

Aprendizado de Máquina

Aula 10: Redes Neurais

André C. P. L. F de Carvalho
ICMC/USP

andre@icmc.usp.br

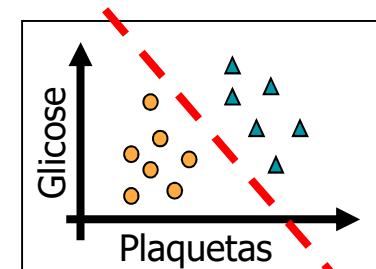


Tópicos

- Técnicas geométricas
- Discriminante linear
- Redes neurais artificiais
- Arquitetura e aprendizado de redes neurais
- Rede perceptron
- Rede adaline
- Rede multi-layer perceptron (MLP)

Discriminante linear

- Busca modelo que melhor se ajuste aos dados
 - Representação matemática
 - Dois atributos preditivos
 - Fronteira de decisão = reta (hiperplano para > 2)
 - Classificação
 - Função discriminante
 - Combinação linear dos atributos preditivos
 - Soma ponderada
 - Como definir valores dos pesos?



$$\begin{aligned}y &= ax + b \\f(x) &= ax + b \\f(x) &= -2x + 15\end{aligned}$$

Função de classificação:

$$classe(x) = \begin{cases} +1 & \text{se } f(x) + 2p - 15 \geq 0 \\ -1 & \text{se } f(x) + 2p - 15 < 0 \end{cases}$$

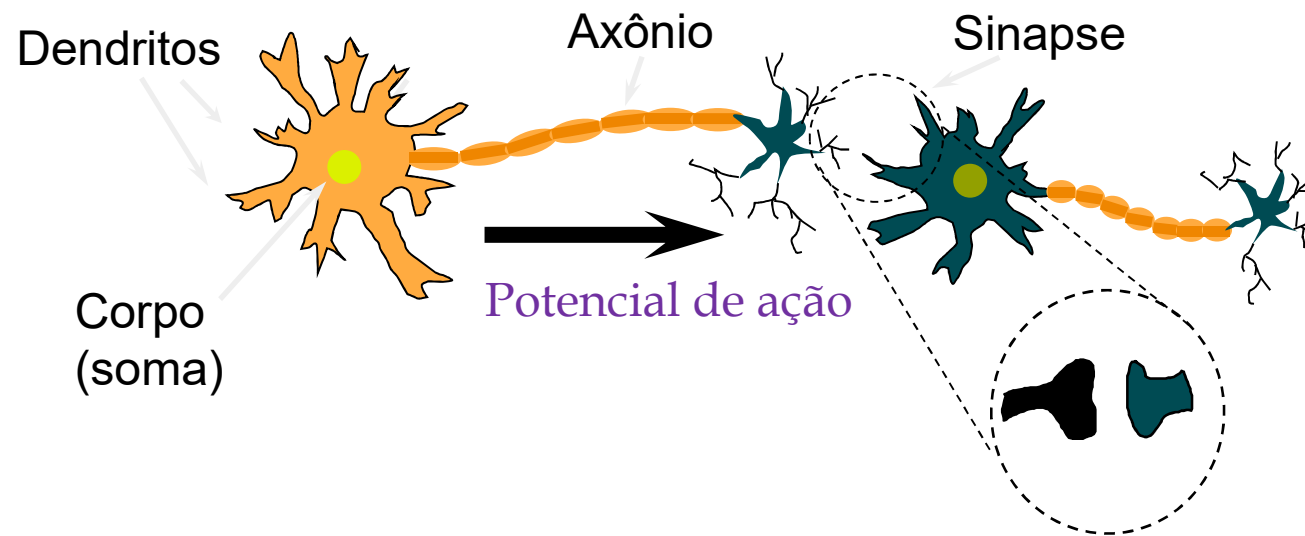
$$\begin{aligned}f(x) &= w_0 + w_1x_1 \\f(x) &= w_0 + w_1x_1 + w_2x_2 + \dots\end{aligned}$$

Redes Neurais

- Sistemas distribuídos inspirados no cérebro humano
 - Redes são compostas por várias unidades de processamento (“neurônios”)
 - Interligadas por um grande número de conexões (“sinapses”)
- Bom desempenho preditivo em várias aplicações

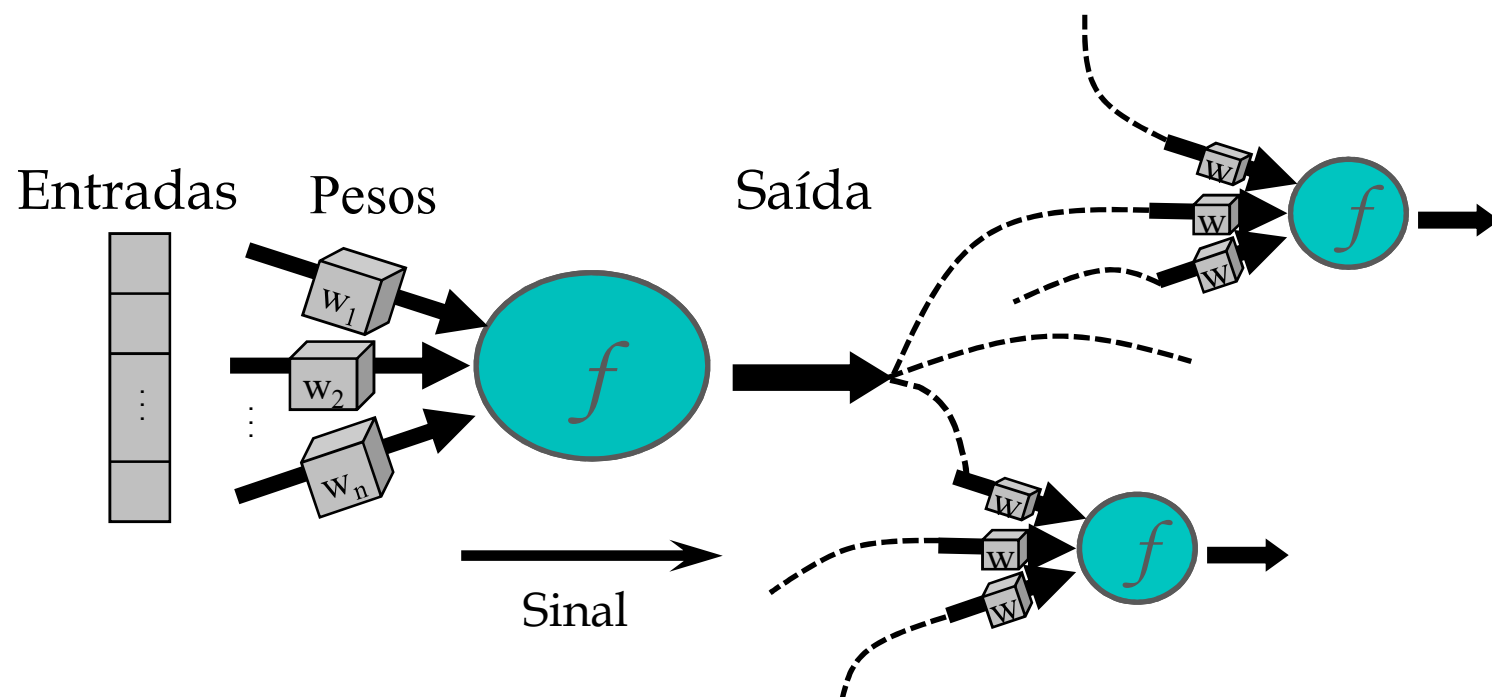
Redes Neurais

- Um neurônio simplificado:



Neurônio artificial

- Modelo de um neurônio abstrato



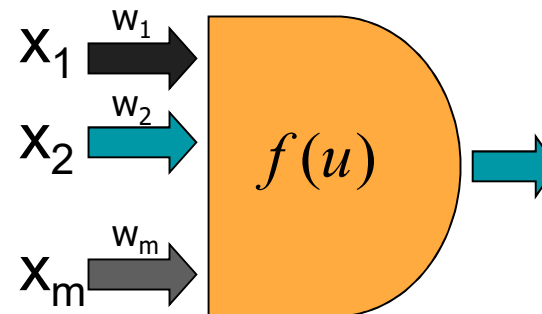
Conceitos básicos

- Principais aspectos das RNA
 - Arquitetura
 - Unidades de processamento (neurônios)
 - Conexões (sinapses)
 - Topologia
 - Aprendizado
 - Algoritmos
 - Paradigmas

