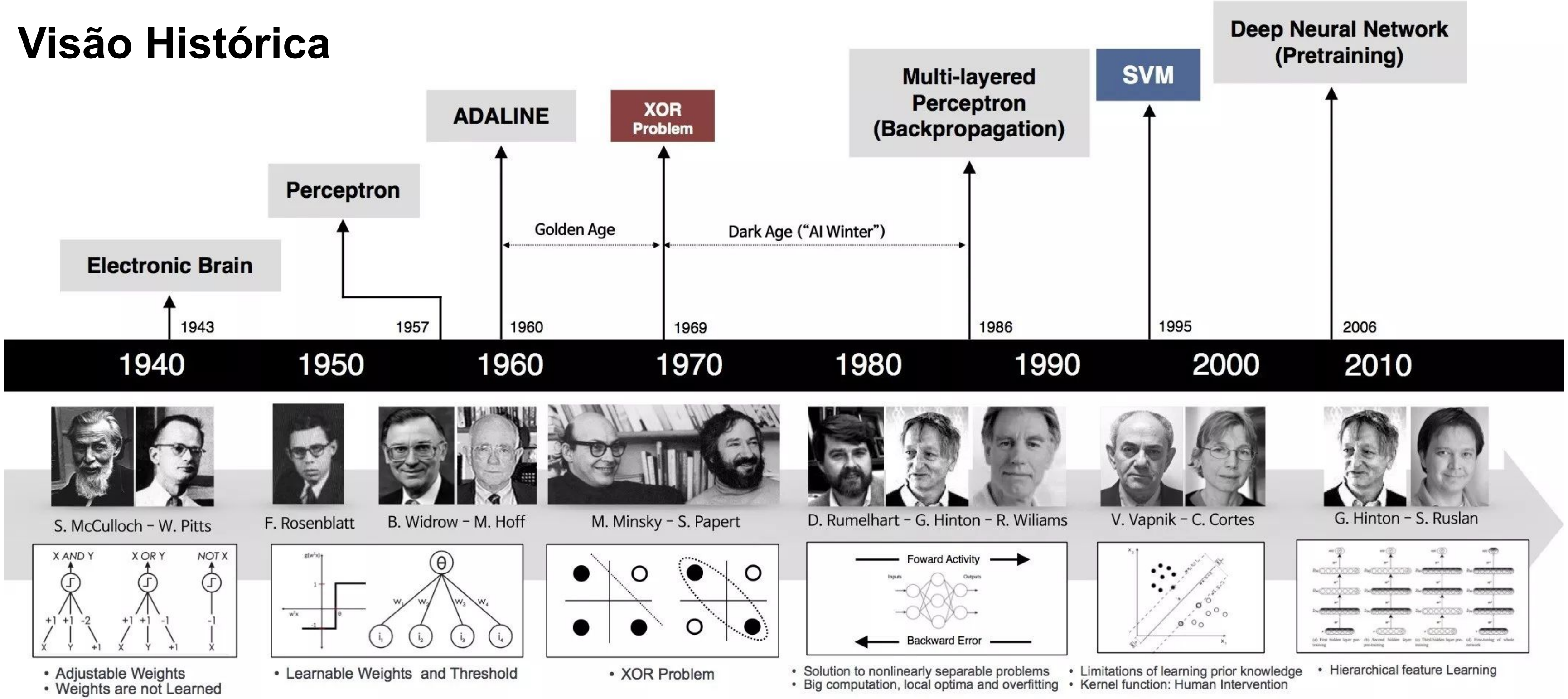
Perceptron Multicamadas (MLP)

Prof. Matheus Santos
matheus.santos@lapisco.ifce.edu.br

1. Rede Neural MLP
2. Funções de Ativação
3. *Feedforward* e *Backpropagation*
4. Função Custo e Otimizadores
5. Algoritmos de Busca Aleatória
6. Aplicações do MLP
7. Bônus: *Extreme Machine Learning* (ELM)

Rede Neural MLP

Visão Histórica

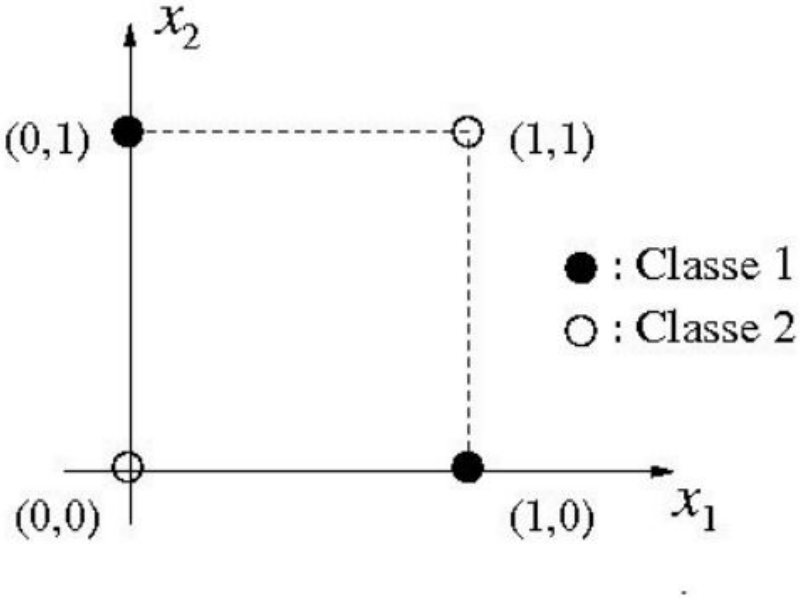


Rede Neural MLP

➢ **Problema:** Porta XOR (?)

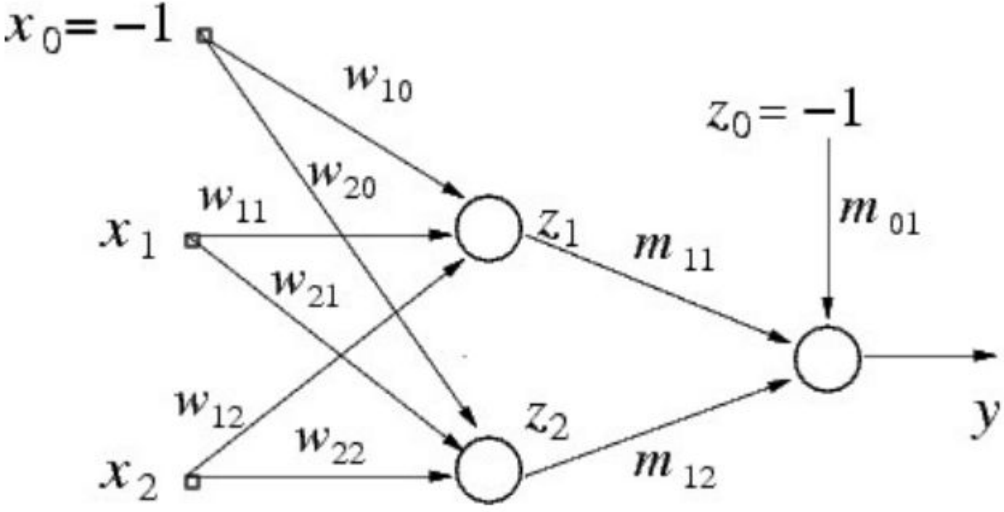
Porta XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



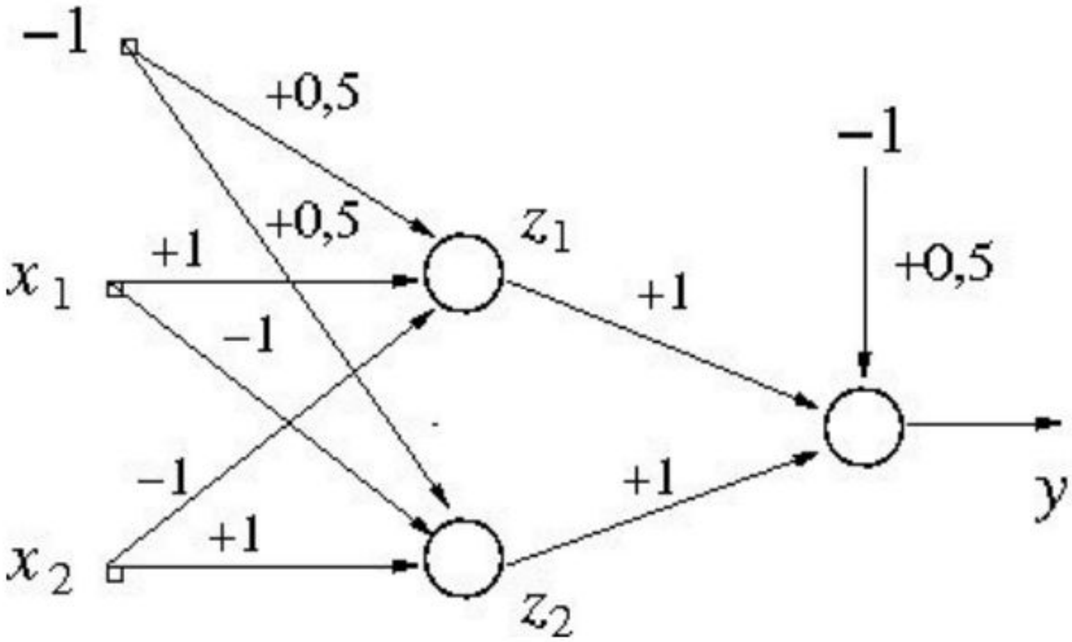
Rede Neural MLP

A porta lógica XOR é **não-linearmente separável.** Logo, uma reta não é suficiente para separar as classes. São necessários pelo menos três neurônios para descrevê-la.



Rede Neural MLP

- **Possível Solução:** Porta XOR



Rede Neural MLP

Perceptron Multicamadas (*Multilayer Perceptron*) - MLP

- **Unidades de entrada:** responsáveis pela simples passagem dos valores de entrada para os neurônios das camadas seguintes.
- **Camada(s) oculta(s):** contém neurônios responsáveis pelo processamento não-linear da informação de entrada, de modo a facilitar a resolução do problema para os neurônios da camada de saída.
- **Camada de saída:** contém neurônios responsáveis pela geração da saída da rede neural, após as entradas terem sido devidamente processadas pelos neurônios ocultos.

