funções de ativação

Degrau de Heaviside

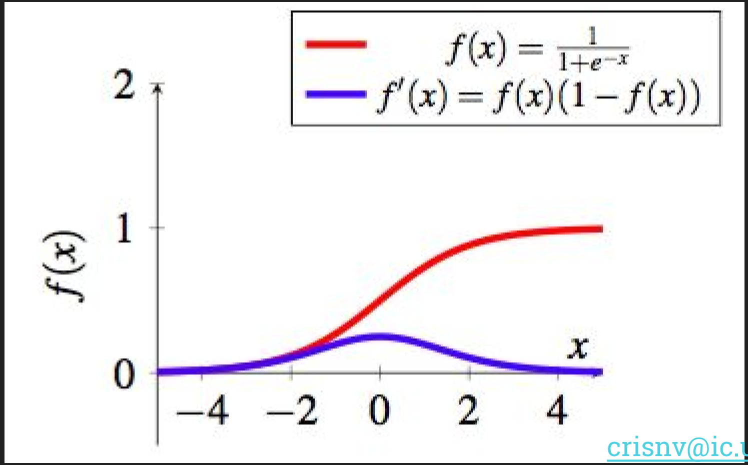
(adotada no perceptron original)

mapeia números reais para o intervalo [0,1]



Sigmoid

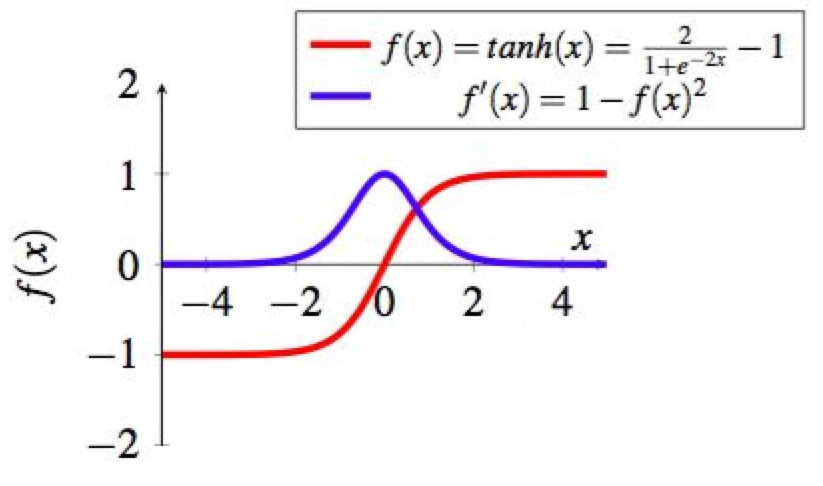
mapeia números reais para o intervalo [0,1]



Funções de ativação

Tanh

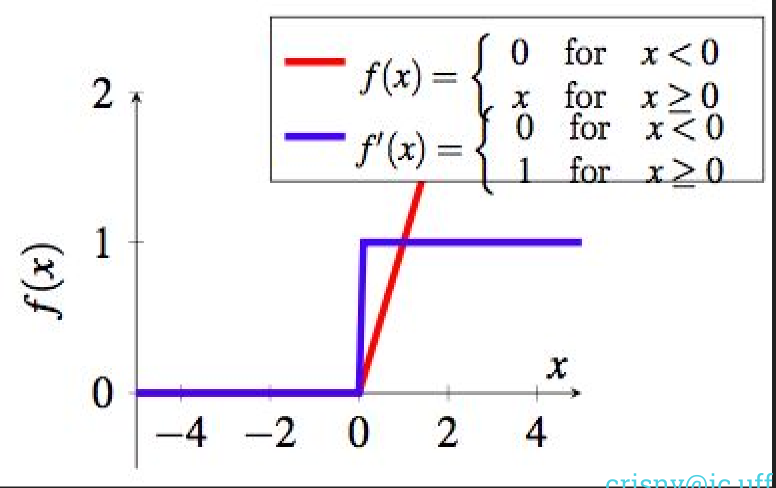
mapeia números reais para o intervalo [-1,1].



Rectified Linear Unit (ReLU)

mapeia números reais para o intervalo [0,1].

Barata computacionalmente!