Unidades de processamento

- Funcionamento
 - Recebem entradas de conjunto de unidades A
 - Aplicam função de ativação sobre entradas
 - Envia resultado para saída ou conjunto de unidades B
- Entrada total

$$u = \sum_{i=1}^m x_i w_i$$

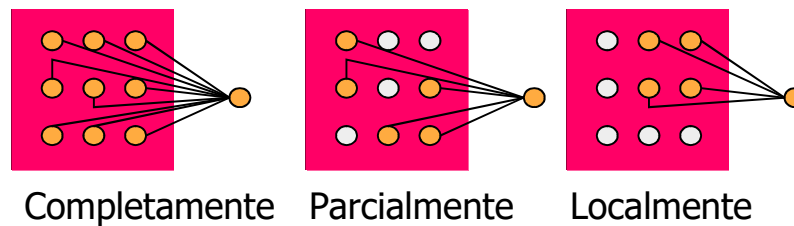


Conexões

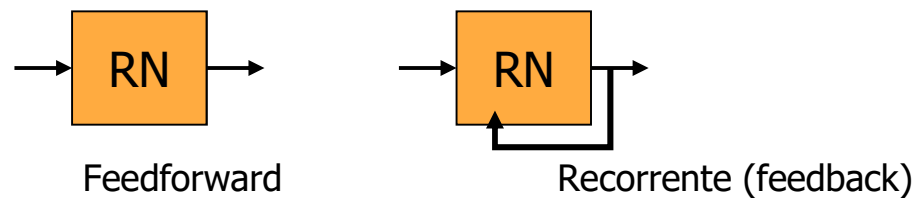
- Definem como neurônios artificiais estão interligados
- Codificam conhecimento da rede
- Tipos de conexões:
 - Excitatória: $(w_{ik}(t) > 0)$
 - Inibitória: $(w_{ik}(t) < 0)$
- Número de conexões de um neurônio
 - *Fan-in*
 - *Fan-out*

Topologia

- Número de camadas
- Cobertura das conexões



- Arranjo das conexões



Algoritmos de aprendizado

- Conjunto de regras que define como ajustar os parâmetros da rede
- Principais formas de ajuste
 - Correção de erro
 - Hebbiano
 - Competitivo
 - Termodinâmico (Boltzmann)
- Diferem na maneira como os pesos são ajustados

Paradigmas de aprendizado

- Definidos pelas informações externas que a rede recebe durante seu aprendizado
 - Principais abordagens
 - Supervisionado
 - Não supervisionado
 - Semi-supervisionado
 - Aprendizado ativo
 - Reforço
 - Híbrido

História das Redes Neurais

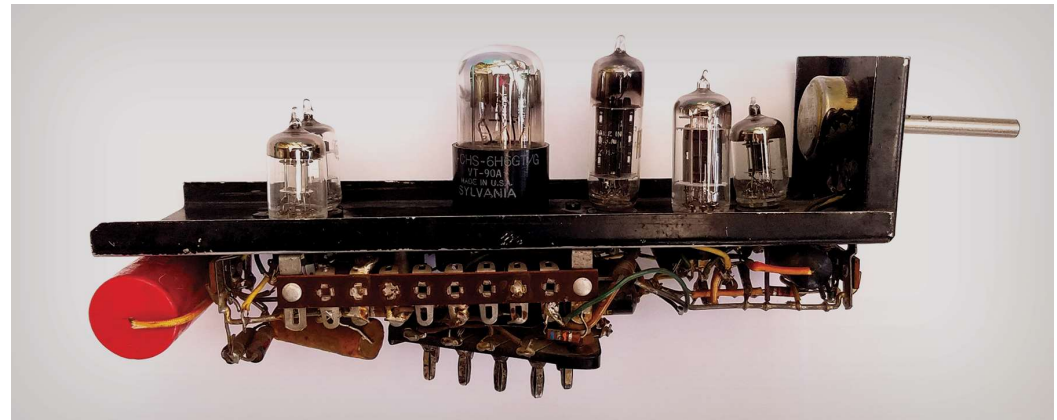
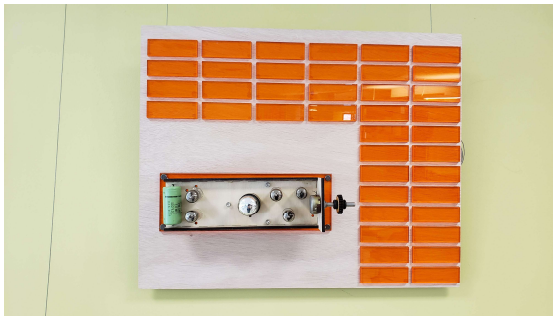
- 300 A. C.: Aristóteles escreveu: *de todos os animais, o homem, proporcionalmente, tem o maior cérebro*
- 1700 D.C.: Descartes acreditava que mente e cérebro eram entidades separadas
- 1911: Ramon y Cajal introduz a idéia de neurônios como estruturas básicas do cérebro
 - Considerado o pai da neurociência moderna
- 1943: McCulloch & Pitts desenvolvem modelo matemático de RNAs
- 1949: Hebb desenvolve algoritmo para treinar RNA (aprendizado Hebbiano)
 - Se dois neurônios estão simultaneamente ativos, a conexão entre eles deve ser reforçada

História das Redes Neurais

- 1951: Marvin Minsky and Dean Edmonds constoem a primeira máquina que implementa uma rede neural (Princeton/Harvard)
 - SNARC (Stochastic neural analog reinforcement calculator)
- 1957: Rosenblatt implementa primeira RNA a ser usada na prática, a rede Perceptron
- 1958: Von Neumann mostra interesse na modelagem do cérebro
 - The Computer and the Brain, Yale University Press
- 1969: Minsky & Papert publicam livro mostrando limitações da rede Perceptron
- 1982: Hopfield mostra que Redes Neurais podem ser tratadas como sistemas dinâmicos

SNARC

- Stochastic **N**eural **A**nalog **R**einforcement **C**alculator
 - Implementada por Marvin Minsky e Dean Edmonds em 1951
 - Primeira implementação de uma rede neural
 - Testada para sair de um labirinto

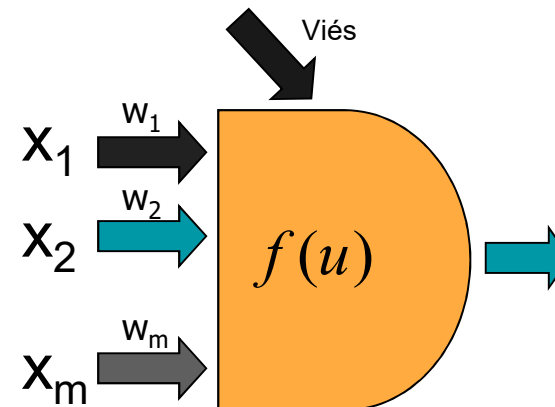


SNARC

- Rede Neural baseada em componentes analógicos e eletromecânicos
- Possuía 40 neurônios conectados em rede
 - Cada neurônio usava:
 - Um capacitor: para memória de curto prazo
 - Componente que armazena energia elétrica
 - Um potenciômetro, para memória de longo prazo
 - Funcionavam como pesos associados às conexões de entrada dos neurônios artificiais
 - Treinada por algoritmo de aprendizado por reforço

Rede Perceptron

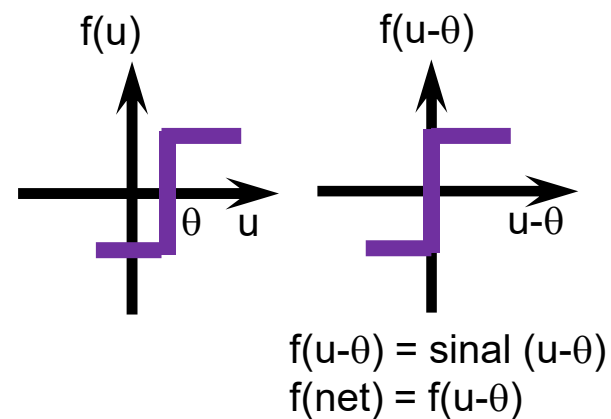
- Proposta por Rosemblat, 1957
 - Modelo de neurônio de McCulloch-Pitts
- Treinamento
 - Supervisionado
 - Correção de erro
 - $w_i(t) = w_i(t-1) + \Delta w_i$
 - $\Delta w_i = \eta x_i \delta$
 - $\Delta w_i = \eta x_i (y - f(\mu))$
- Teorema de convergência