Rede Neural MLP

Perceptron Multicamadas (*Multilayer Perceptron*) - MLP

➢ Uma Rede MLP com 1 camada oculta é representada por:

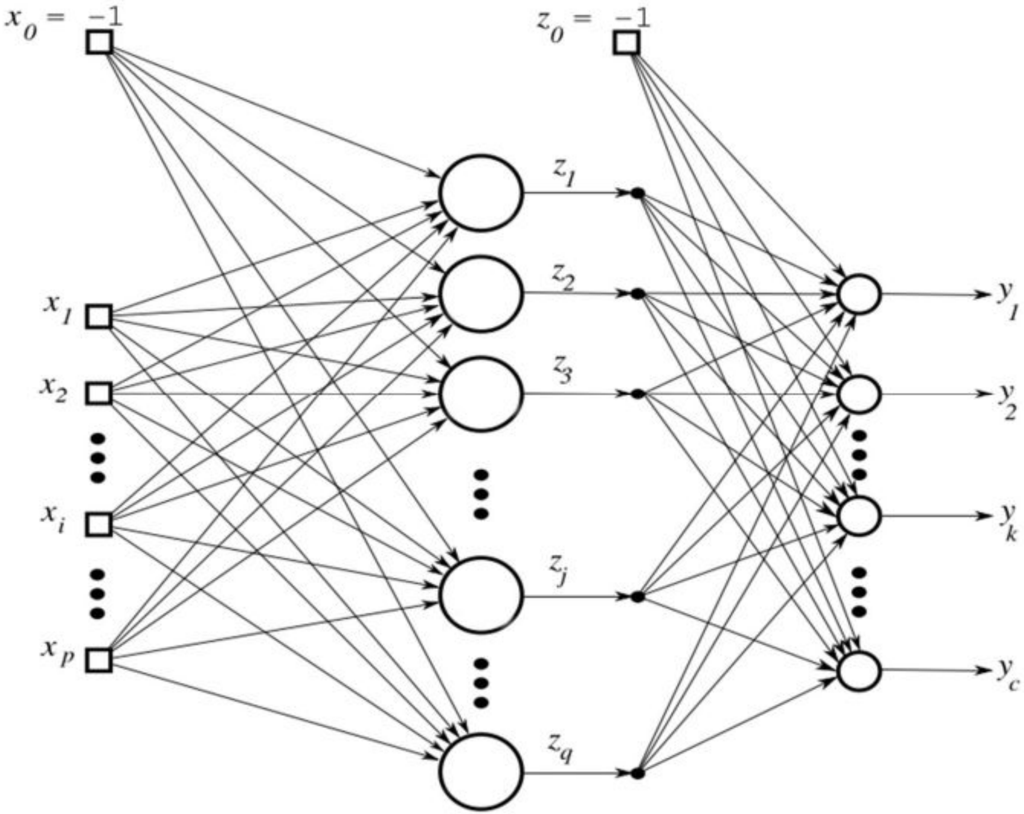
$$\text{MLP}(p, q_1, c)$$

Onde: p é o número de variáveis de entrada
q_1 é o número de neurônios ocultos
c é o número de neurônios de saída.

Logo, o número total de parâmetros (Z) de uma rede MLP de uma camada oculta é dado por

$$Z = (p+1)q_1 + (q_1+1)c$$

Rede Neural MLP



Rede Neural MLP

Perceptron Multicamadas (*Multilayer Perceptron*) - MLP

➢ Uma Rede MLP com 2 camada oculta é representada por:

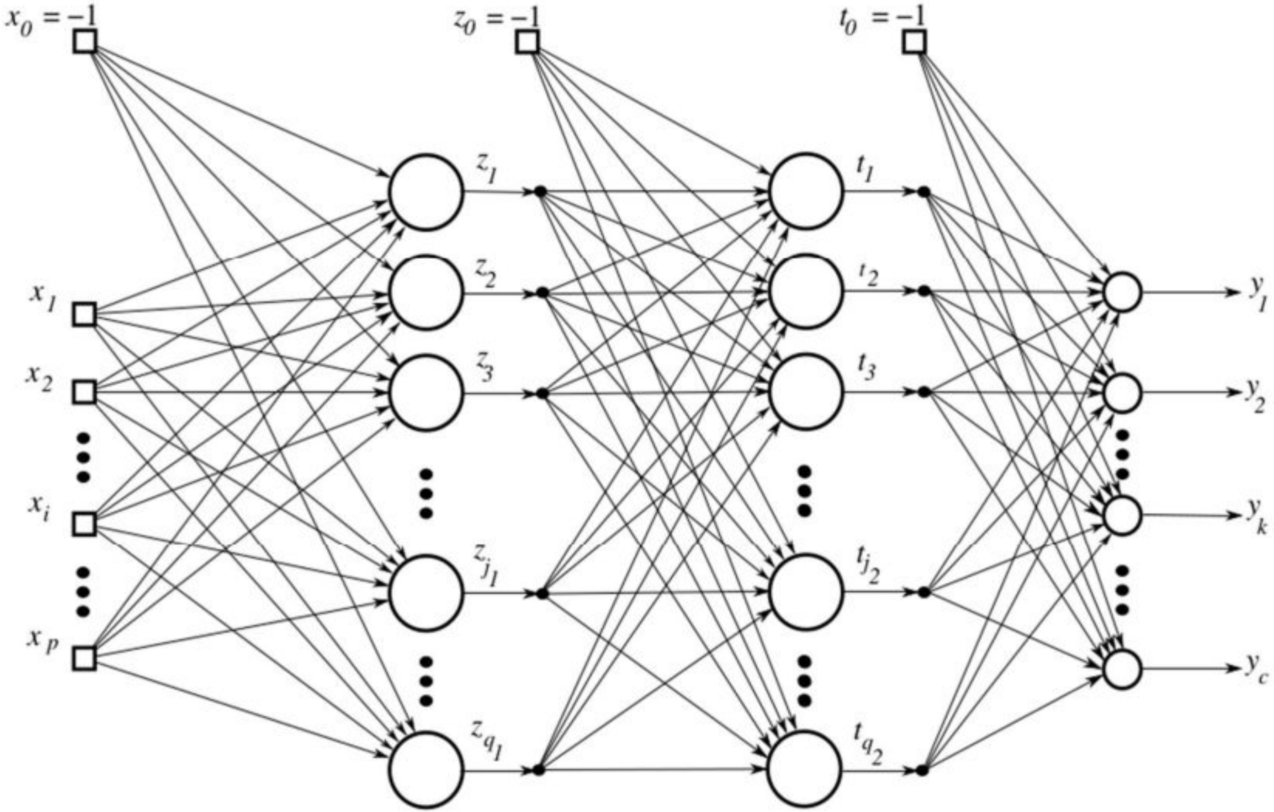
$$\text{MLP}(p, q_1, q_2, c)$$

Onde: p é o número de variáveis de entrada
q_1 é o número de neurônios da 1a. camada oculta
q_2 é o número de neurônios da 2a. camada oculta
c é o número de neurônios de saída.

Logo, o número total de parâmetros (Z) de uma rede MLP de duas camadas ocultas é dado por

$$Z = (p+1)q_1 + (q_1+1)q_2 + (q_2+1)c$$

Rede Neural MLP



Rede Neural MLP

Perceptron Multicamadas (*Multilayer Perceptron*) - MLP

- Uma rede MLP com 4 variáveis de entrada (p=4), 10 neurônios ocultos (q_1=10) e 2 neurônios de saída (c=2) é representada como MLP(4,10,2).

- Uma rede MLP com 15 variáveis de entrada (p=15), 20 neurônios na 1ª camada oculta (q_1=20), 10 neurônios na 2ª camada oculta (q_2=10) e 4 neurônios de saída (c=4) é representada como MLP(15,20,10,4).

Rede Neural MLP

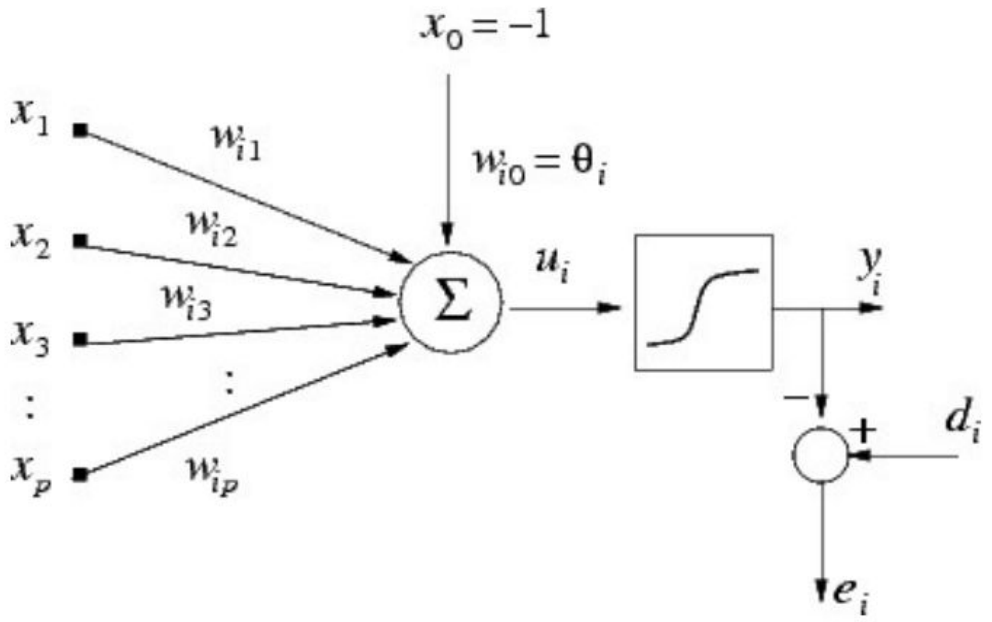
Perceptron Multicamadas (*Multilayer Perceptron*) - MLP

- **DETALHE 1:** A especificação de p e c são ditadas pela forma como o problema é codificado para ser resolvido por uma rede neural.

- **DETALHE 2:** As especificações de q_1 e q_2 dependem da complexidade do problema, ou seja, é preciso realizar vários testes até encontrar os valores mais adequados, ou calcular através de algoritmos de busca aleatória.

Rede Neural MLP

➢ Um neurônio qualquer da rede MLP (oculto ou de saída) é representado genericamente como se segue:



Rede Neural MLP

Perceptron Multicamadas (*Multilayer Perceptron*) - MLP

- **DETALHE 1:** Note que a função de ativação do neurônio M-P, que é do tipo **Degrau** (não-linearidade dura ou *hard*) foi substituída por uma função de ativação do tipo **Sigmoidal** (não-linearidade suave ou *soft*).

- **DETALHE 2:** Assim, a saída deixa de ser uma variável do tipo ON-OFF (binária [0,1] ou bipolar [-1,+1]) e passa a ser uma variável do tipo Real ou Analógica (qualquer valor entre [0,1] ou [-1,+1]).