Funções de ativação Softmax

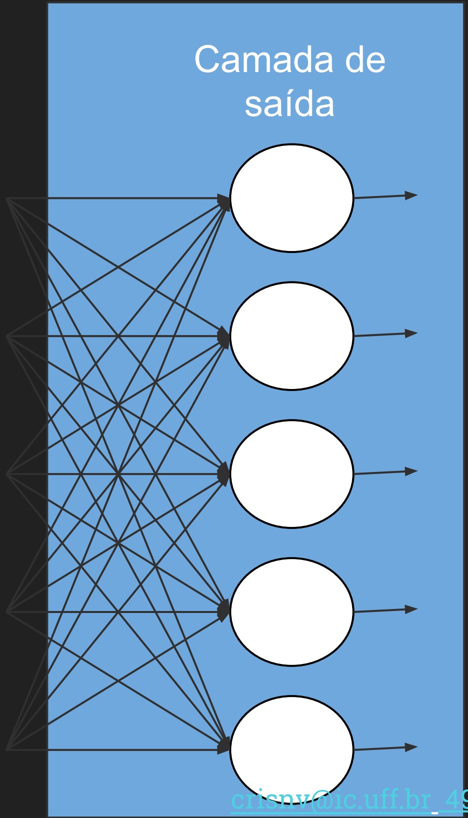
$$P(y = j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

Softmax (normalized exponential function):

Exemplo:

X = [1.0, 2.0, 3.0, 4.0, 5.0]

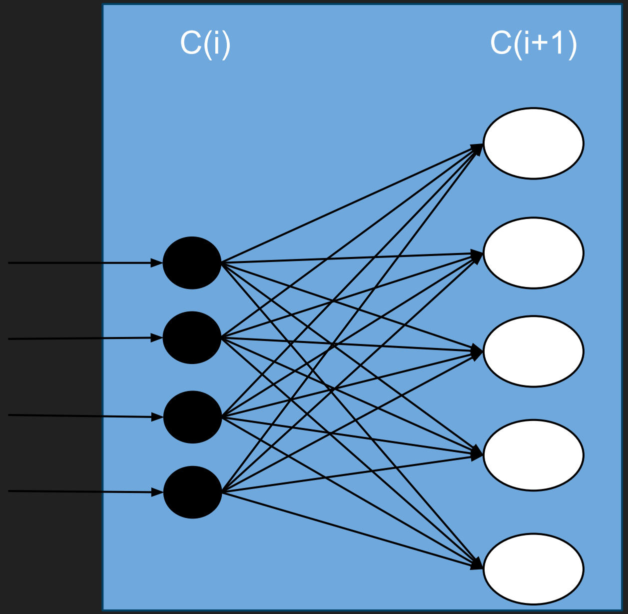
O = [0.012, 0.032, 0.086, 0.234, 0.636]