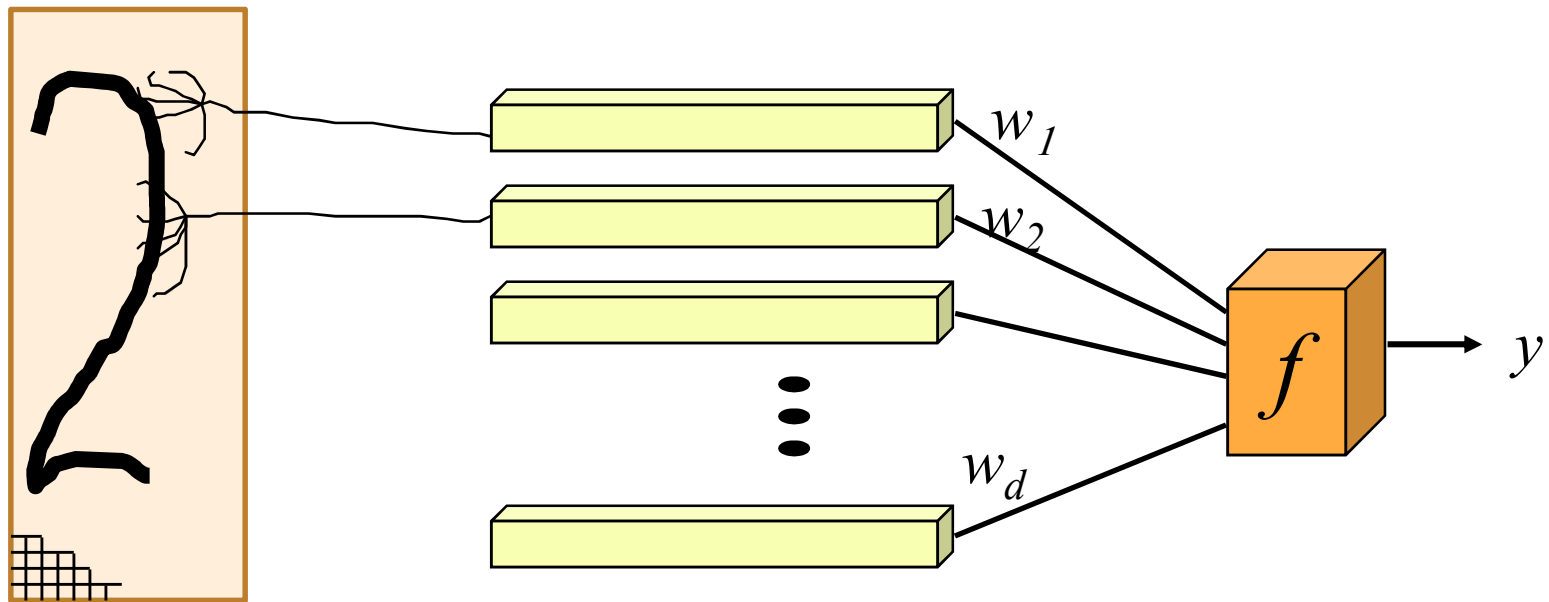
Rede Perceptron

- Resposta / saída da rede
 - Aplica função de ativação limiar sobre soma total de entrada recebida por um neurônio

$$u = \sum_{i=1}^m x_i w_i$$
$$f(u) = \begin{cases} +1 & \text{if } u \geq \theta \\ -1 & \text{if } u < \theta \end{cases}$$
$$net = \sum_{i=0}^m x_i w_i$$

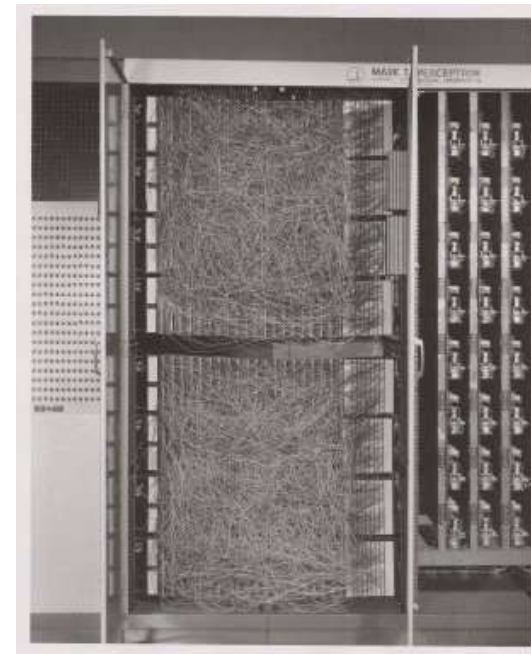
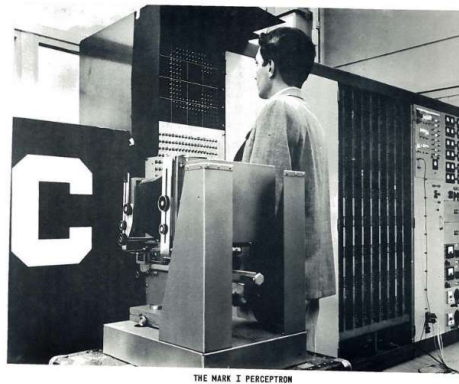


Rede Perceptron

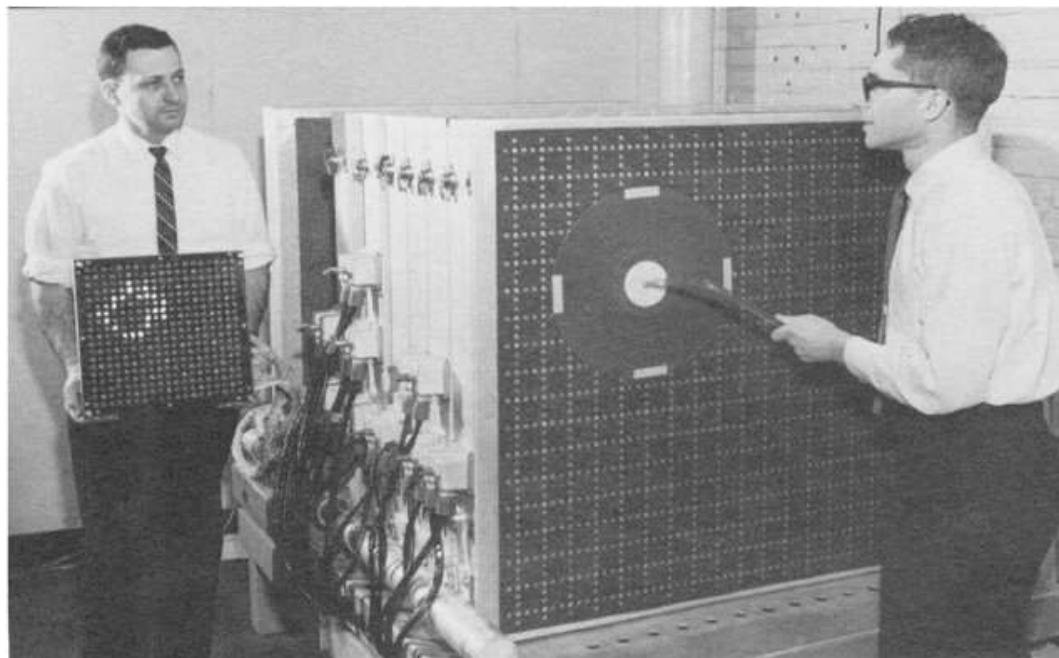


Rede Perceptron

- Primeira implementação:
 - Mark I Perceptron
 - Cornell Aeronautical Laboratory



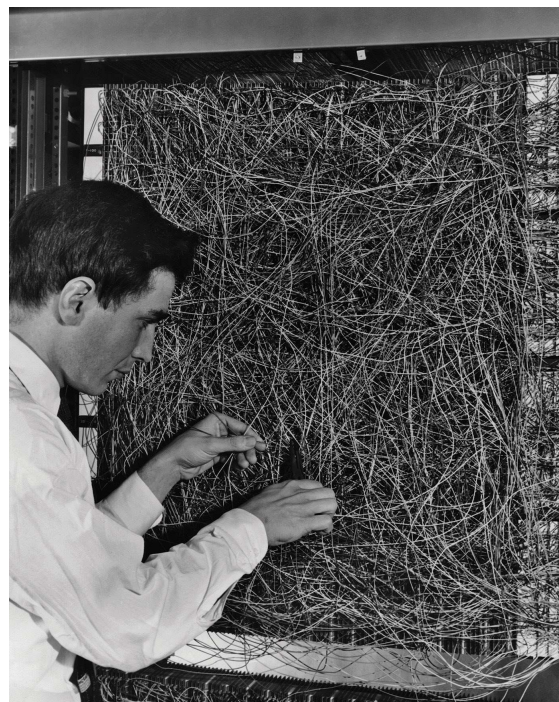
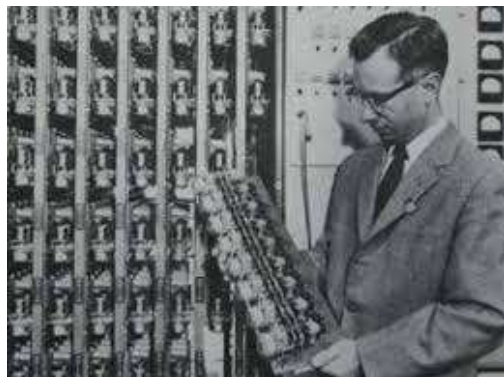
Começando a implementação



Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



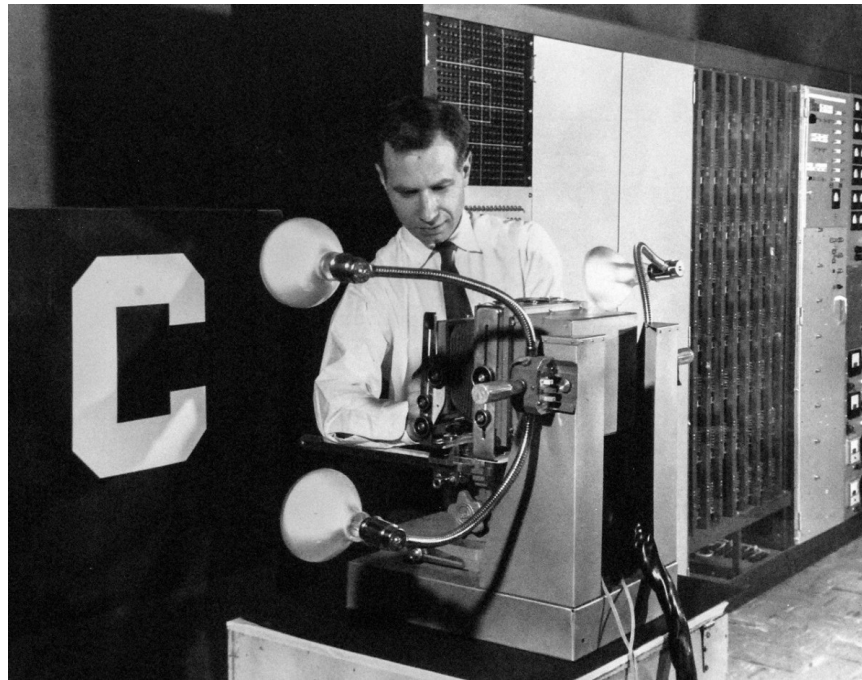
Preparando a rede Perceptron



Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



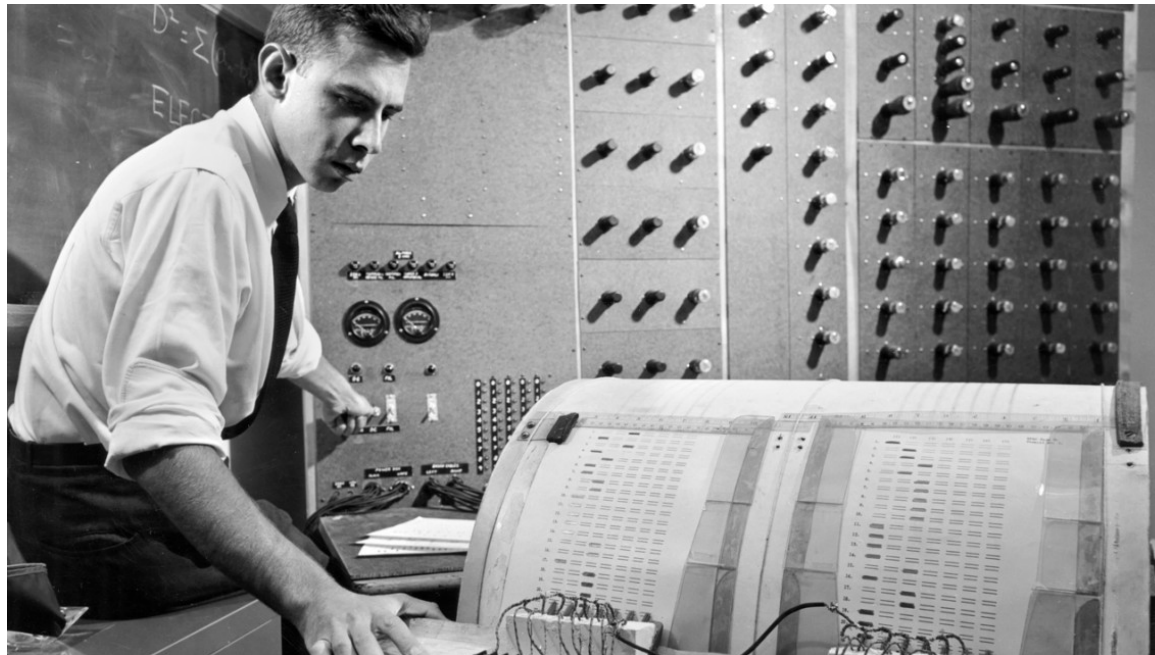
Finalizando a rede Perceptron



Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



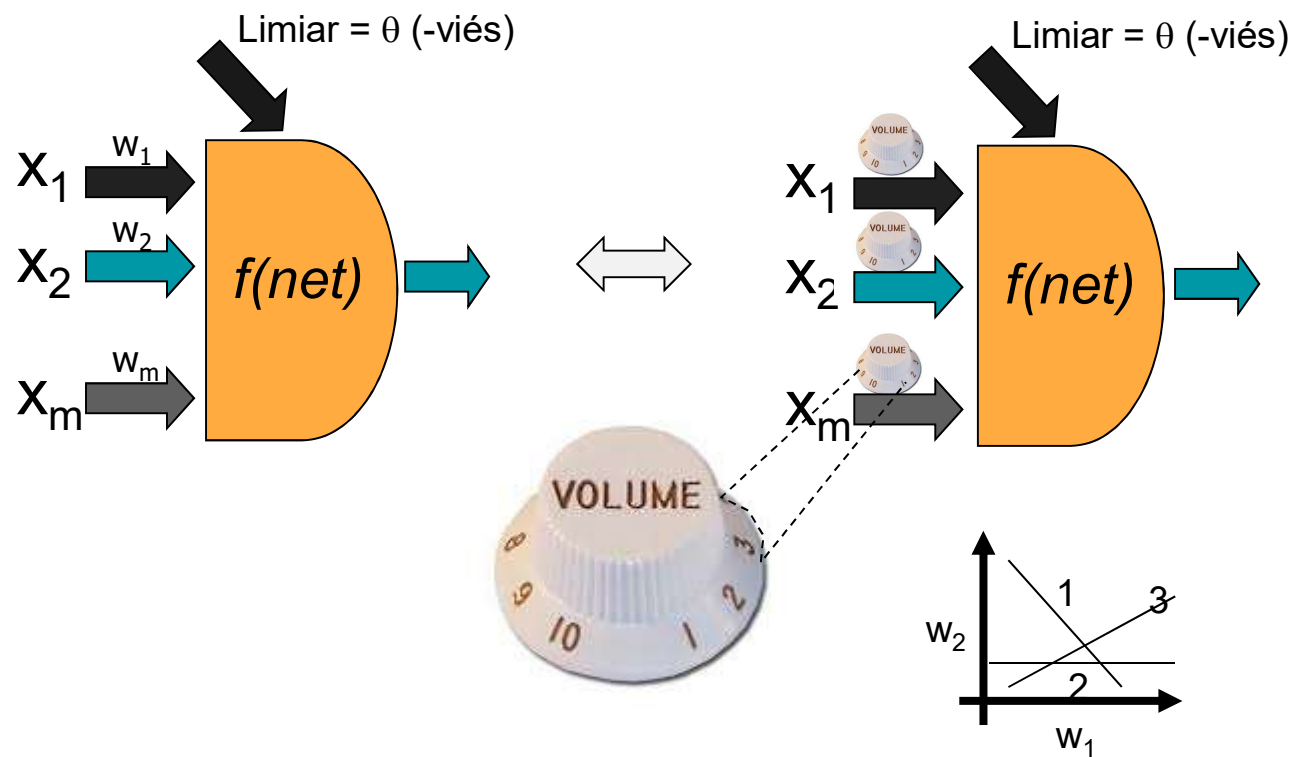
Perceptron funcionando



Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



Treinamento



Algoritmo de treinamento

1 Iniciar peso de cada conexão com o valor 0

2 Repita

Para cada par de treinamento (X, y)