Funções de Ativação

Devem seguir os seguintes requisitos:

- ➢ Não Linear
- ➢ Diferenciável

Principais tipos:

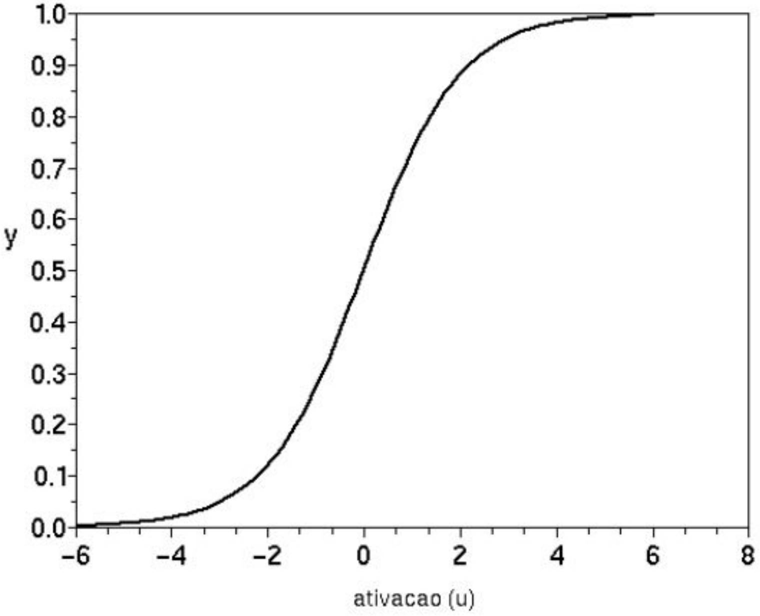
- ➢ Sigmóide Logística
- ➢ Tangente Hiperbólica

Funções de Ativação

Função de ativação ***Sigmóide Logística***

$$y_i(t) = \frac{1}{1+\exp\{-u_i(t)\}}$$

$$y_i(t) \in (0,1)$$

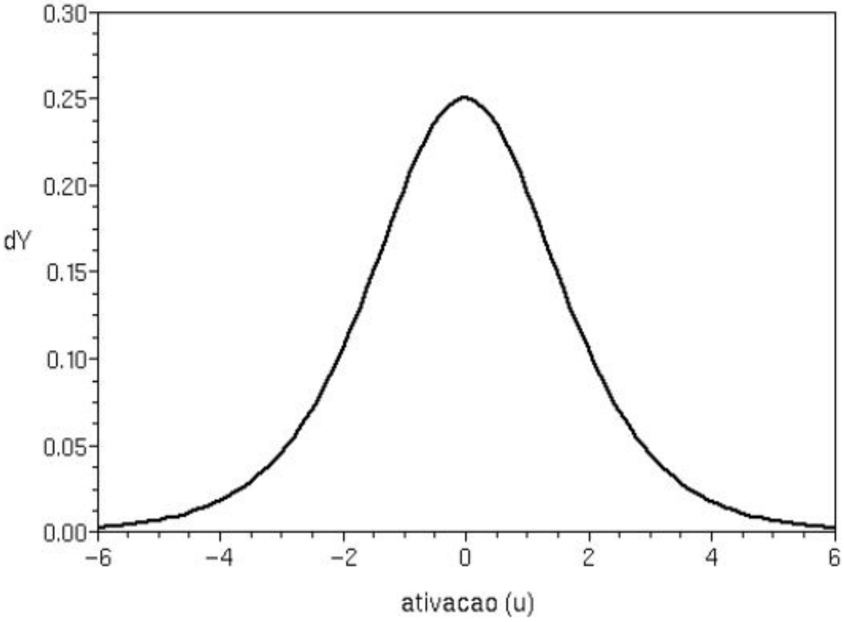


Funções de Ativação

Derivada da *Sigmóide Logística*

$$y_i'(t) = \frac{dy_i(t)}{du_i(t)}$$

$$y_i'(t) = y_i(t)(1 - y_i(t))$$

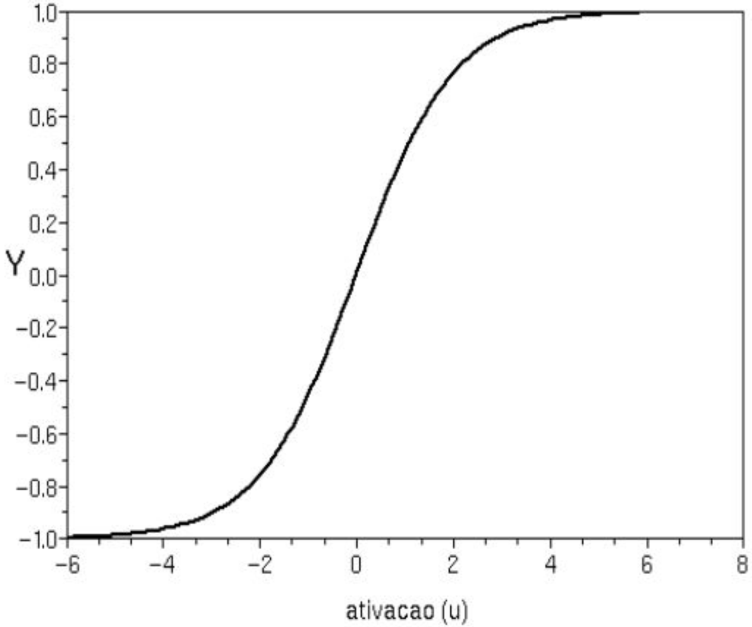


Funções de Ativação

Função de ativação ***Tangente Hiperbólica***

$$y_i(t) = \frac{1 - \exp(-u_i(t))}{1 + \exp(-u_i(t))}$$

$$y_i(t) \in (-1,1)$$

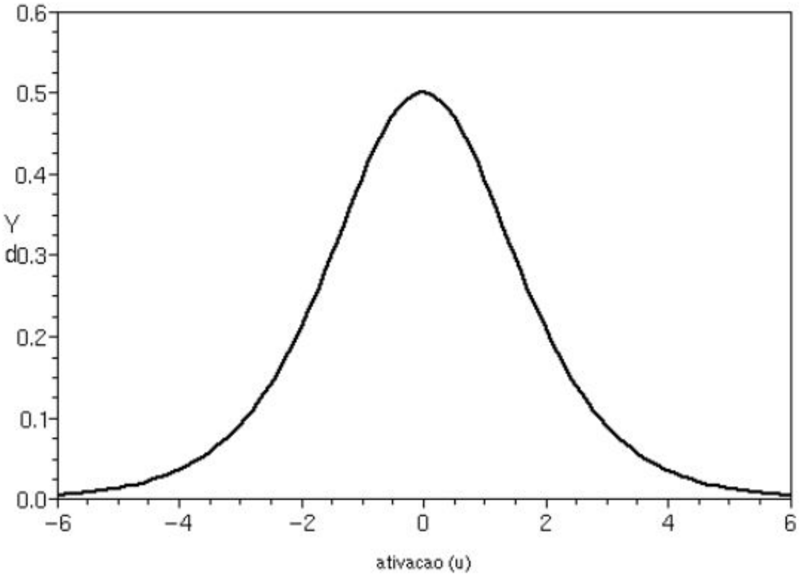


Funções de Ativação

Derivada da *Tangente Hiperbólica*

$$y_i'(t) = \frac{dy_i(t)}{du_i(t)}$$

$$y_i'(t) = 0{,}5(1 - y_i^2(t))$$



Funções de Ativação

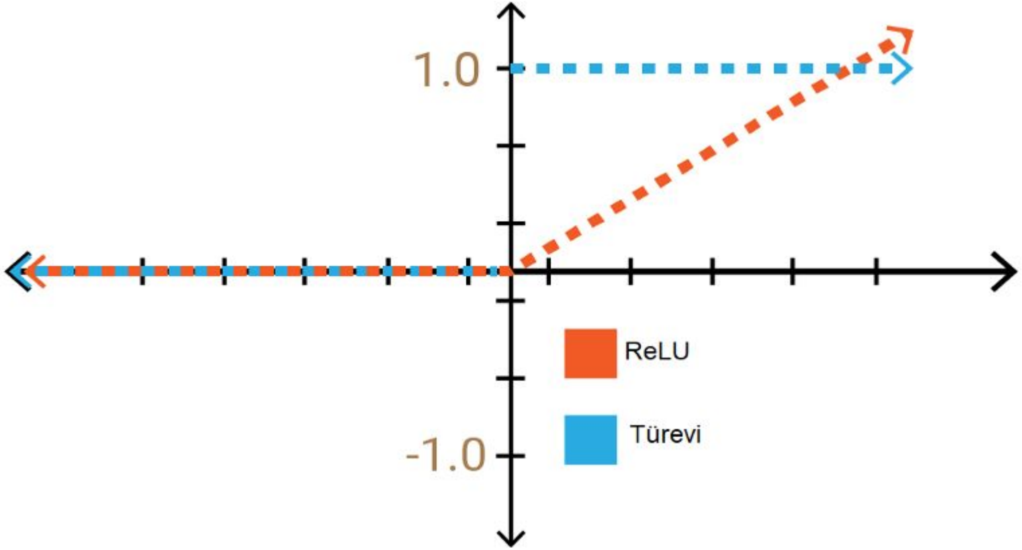
Funções de Ativação Sigmoidais:

- **Vantagens:**
  - Derivadas fáceis de calcular
  - Não-linearidade fraca (trecho central é quase linear)
  - Interpretação da saída como taxa média de disparo, em vez de simplesmente indicar se o neurônio está ou não ativado (ON-OFF).
- **Desvantagens:**
  - Elevado custo computacional para implementação (principalmente em sistemas embarcados) devido à presença da função exponencial

Funções de Ativação

Outras funções de ativação:

- ➢ **RELU:**

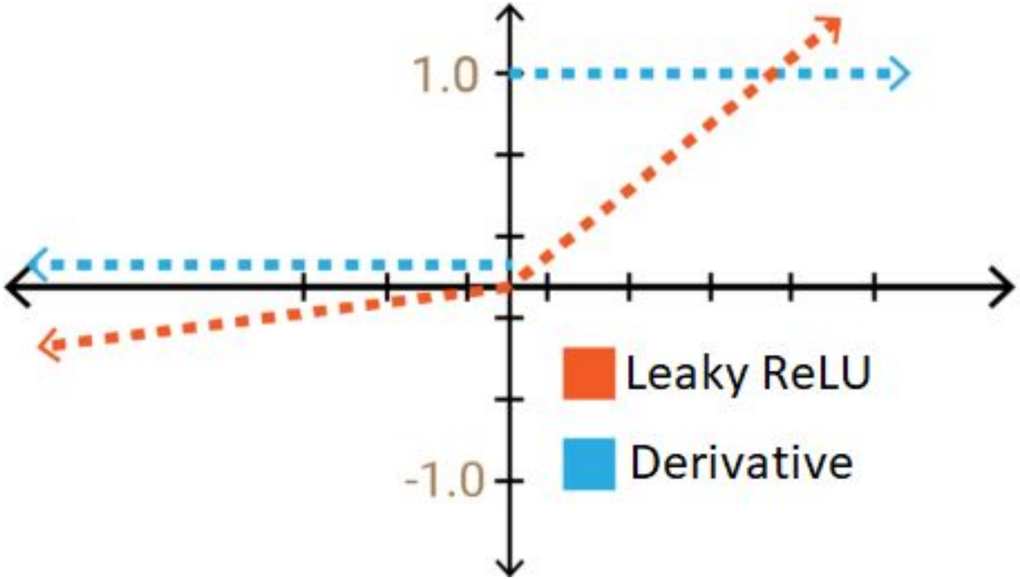


$$ReLU(x) = max\{0, x\} \qquad ReLU'(x) = \begin{cases} 1, & \text{se } x \geq 0 \\ 0, & \text{c.c.} \end{cases}$$

Funções de Ativação

Outras funções de ativação:

➢ **Leaky RELU:**

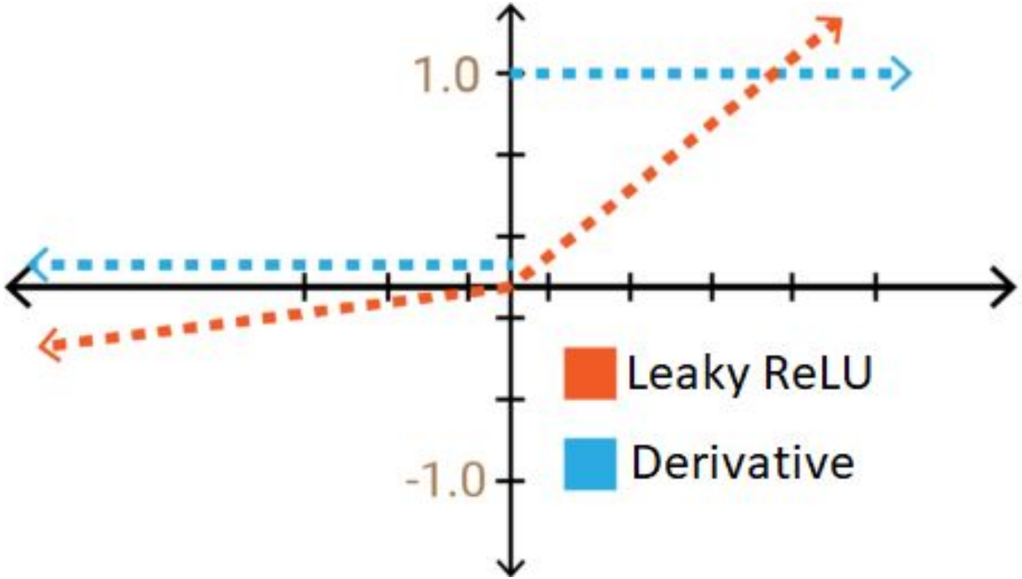


$$LeakyReLU(x,\alpha) = max\{\alpha x, x\} \qquad LeakyReLU'(x,\alpha) = \begin{cases} 1, & \text{se } x \geq 0 \\ \alpha, & \text{c.c.} \end{cases}$$