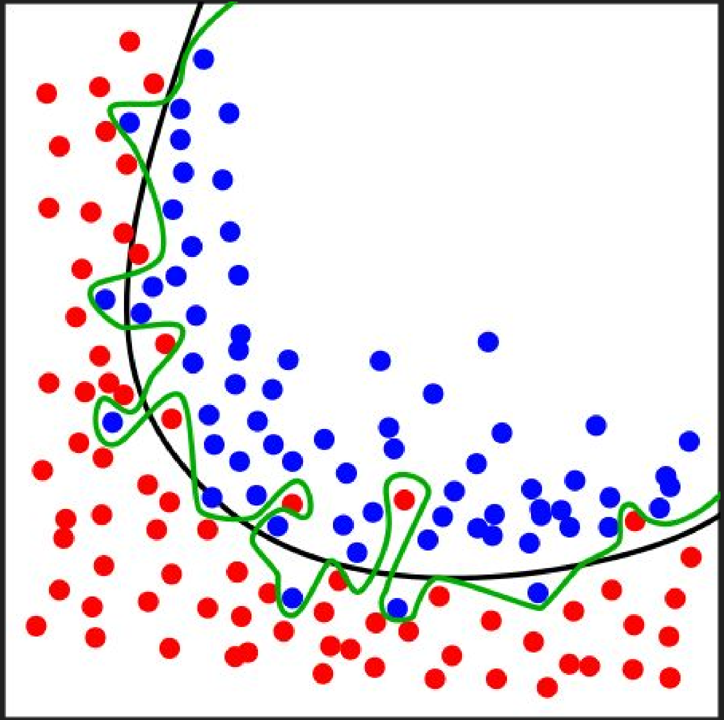
Problemas com a arquitetura MLP padrão

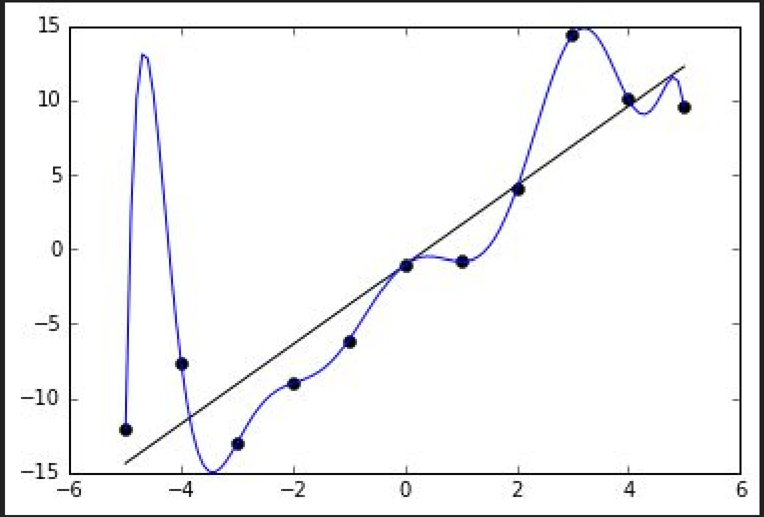
Camadas completamente conectadas (ou densas):

- Número de parâmetros cresce rápido



Overfitting





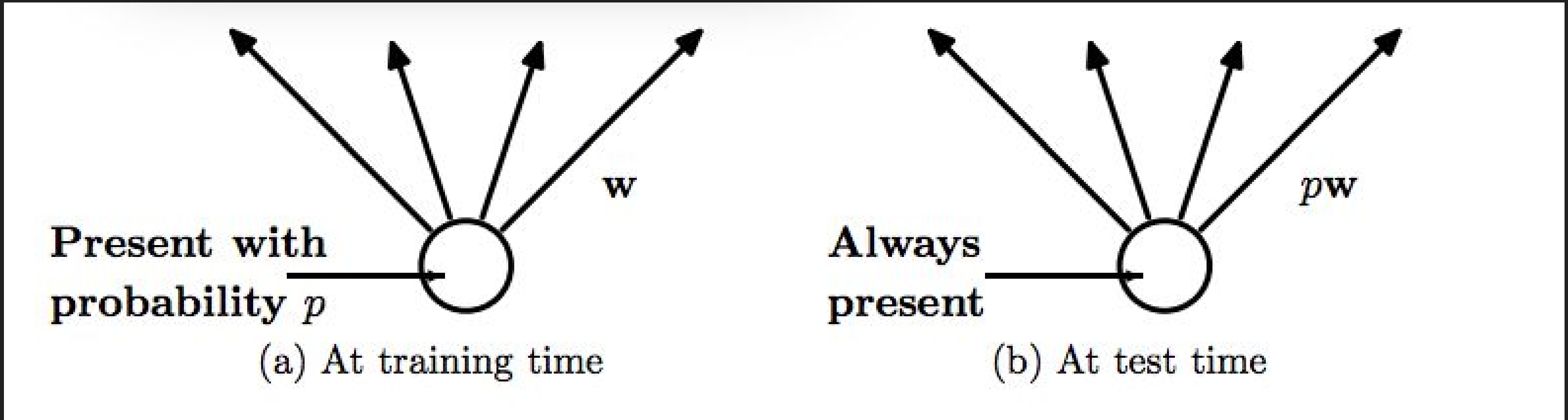
Validação cruzada

A validação cruzada é uma técnica para **avaliar a capacidade de generalização** de um modelo, a partir de um conjunto de dados não utilizado no treinamento.