Calcular a saída $f(X)$

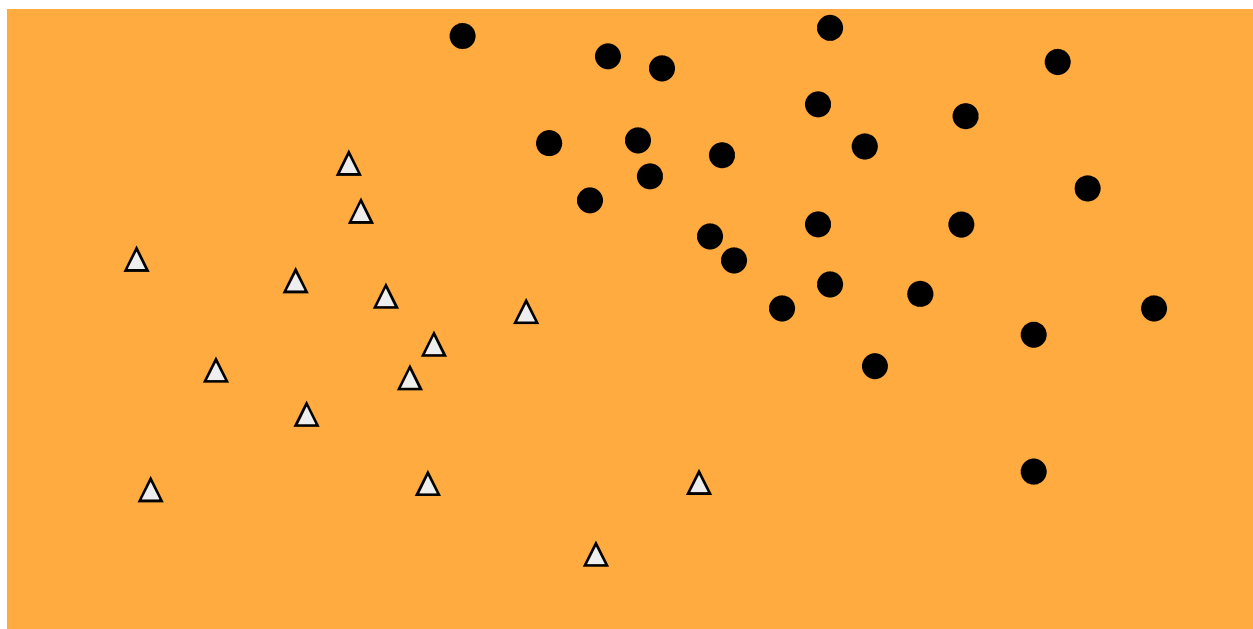
Se $(y \neq f(X))$

Então

Atualizar pesos do neurônio

Até condição de parada

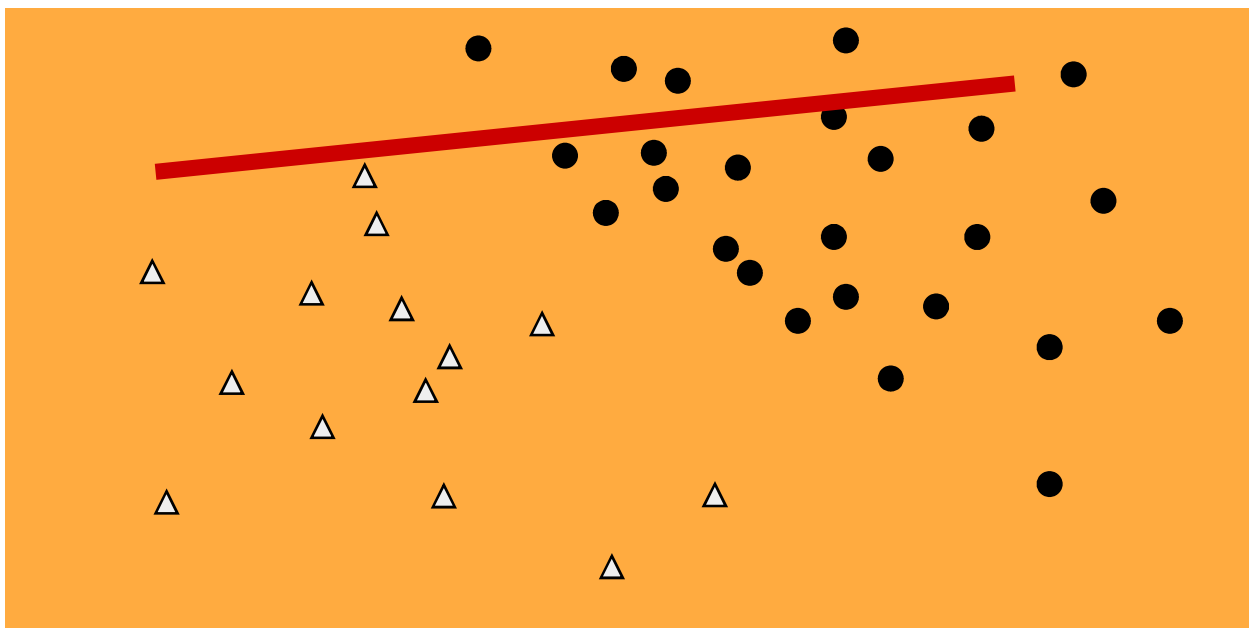
Treinamento modificando fronteiras



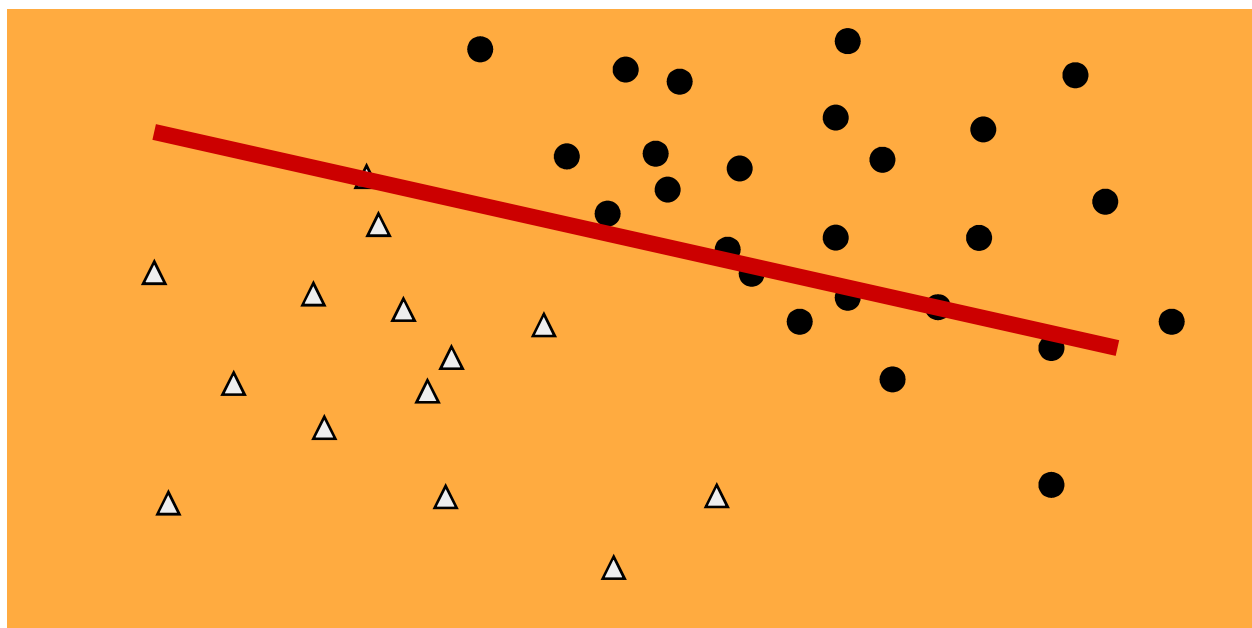
Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