Funções de Ativação

Outras funções de ativação:

- **Leaky RELU:**



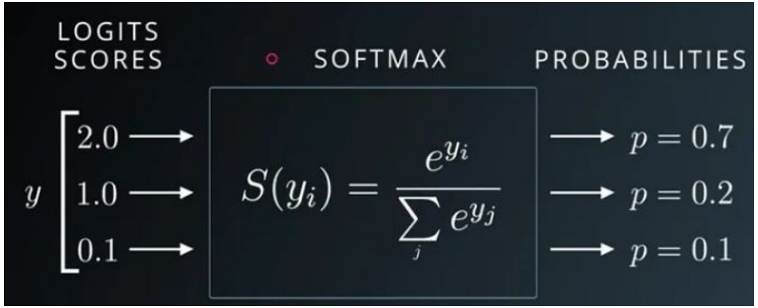
$$LeakyReLU(x,\alpha) = max\{\alpha x, x\} \qquad LeakyReLU'(x,\alpha) = \begin{cases} 1, & \text{se } x \geq 0 \\ \alpha, & \text{c.c.} \end{cases}$$

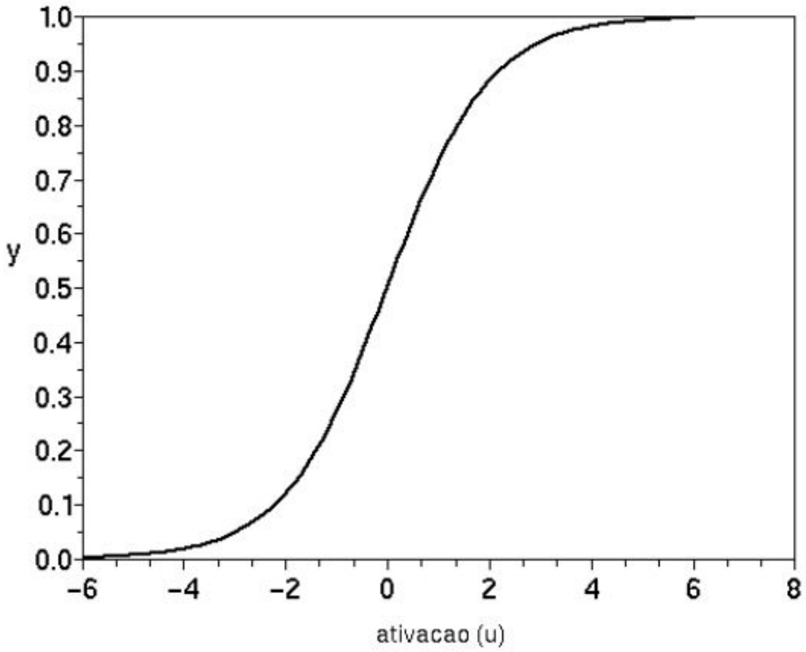
Funções de Ativação

Bônus - Classificação Multiclasse:

➢ ***Softmax***:

$$\text{softmax}(z_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1,\dots,K$$





Feedforward e *Backpropagation*

Funcionamento de uma rede MLP com 1 camada oculta:

(1) A ativação do i-ésimo neurônio da camada oculta é dada por

$$u_i = \mathbf{w}_i^T \mathbf{x} = w_{i0}x_0 + w_{i1}x_1 + \ldots + w_{ip}x_p \qquad i=1, \ldots, q_1$$

(2) A saída do i-ésimo neurônio da camada oculta é dada por

$$z_i(t) = \frac{1}{1+\exp(-u_i(t))}, \quad i=1, \ldots, q_1$$

Feedforward e *Backpropagation*

Funcionamento de uma rede MLP com 1 camada oculta:

(1) A ativação do k-ésimo neurônio de saída é dada por

$$a_k = \mathbf{m}_k^T \mathbf{z} = m_{k0} z_0 + m_{k1} z_1 + m_{k2} z_2 + \ldots + m_{kq} z_q \qquad k=1, \ldots, c$$

(2) A saída do k-ésimo neurônio de saída é dada por

$$y_k(t) = \frac{1}{1+\exp(-a_k(t))}, \quad k=1, \ldots, c$$

Feedforward e *Backpropagation*

Funcionamento de uma rede MLP com 1 camada oculta:

O k-ésimo neurônio de saída têm acesso à saída desejada, d_k.

Assim, é possível calcular o erro associado a esse neurônio:

$$e_k = d_k - y_k$$

Este erro pode então ser utilizado em uma regra de aprendizagem similar àquela usada pelo algoritmo Perceptron Simples.

$$\begin{aligned} \mathbf{m}_k(t+1) &= \mathbf{m}_k(t) + \eta e_k(t) y'_k(t) \mathbf{z}(t) \qquad k=1, \ldots, c \\ &= \mathbf{m}_k(t) + \eta \delta_k(t) \mathbf{z}(t) \end{aligned}$$

Onde: $\mathbf{z}(t)$ é o vetor de entrada da camada de saída.

$y'_k(t) = y_k(t)\left[1 - y_k(t)\right]$ (p/ sigmóide logística)

$y'_k(t) = 0{,}5\left[1 - y_k^2(t)\right]$ (p/ tangente hiperbólica)

Feedforward e *Backpropagation*

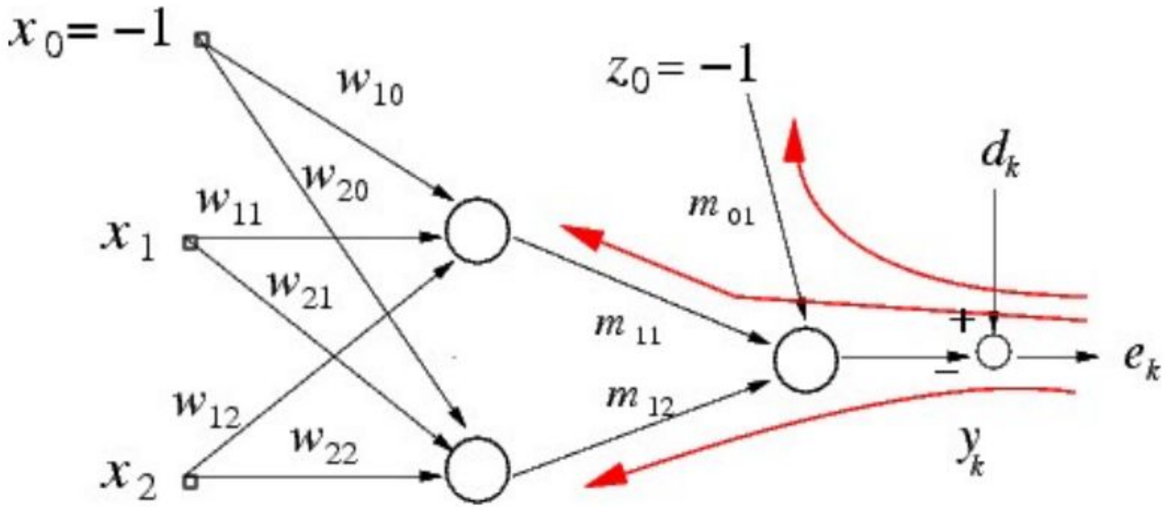
Funcionamento de uma rede MLP com 1 camada oculta:

- Contudo, o i-ésimo neurônio oculto não tem acesso a uma saída desejada equivalente, d_i. Assim, NÃO é possível calcular o erro associado a esse neurônio.
- A saída encontrada pelos pesquisadores foi “inventar” uma espécie de erro para os neurônios ocultos, sem que houvesse a necessidade de uma saída desejada, d_i.
- O erro dos neurônios ocultos são obtidos a partir dos erros dos neurônios de saída por meio de uma **projeção no sentido inverso** ao do fluxo de informação convencional.

Feedforward e *Backpropagation*

Funcionamento de uma rede MLP com 1 camada oculta:

Esta projeção no sentido inverso dos erros de saída é mais conhecida pelo nome de retropropagação dos erros (*Error Backpropagation*).



Feedforward e *Backpropagation*

Funcionamento de uma rede MLP com 1 camada oculta:

Neurônios Ocultos:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \eta e_i(t) z'_i(t)\mathbf{x}(t) \qquad i=1, \ldots, q_1$$
$$= \mathbf{w}_i(t) + \eta \delta_i(t)\mathbf{x}(t)$$

Onde: e_i é o erro retroprojetado do i-ésimo neurônio de saída

$$e_i(t) = \sum m_{ki}(t)\delta_k(t), \qquad i=1, \ldots, q_1$$

$\mathbf{x}(t)$ é o vetor de entrada da rede.

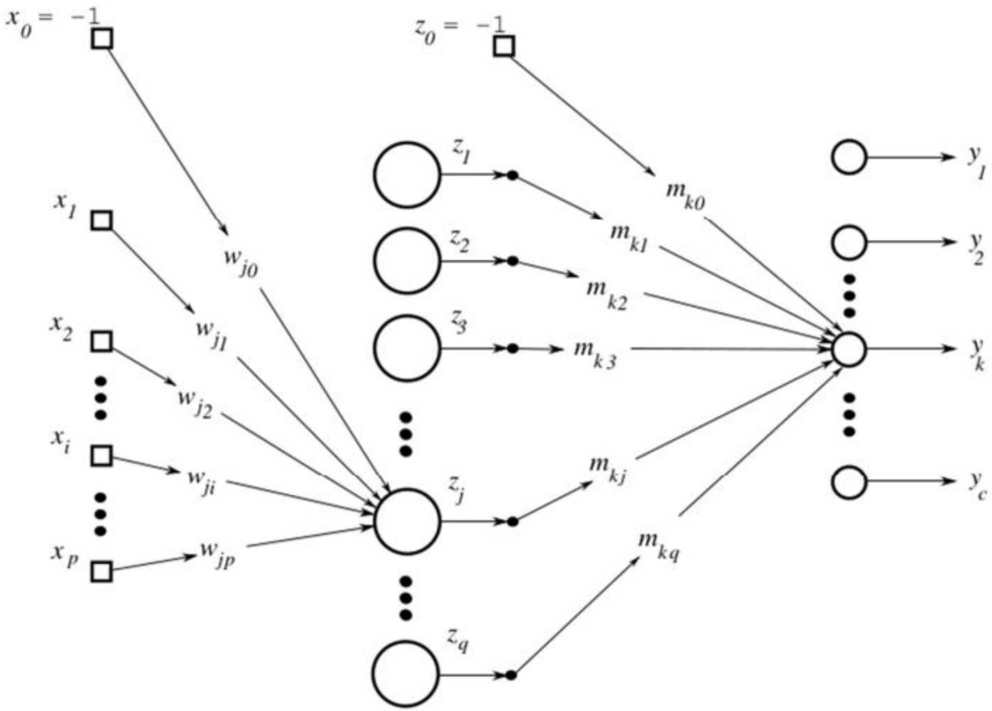
$$z'_i(t) = z_i(t)\left[1 - z_i(t)\right] \qquad \text{(p/ sigmóide logística)}$$

$$z'_i(t) = 0{,}5\left[1 - z_i^2(t)\right] \qquad \text{(p/ tangente hiperbólica)}$$

Feedforward e *Backpropagation*

Funcionamento de uma rede MLP com 1 camada oculta:

Feedforward:



Função Custo e Otimizadores

Para a rede PS, a regra de aprendizagem foi obtida através de uma análise geométrica do problema.