Faz-se o particionamento do conjunto de dados em **subconjuntos mutualmente exclusivos**, e posteriormente, utiliza-se alguns destes subconjuntos para a estimação dos parâmetros do modelo (**dados de treinamento**) e o restante dos subconjuntos (**dados de validação ou de teste**) são empregados na validação do modelo.

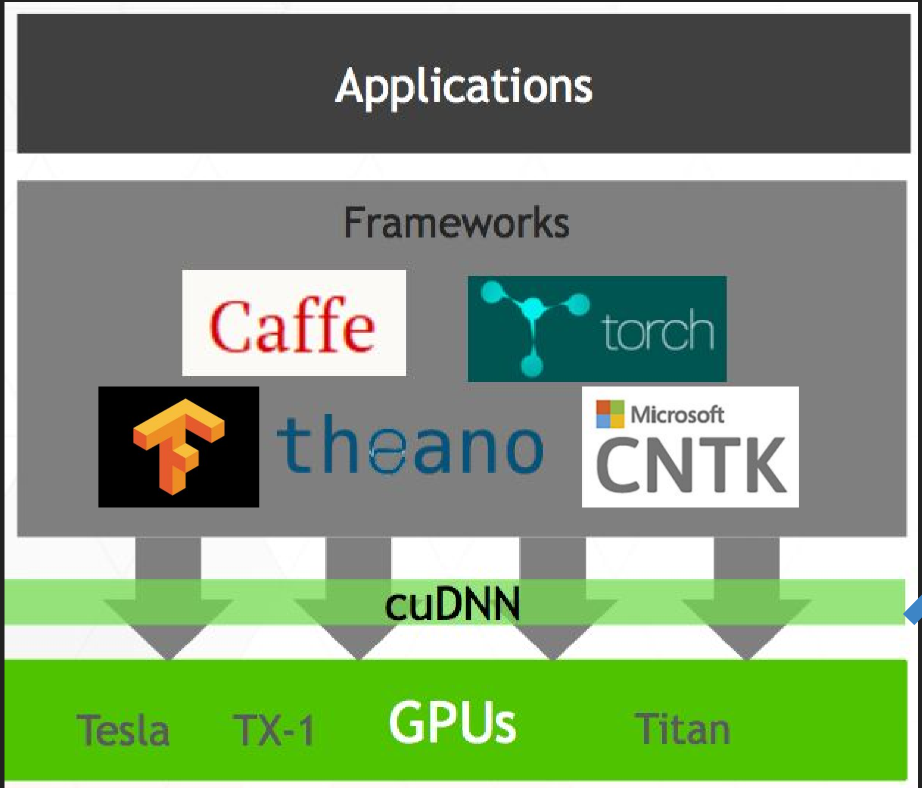
Diversas técnicas: o método **holdout** (+-2/3), o k-fold, e o leave-one-out.

Dropout



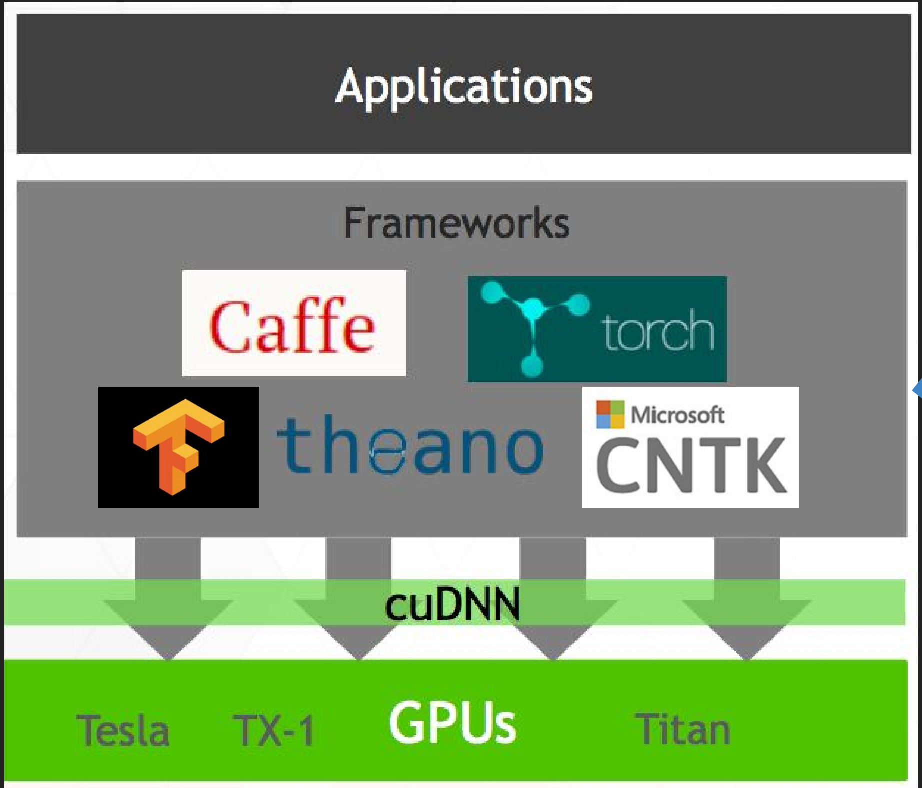
[Srivastava et al., 2014]