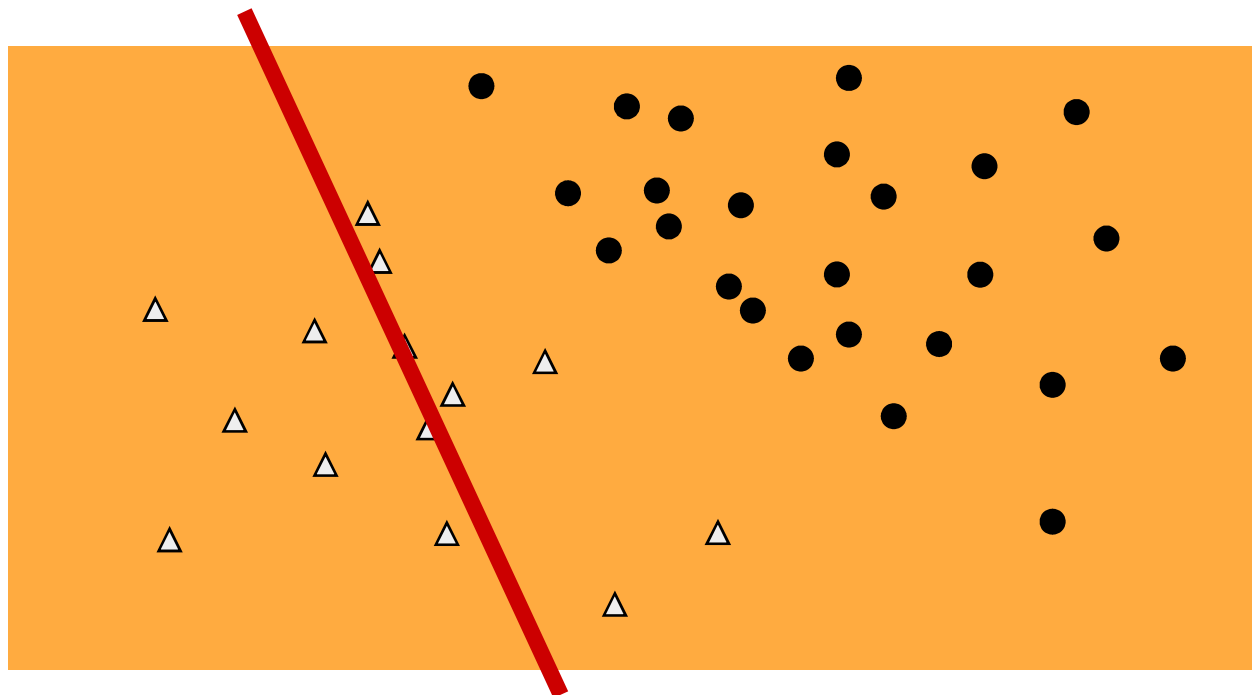
Treinamento modificando fronteiras



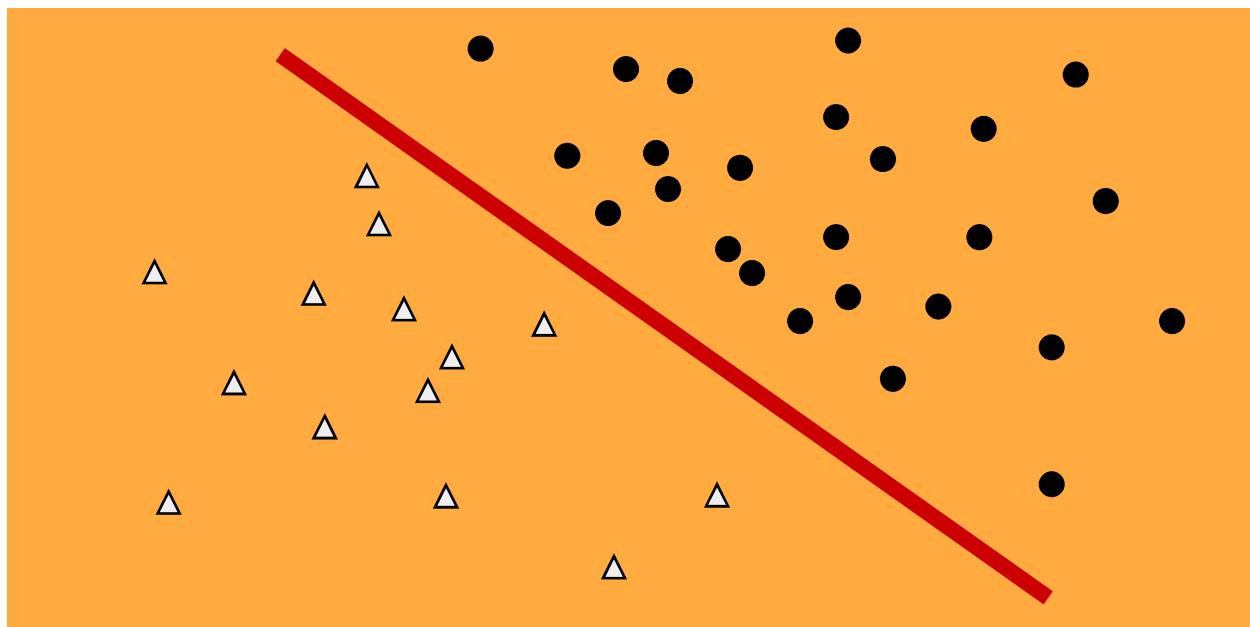
Treinamento modificando fronteiras



Treinamento modificando fronteiras



Treinamento modificando fronteiras



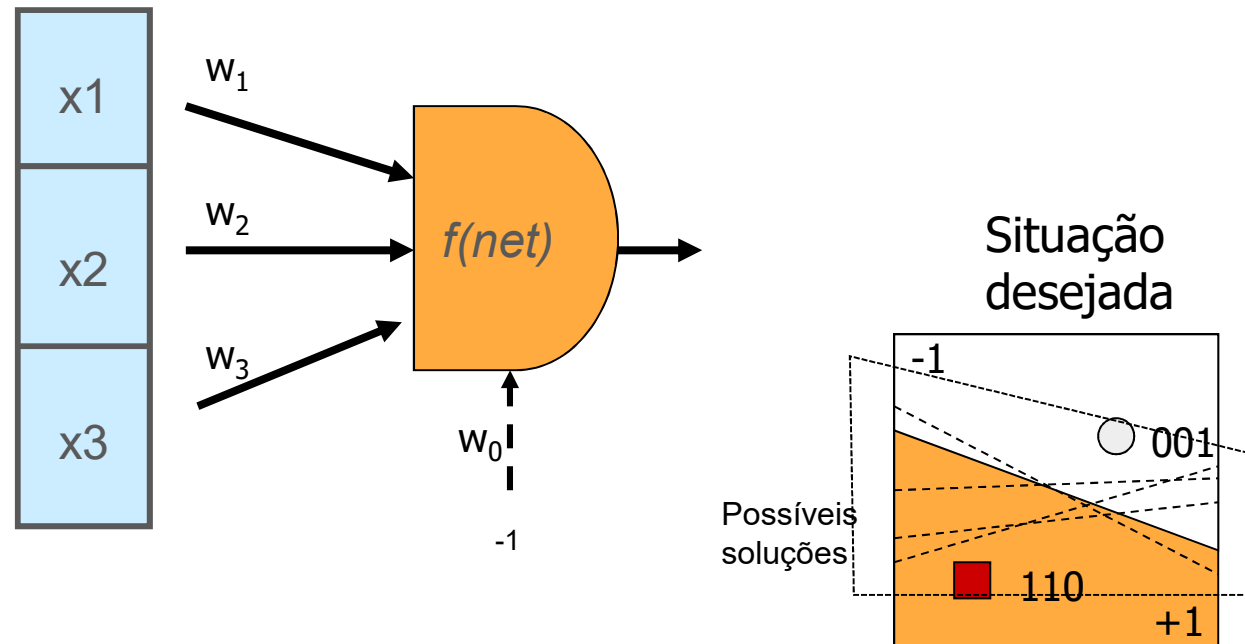
Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



Exemplo

- Dada uma rede Perceptron com:
 - Três entradas, pesos $w_1 = 0.4$, $w_2 = -0.6$ e $w_3 = 0.6$, e limiar $\theta = 0.5$:
 - Ensinar a rede com os exemplos (001, -1) e (110, +1)
 - Utilizar taxa de aprendizado $\eta = 0.4$
 - Predizer a classe dos exemplos: 111, 000, 100 e 011

Exemplo



Exemplo

- Treinar a rede $x_0, w_0(\theta), w_1, w_2, w_3, \eta = -1, 0.5, 0.4, -0.6, 0.6, 0.4$

1) Para o exemplo 001

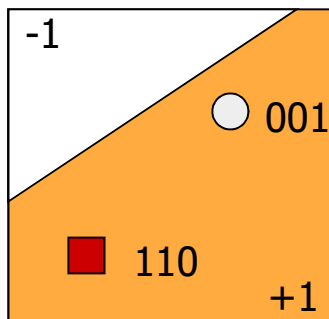
($y = -1$)