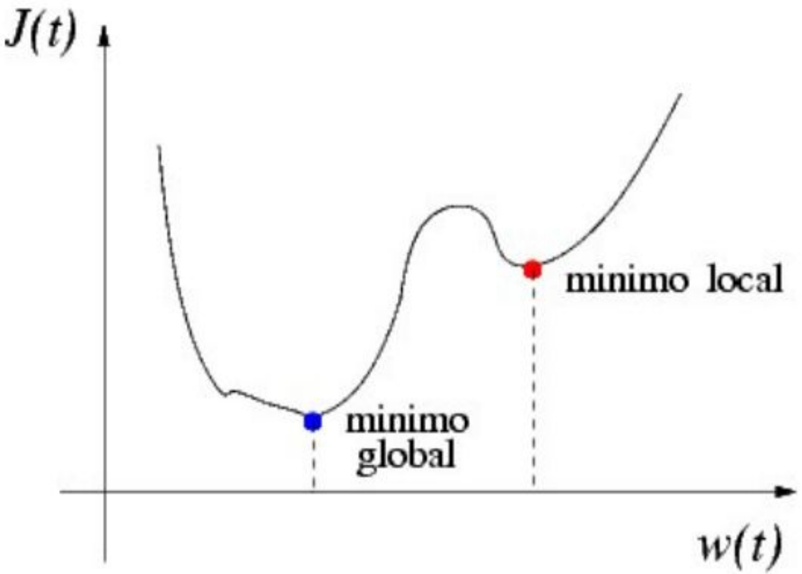
Para a rede MLP vamos obter uma regra de aprendizagem semelhante, a partir da minimização de uma função-custo (ou função objetivo).

Para isso, considere inicialmente que o *erro quadrático instantâneo* para todos os *m* neurônios de saída é dado por:

$$J(t) = \frac{1}{2}\sum e_k^2(t) = \frac{1}{2}\sum \left(d_k(t) - y_k(t)\right)^2$$

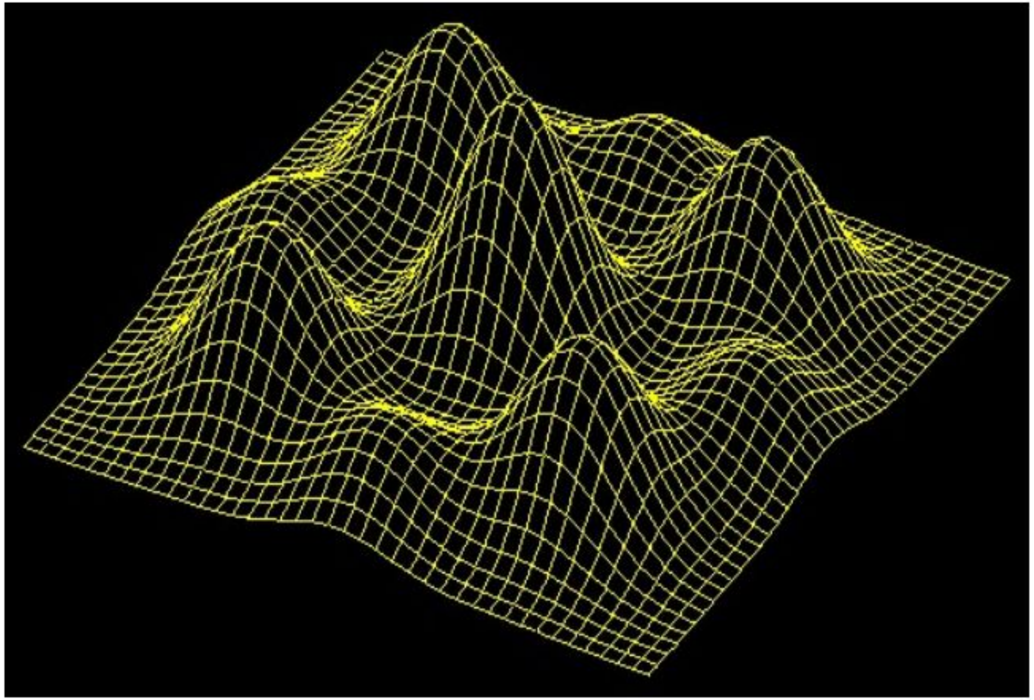
Função Custo e Otimizadores

Gráfico hipotético que ilustra o efeito da não-linearidade na função $J(t)$ para um único neurônio de saída com peso w é mostrado abaixo:



Função Custo e Otimizadores

Em três dimensões:



Função Custo e Otimizadores

A função-custo de interesse é o *Erro Quadrático Médio* (EQM), para os N exemplos de treinamento:

$$J(W) = \frac{1}{N} \sum J(t) = \frac{1}{2N} \sum \sum e_k^2(t)$$

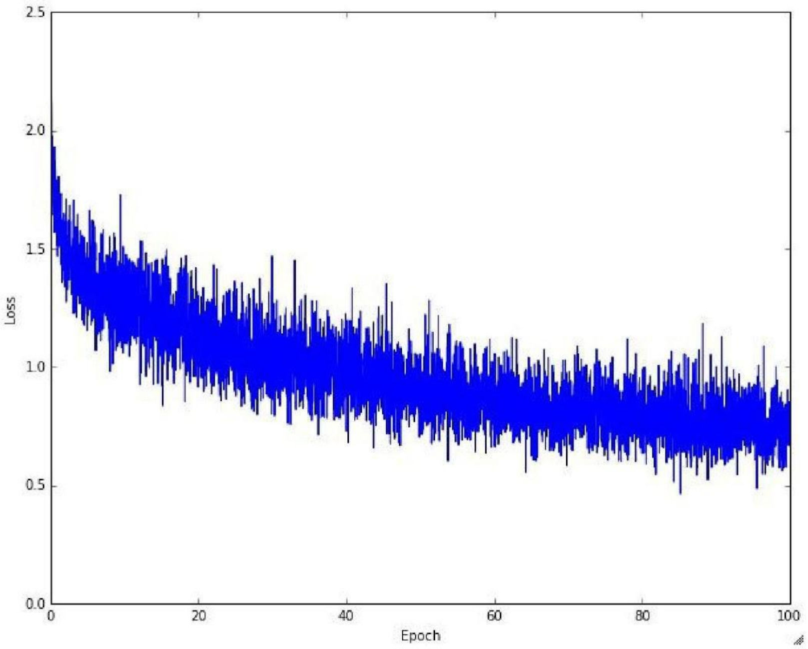
$$= \frac{1}{2N} \sum \sum \left(d_k(t) - y_k(t) \right)^2$$

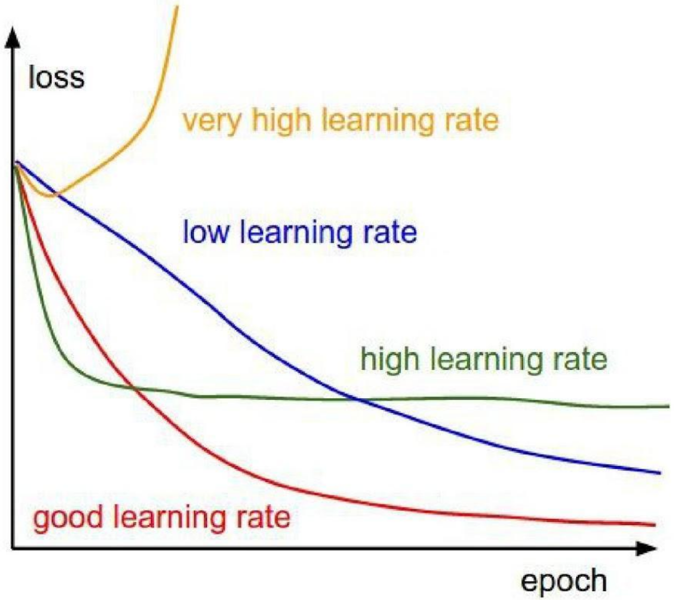
onde **W** é o conjunto de todos os parâmetros (pesos e limiares) da rede.

Note que a função $J(\mathbf{W})$ pode ser minimizada ao se minimizar $J(t)$!

Função Custo e Otimizadores

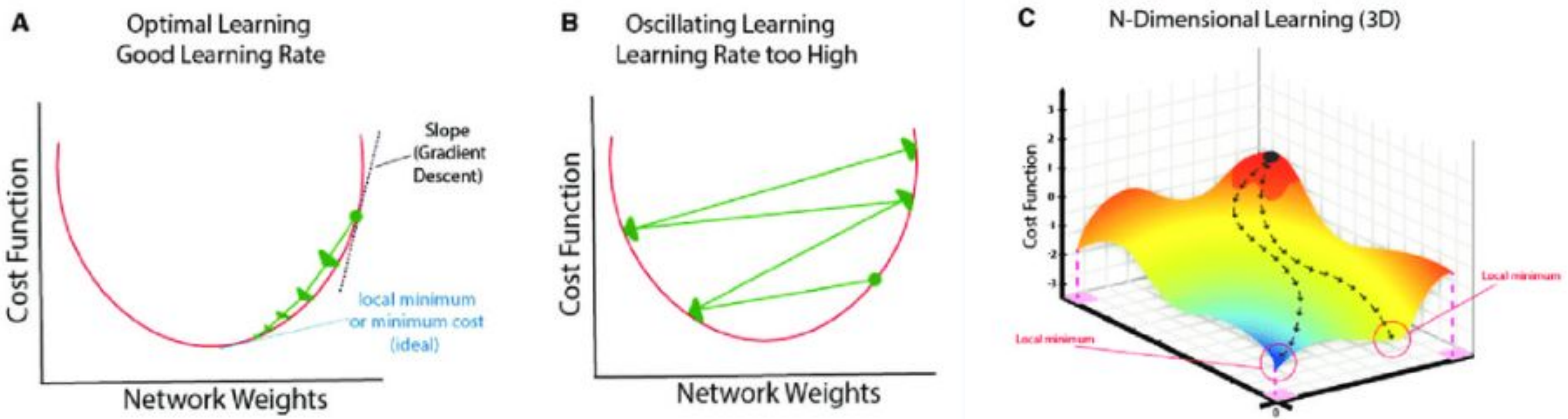
Mudança dos pesos iterativa. Muda mais rápido ou mais lento com base na taxa de aprendizagem