Hierarquia de programação:



cuDNN: biblioteca de primitivas de deep learning

Hierarquia de programação



Frameworks:

- Caffe, Torch, Theano, CNTk, Tensor Flow

Diferenciação automática

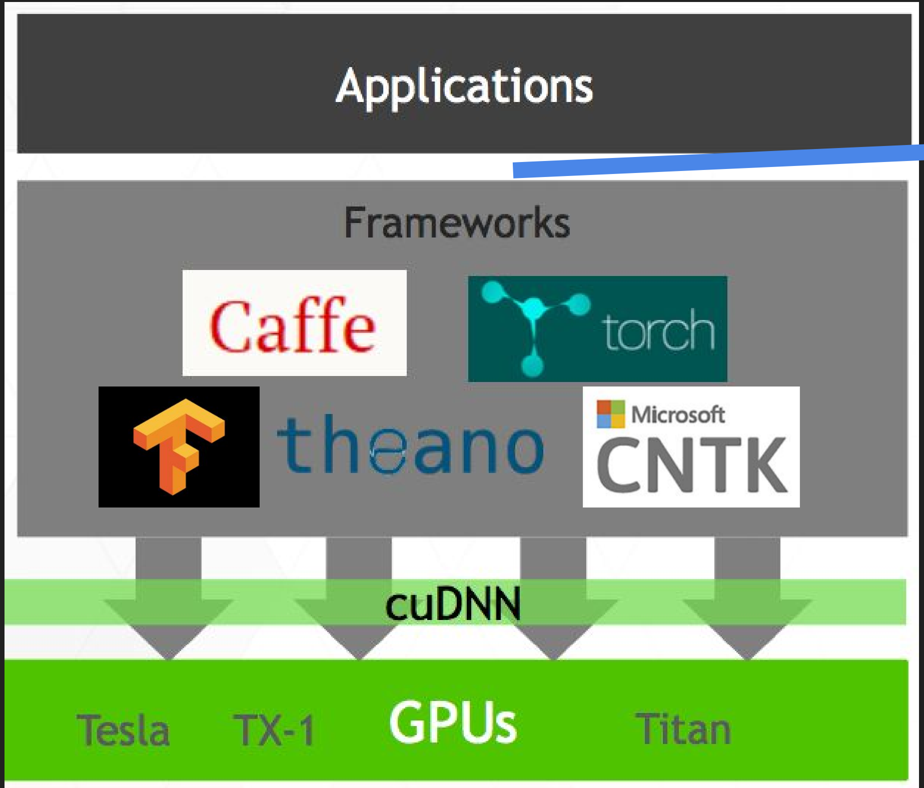
Durante a passada para frente, é realizado o cálculo de gradiente:

- derivando-se simbolicamente a função obtendo uma expressão e avaliando-a em um ponto dado; ou
- utilizando-se derivação numérica;

Assim, cada nó deve saber como realizar as operações para produção de sua saída, mas também como calcular seu gradiente em relação a suas entradas

Com isso, bibliotecas conseguem tratar um número arbitrário de camadas empilhadas

Hierarquia de programação



- Digits, Keras

Aprendizado profundo e GPUs

	redes neurais	GPUs
inerentemente paralela	sim	sim
operações de matriz	sim	sim
FLOPS	sim	sim

crisnv@ic.uff.br