Passo 1: definir a saída da rede ($\sum xw$)

$$u - \theta = -1(0.5) + 0(0.4) + 0(-0.6) + 1(0.6) = 0.1$$

$$f(\text{net}) = +1 \text{ (uma vez que } 0.1 \geq 0 \text{)}$$

Passo 2: atualizar pesos ($y \neq f(\text{net})$)



$$w_0 = 0.5 + 0.4(-1)(-1 - (+1)) = 1.3$$

$$w_1 = 0.4 + 0.4(0)(-1 - (+1)) = 0.4$$

$$w_2 = -0.6 + 0.4(0)(-1 - (+1)) = -0.6$$

$$w_3 = 0.6 + 0.4(1)(-1 - (+1)) = -0.2$$

Exemplo

- Treinar a rede $x_0, w_0(\theta), w_1, w_2, w_3, \eta = -1, 1.3, 0.4, -0.6, -0.2, 0.4$

2) Para o exemplo 110

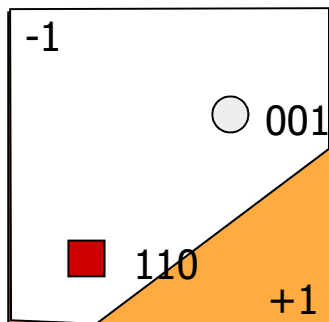
($y = +1$)

Passo 1: definir a saída da rede

$$u - \theta = -1(1.3) + 1(0.4) + 1(-0.6) + 0(-0.2) = -1.5$$

$$f(\text{net}) = -1 \text{ (uma vez que } -1.5 < 0)$$

Passo 2: atualizar pesos ($y \neq f(\text{net})$)



$$w_0 = 1.3 + 0.4(-1)(1 - (-1)) = 0.5$$

$$w_1 = 0.4 + 0.4(1)(1 - (-1)) = 1.2$$

$$w_2 = -0.6 + 0.4(1)(1 - (-1)) = 0.2$$

$$w_3 = -0.2 + 0.4(0)(1 - (-1)) = -0.2$$

Exemplo

- Treinar a rede $x_0, w_0(\theta), w_1, w_2, w_3, \eta = -1, 0.5, 1.2, 0.2, -0.2, 0.4$

3) Para o exemplo 001

($y = -1$)

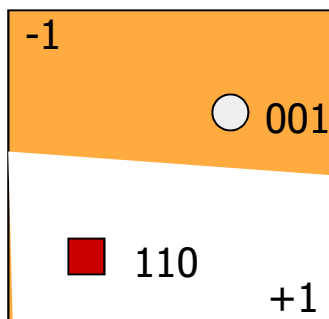
Passo 1: definir a saída da rede

$$u-\theta = -1(0.5) + 0(1.2) + 0(0.2) + 1(-0.2) = -0.7$$

$$f(\text{net}) = -1 \text{ (uma vez que } -0.7 < 0)$$

Passo 2: atualizar pesos ($y = f(\text{net})$)

Como $y = f(\text{net})$, os pesos não precisam ser modificados



Exemplo

- Treinar a rede $x_0, w_0(\theta), w_1, w_2, w_3, \eta = -1, 0.5, 1.2, 0.2, -0.2, 0.4$

4) Para o exemplo 110 ($y = +1$)

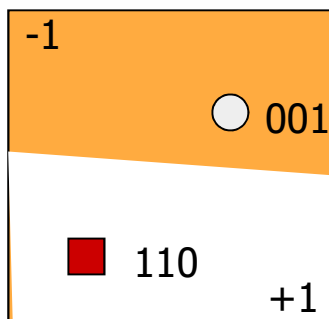
Passo 1: definir a saída da rede

$$u-\theta = -1(0.5) + 1(1.2) + 1(0.2) + 0(-0.2) = +0.7$$

$$f(\text{net}) = +1 \text{ (uma vez } 0.7 \geq 0 \text{)}$$

Passo 2: atualizar pesos ($y = f(\text{net})$)

Como $y = f(\text{net})$, os pesos não precisam ser modificados



Exemplo

- Utilizar a rede treinada para classificar os exemplos 111, 000, 100 e 011
 - Pesos aprendidos: 0.5, 1.2, 0.2, -0.2

b.1) Para o exemplo 111

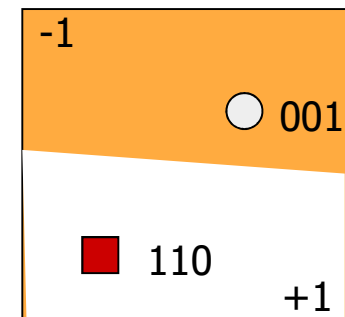
$$u-\theta = -1(0.5) + 1(1.2) + 1(0.2) + 1(-0.2) = 0.7$$

$$f(\text{net}) = +1 \text{ (porque } 0.7 \geq 0 \text{)} \Rightarrow \text{classe } +1$$

b.2) Para o exemplo 000

$$u-\theta = -1(0.5) + 0(1.2) + 0(0.2) + 0(-0.2) = -0.5$$

$$f(\text{net}) = -1 \text{ (porque } -0.5 < 0 \text{)} \Rightarrow \text{classe } -1$$



Exemplo

- Utilizar a rede treinada para classificar os exemplos 111, 000, 100 e 011
 - Pesos aprendidos: 0.5, 1.2, 0.2, -0.2

b.1) Para o exemplo 100

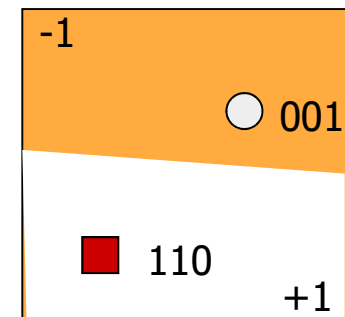
$$u-\theta = -1(0.5) + 1(1.2) + 0(0.2) + 0(-0.2) = 0.7$$

$$f(\text{net}) = +1 \text{ (porque } 0.7 \geq 0 \text{)} \Rightarrow \text{classe } +1$$

b.2) Para o exemplo 011

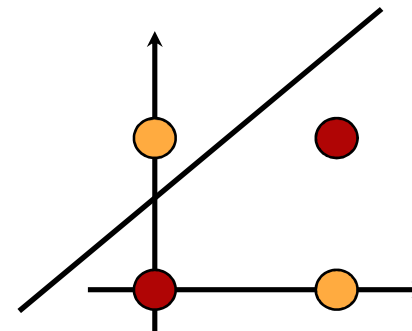
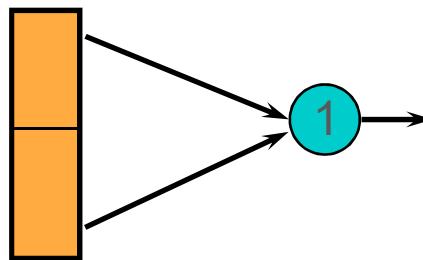
$$u-\theta = -1(0.5) + 0(1.2) + 1(0.2) + 1(-0.2) = -0.5$$

$$f(\text{net}) = -1 \text{ (porque } -0.5 < 0 \text{)} \Rightarrow \text{classe } -1$$



Problema da rede Perceptron

$0, 0 \rightarrow 0$
 $0, 1 \rightarrow 1$
 $1, 0 \rightarrow 1$
 $1, 1 \rightarrow 0$



Rede Adaline

- Problema do Perceptron: ajuste de pesos não leva em conta distância entre saída e resposta desejada
- Rede Adaline
 - Proposta por Widrow e Hoff em 1960
 - Utiliza modelo de McCulloch-Pitts como neurônio

Rede Adaline

- Estado de ativação
 - 1 = ativo
 - 0 = inativo
- Função de ativação
 - $a_i(t+1) = u_i(t)$
- Função de saída = função identidade

Rede Adaline

- Treinamento
 - Supervisionado
 - Correção de erro (regra LMS (delta, Widrow-Hoff)
 - $\Delta w_{ij} = \eta x_i (d_j - y_j)$ ($d \neq y$)
 - $\Delta w_{ij} = 0$ ($d = y$)
 - Implícito na primeira equação
 - Reajuste gradual do peso
 - Leva em conta distância entre saída e resposta desejada

Algoritmo de treinamento

1 Iniciar peso de cada conexão com o valor 0

2 Repita

Para cada par de treinamento (X, y)

Calcular a saída $f(X)$

Se $(y \neq f(X))$

Então

Atualizar pesos do neurônio

Até condição de parada

Fase de teste

*1 Para cada objeto de teste X faça
apresentar X a entrada da rede
calcular a saída y
se $(y < \lim_inf)$
então $X \in \text{classe } 0$
senão se $(y > \lim_sup)$
então $X \in \text{classe } 1$
senão indefinido*