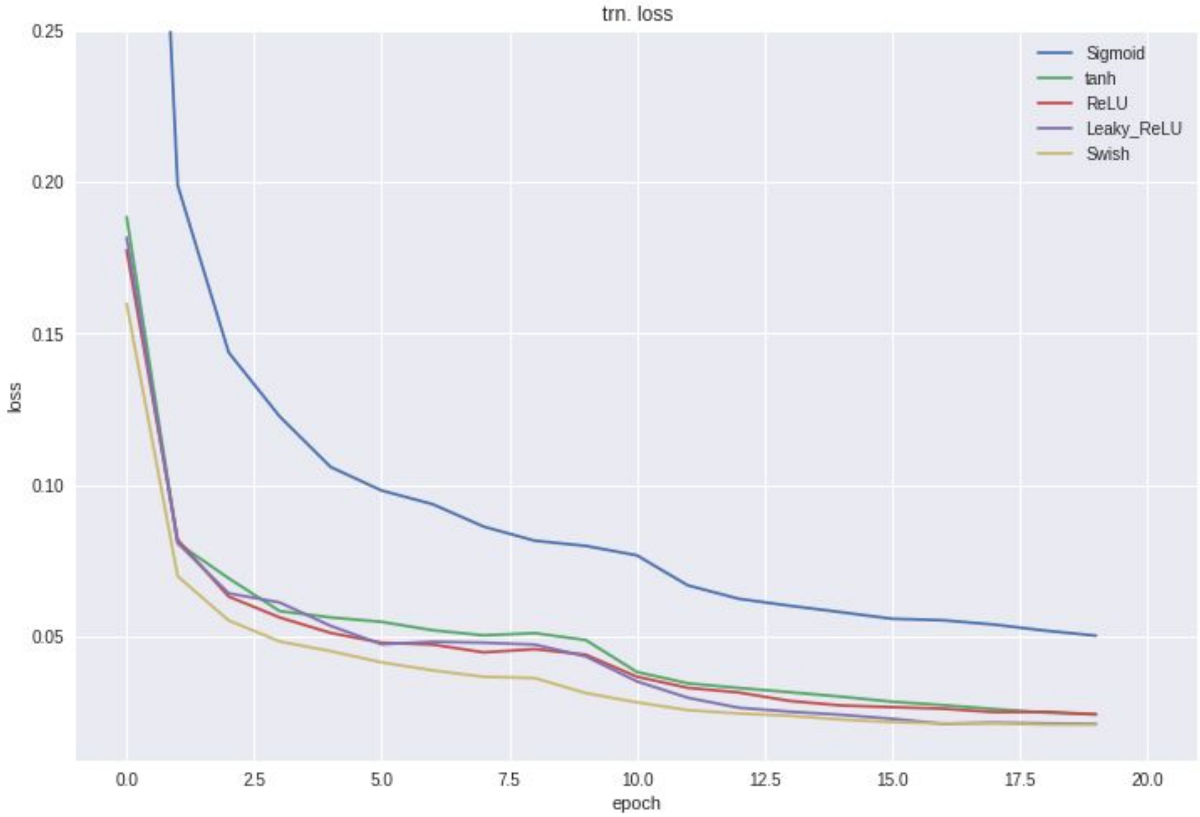


Função Custo e Otimizadores

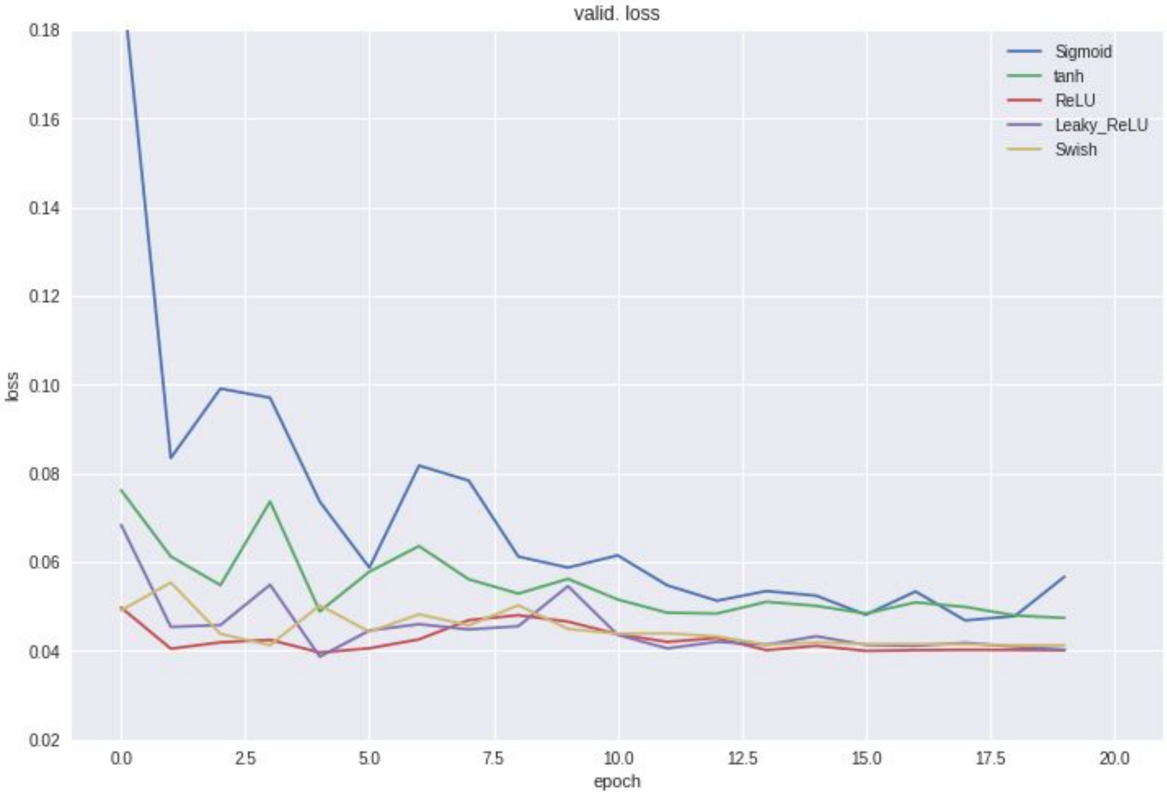
Mudança dos pesos iterativa. Muda mais rápido ou mais lento com base na taxa de aprendizagem



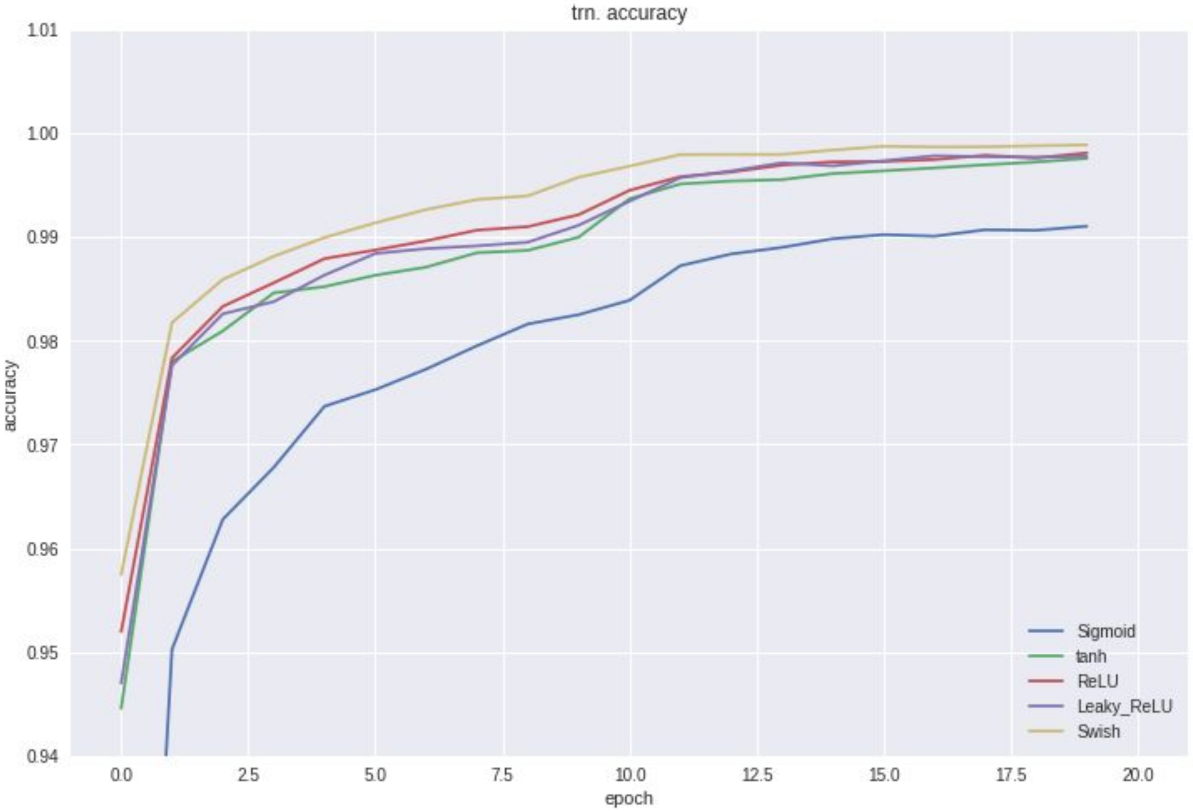
Função Custo e Otimizadores



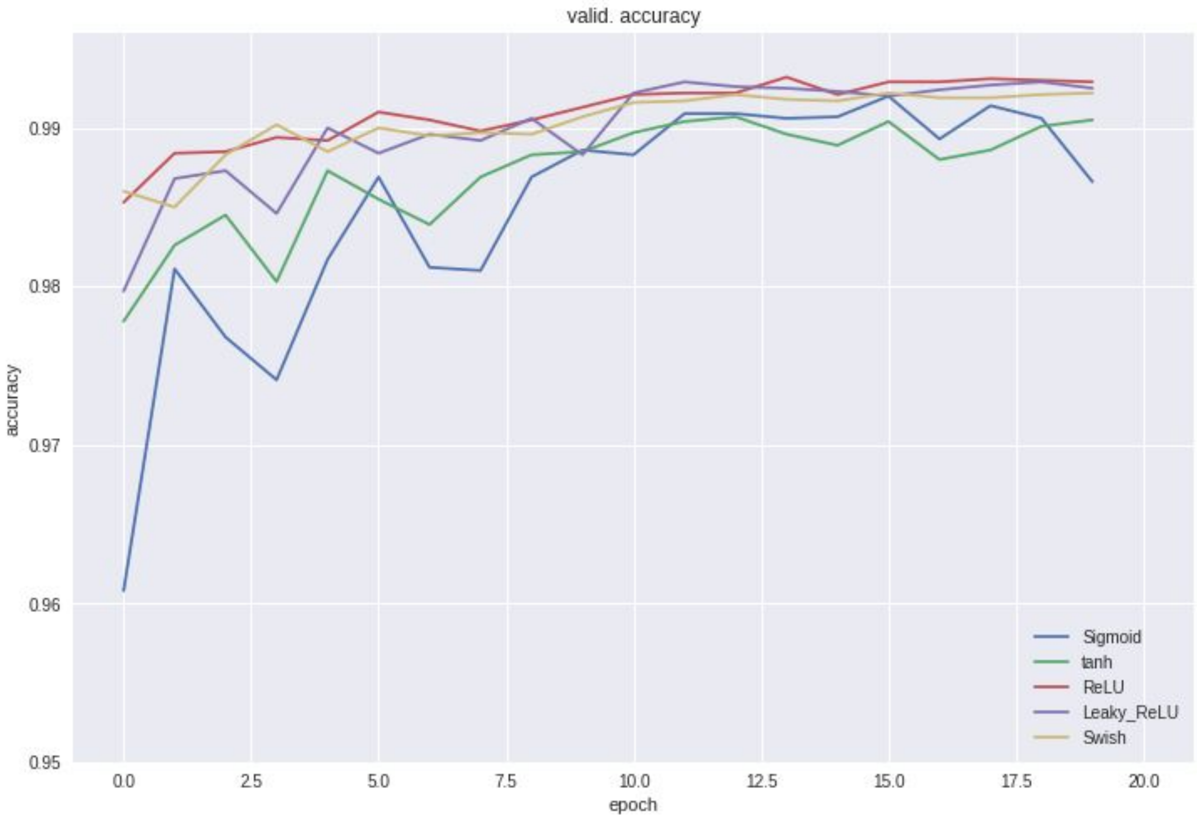
Função Custo e Otimizadores



Função Custo e Otimizadores



Função Custo e Otimizadores



Função Custo e Otimizadores

Comparativo:

- A convergência fácil e rápida da rede pode ser o primeiro critério
- Sigmóide apresenta a convergência mais lenta
- A TanH apresenta melhor ajuste de peso em relação a sigmóide
- A ReLU será vantajosa em termos de velocidade, apesar da perda dos gradientes. Geralmente é usado em camadas intermediárias, e não em uma saída
- A Leaky ReLU pode ser a primeira solução para o problema de os gradientes desaparecerem

Função Custo e Otimizadores

Otimizadores:

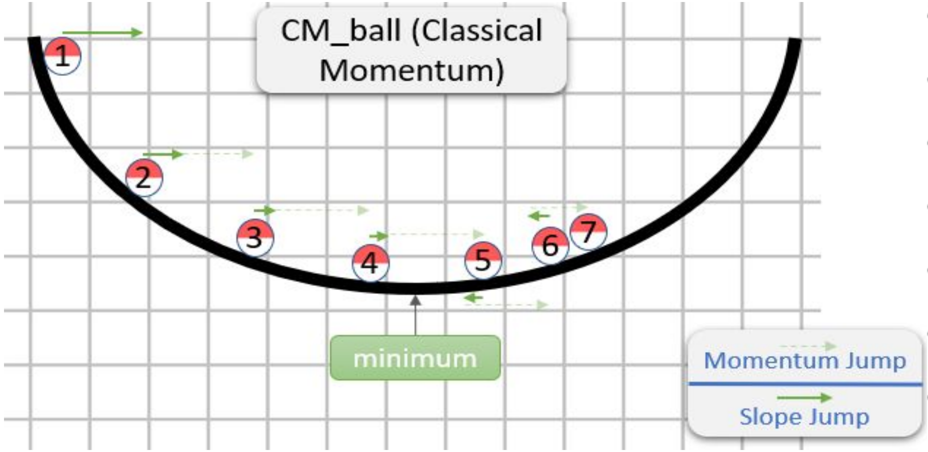
- **SGD+Momentum**
  a. Calcula o vetor gradiente
  b. Calcula a “velocidade” acumulada
  c. Acrescenta velocidade ao vetor de peso

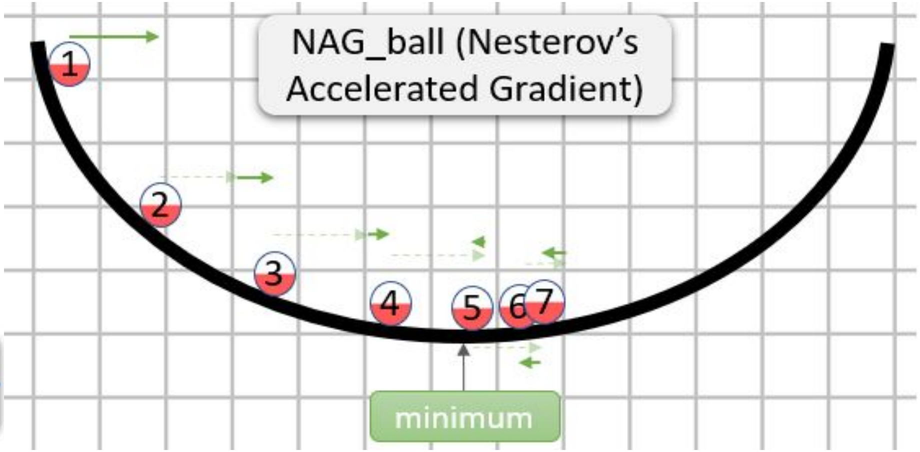
- **SGD+Nesterov (NAG)**
  a. Calcula o vetor gradiente
  b. Avança com a “velocidade” aplicando o momentum
  c. Ajusta o vetor com a componente de Nesterov e o calcula o gradiente.

- **Momentum x Nesterov (NAG - Nesterov Accelerated Gradient)**

Função Custo e Otimizadores

Otimizadores: Momentum x Nesterov (NAG - Nesterov Accelerated Gradient)





Função Custo e Otimizadores

Otimizadores: AdaGrad

- Até então os otimizadores atuavam com taxa de aprendizagem fixa.
- Cria-se um acumulador quadrático do valor do gradiente
- Atualiza a taxa de aprendizagem baseada na raíz do gradiente quadrático acumulado.

Função Custo e Otimizadores

Otimizadores: AdaDelta

- ➢ Atualiza a taxa de aprendizado dividindo-a pelo gradiente quadrático acumulado.
- ➢ O problema é que o gradiente irá acumular significativamente reduzindo muito a taxa de aprendizado. Sua influência tende a ser nula.
- ➢ O AdaDelta impõem um limitante a essa redução, evitando que a taxa de aprendizagem seja atenuada excessivamente.
- ➢ Não decai a taxa, ele adiciona um segundo momento do cálculo da velocidade (Momentum)

Função Custo e Otimizadores

Otimizadores: RMSProp

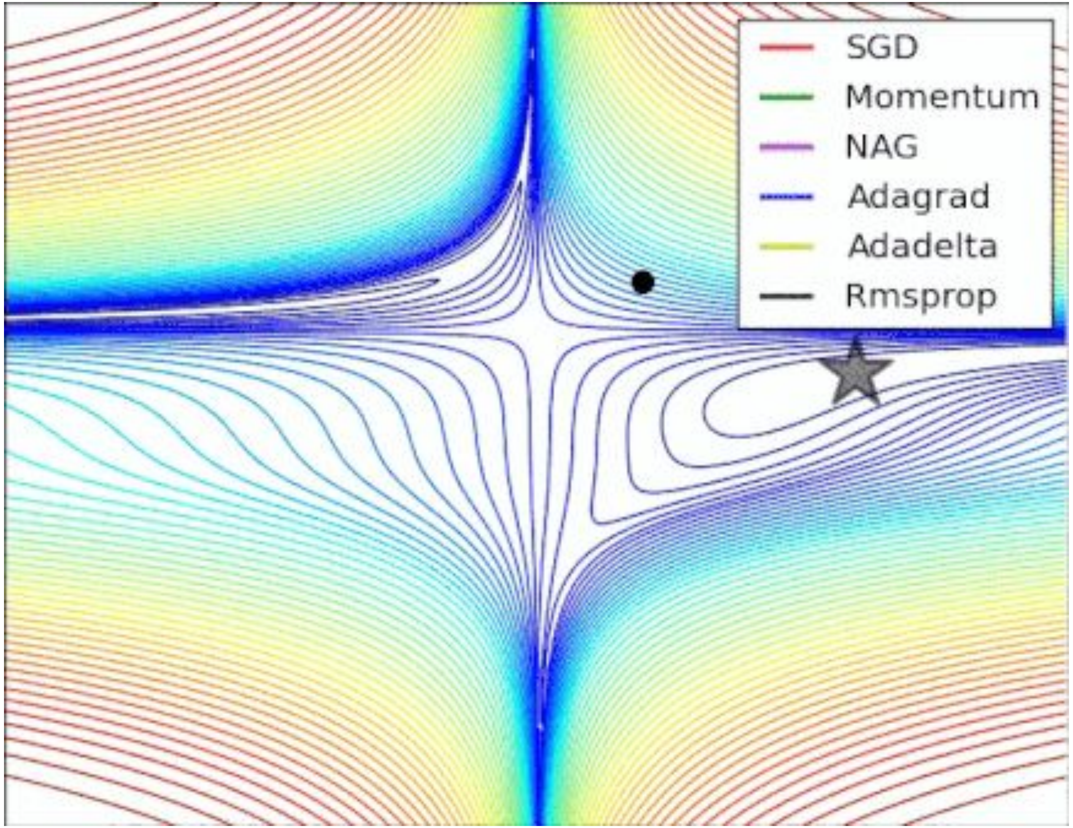
- ➢ Assim como no AdaGrad a taxa de aprendizado é reduzida, todavia, não mais em relação ao quadrado do gradiente acumulado.
- ➢ Adiciona um decaimento “rho” para ponderar o acúmulo do gradiente, evitando um alto crescimento e taxa de aprendizagem nula (Momento de segunda ordem)

Função Custo e Otimizadores

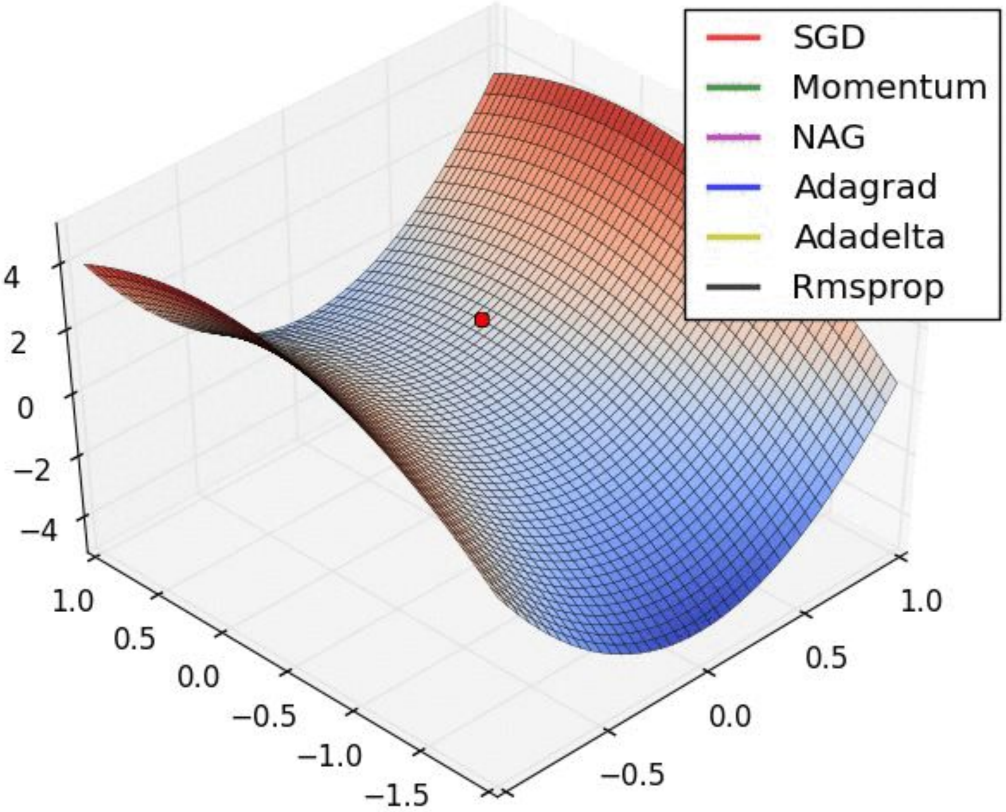
Otimizadores: Adam

- ➢ Combina o segundo momento do cálculo do gradiente do RMSProp com o segundo momento da velocidade do AdaDelta.
- ➢ Taxa de aprendizado decai sem atingir nulidade quando gradiente é muito elevado.
- ➢ Taxa de momentum (velocidade) adaptada ao erro. (“freia o vetor quando o erro reduz”).
- ➢ Adam é considerado um dos mais rápidos otimizadores.

Função Custo e Otimizadores



Função Custo e Otimizadores



Algoritmos de Busca Aleatória

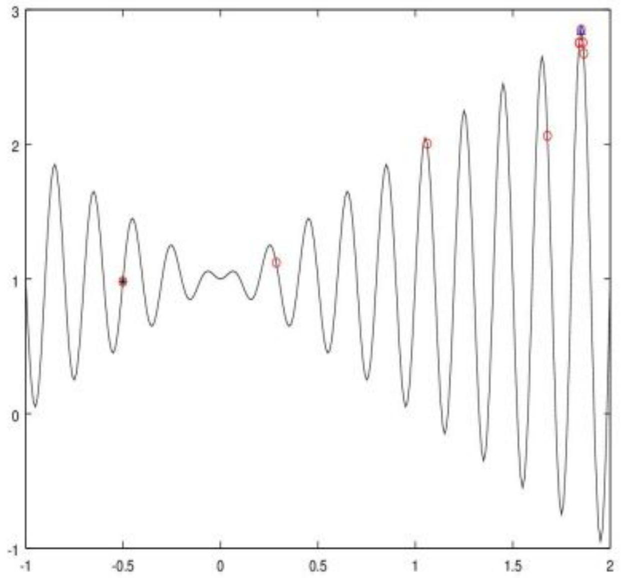
Quantos neurônios ocultos usar?

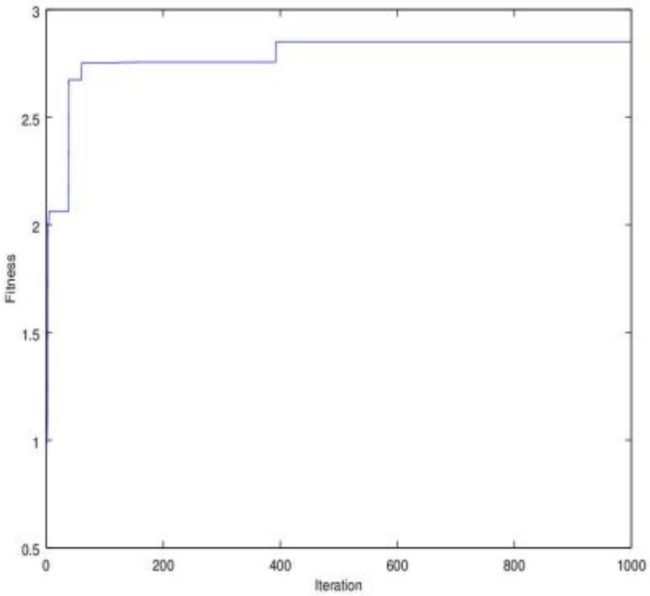
- Heurísticas Estocásticas
- Busca Aleatória Global (*Global Random Search*) - GRS
- Busca Aleatória Local (Local Random Search) - LRS
- Exemplo: Encontrar x que maximize a função $f(x) = x \cdot \sin(10\pi x) + 1$

Algoritmos de Busca Aleatória

Exemplo: Encontrar x que maximize a função $f(x) = x \cdot \sin(10\pi x) + 1$

GRS:

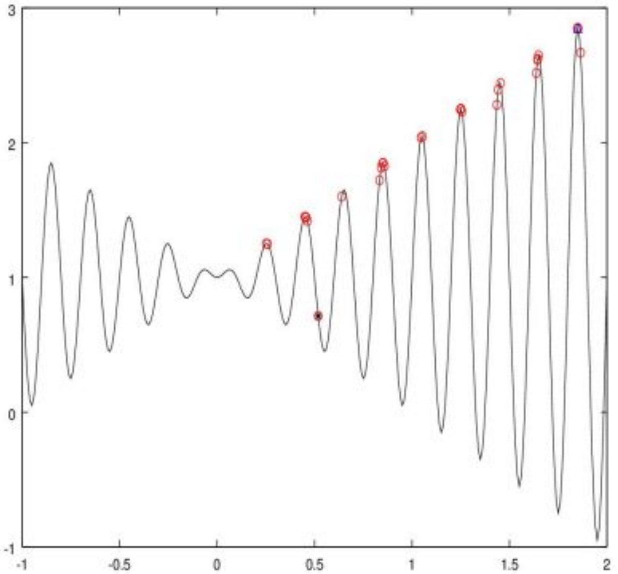


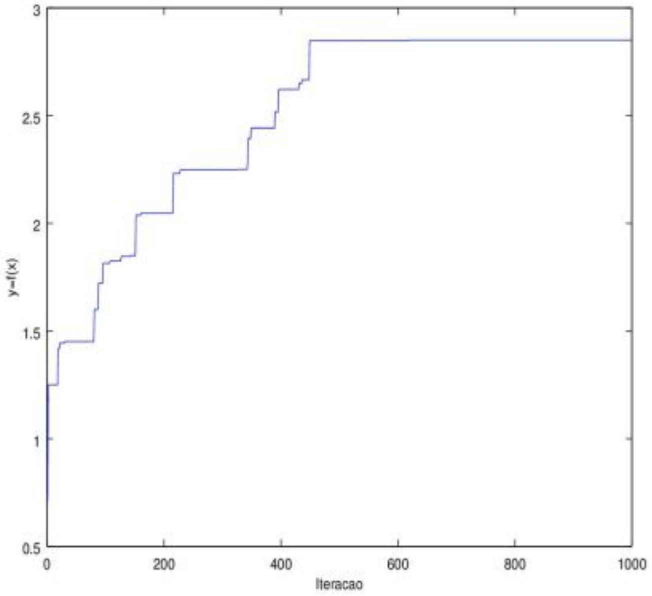


Algoritmos de Busca Aleatória

Exemplo: Encontrar x que maximize a função $f(x) = x \cdot \sin(10\pi x) + 1$

LRS:





Aplicações usando MLP

- Porta Lógica XOR
- Implementação para classificação de imagens (MNIST)
- Implementação para classificação com dados (Iris)
- Implementação com GRS
- Implementação com otimizador Adam
- Implementação como regressor

Bônus: *Extreme Machine Learning* (ELM)

- ➢ Máquina de Aprendizado Extremo
- ➢ Alternativa ao MLP
- ➢ Somente *feedforward* (sem *backpropagation*)
- ➢ Baseado no cálculo de mínimos quadrados (MQO)
- ➢ Treinamento infinitamente mais rápido
- ➢ Cálculo dos pesos com base na expressão:

$$\mathbf{M} = \mathbf{D}\mathbf{Z}^T \left(\mathbf{Z}\mathbf{Z}^T\right)^{-1}$$

Bônus: *Extreme Machine Learning* (ELM)

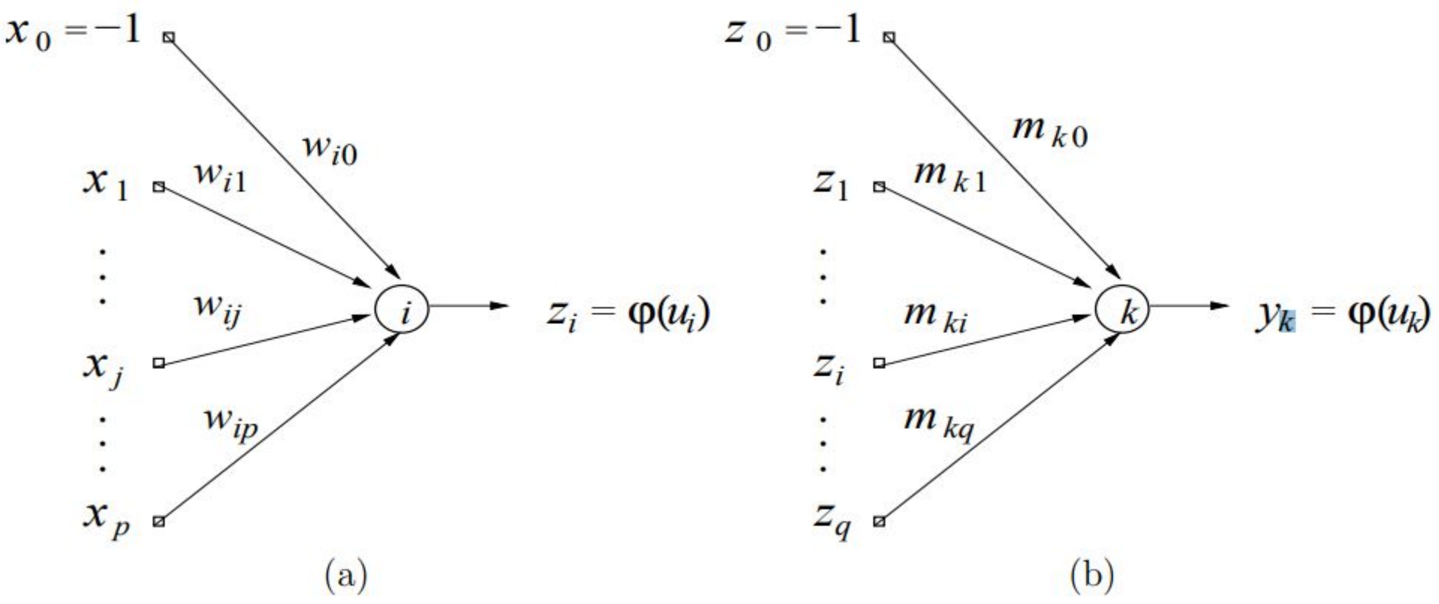


Figura 2: (a) Neurônio da camada escondida. (b) Neurônio da camada de saída.

Aplicações usando ELM

- Implementação como regressor
- Implementação para classificação de imagens (MNIST)
- Implementação para classificação com dados (Iris)
- PS x MLP x ELM

Próxima aula

- Comitês de classificadores (Ensembles)
- Técnicas de *bagging* e *bootstrap*
- Tira-dúvidas

Referências

1. A. Webb. Statistical Pattern Recognition. John Wiley & Sons, 2nd edition, 2002.
2. HAYKIN, Simon. Redes Neurais: Princípios e prática. Bookman, 2nd edition, 2001.
3. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.
4. M.-L. Zhang and Z.-H. Zhou, “A review on multi-label learning algorithms", IEEE transactions on knowledge and data engineering, vol. 26, no. 8, pp. 1819–1837, 2013.
5. Hastie, T., Tibshirani, R., Friedman, J., The elements of statistical learning: Data mining, inference, and prediction. Springer, 2009.
6. FISHER, Ronald A.; MARSHALL, Michael. Iris data set. RA Fisher, UC Irvine Machine Learning Repository, v. 440, p. 87, 1936.

Obrigado pela atenção!

Dúvidas?

Prof. Matheus Santos
matheus.santos@lapisco.ifce.edu.br