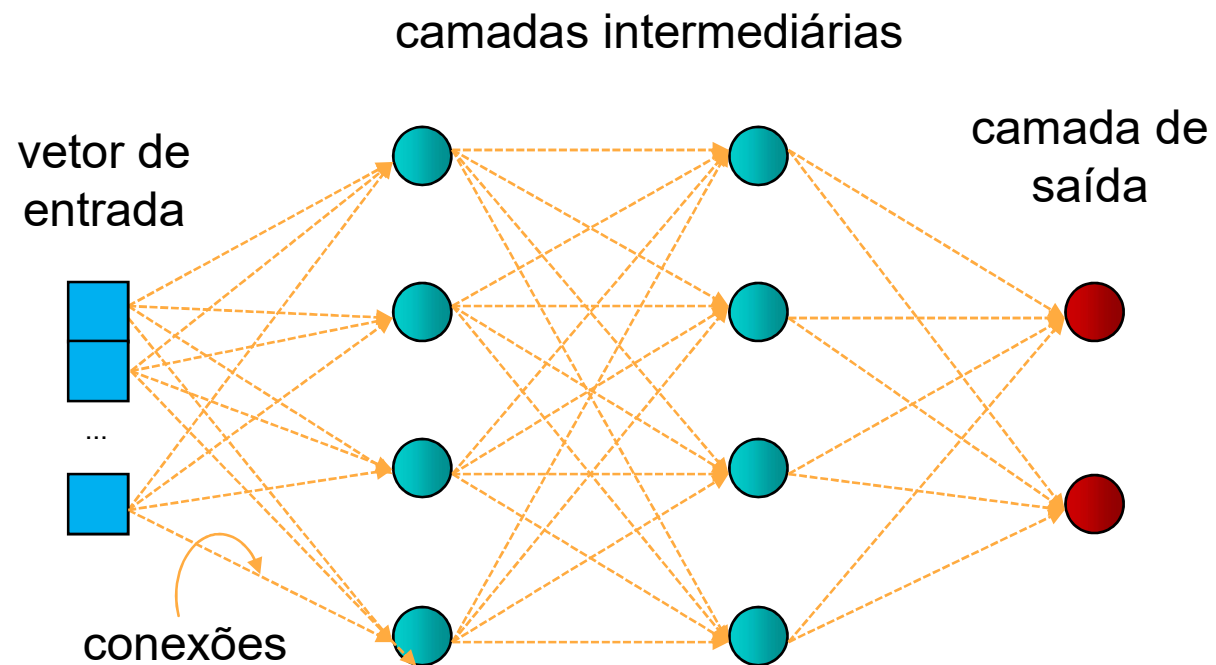
Rede madaline

- Trata algumas funções não linearmente separáveis
- Cada Adaline pode estar associado a uma reta
- Multicamadas
 - Primeira camada
 - Adaptativa
 - Várias redes Adaline
 - Segunda camada
 - Fixa
 - Funções AND ou maioria

Rede Multi-Layer Perceptron (MLP)

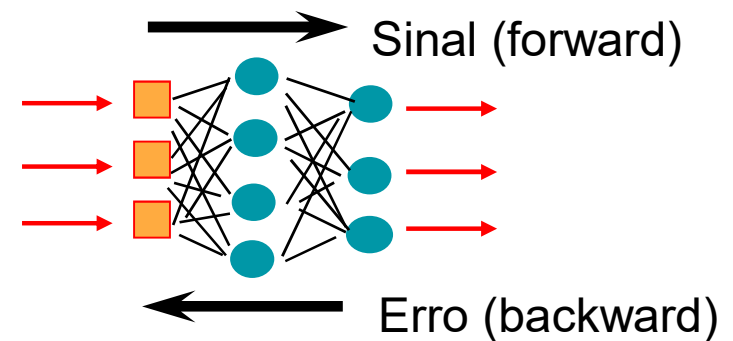
- Arquitetura de RNA mais utilizada
 - Uma ou mais camadas intermediárias de neurônios
- Funcionalidade (teórica)
 - Uma camada intermediária: qualquer função contínua ou Booleana
 - Duas camadas intermediárias: qualquer função
- Originalmente treinada com o algoritmo *backpropagation*

MLP e backpropagation



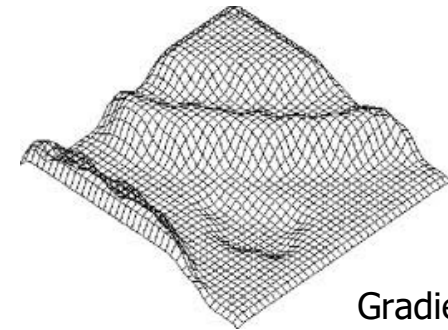
Backpropagation

- Treina a rede com pares entrada-saída
 - Cada vetor de entrada é associado a uma saída desejada
- Treinamento em duas fases, cada uma percorrendo a rede em um sentido
 - Fase forward
 - Fase backward
- Rumelhart, Hinton e Williams (1986)
 - Werbos (1974)



Backpropagation

- Treinamento
 - Supervisionado
 - Ajuste dos pesos: $\Delta w_{ij} = \eta x_i \delta_j$



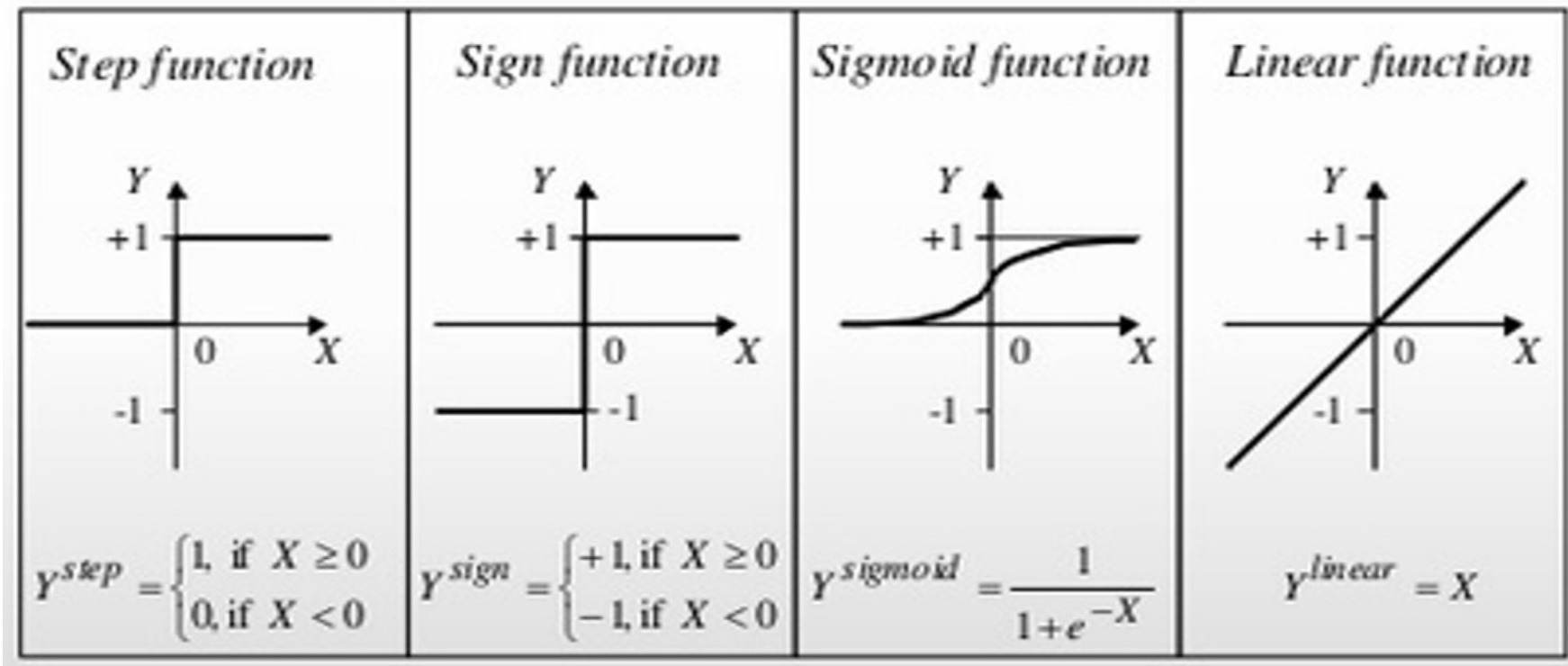
Gradiente
da função

$$\delta_j = \begin{cases} f'(net)erro_j & \text{se } j \text{ for camada de saída} \\ f'(net)\sum w_{jk}\delta_k & \text{se } j \text{ for camada intermediária} \end{cases}$$

$$erro_j = \frac{1}{2} \sum_{q=1}^c (y_q - f(net_q))$$

$$net = \sum_{i=0}^m x_i w_i$$

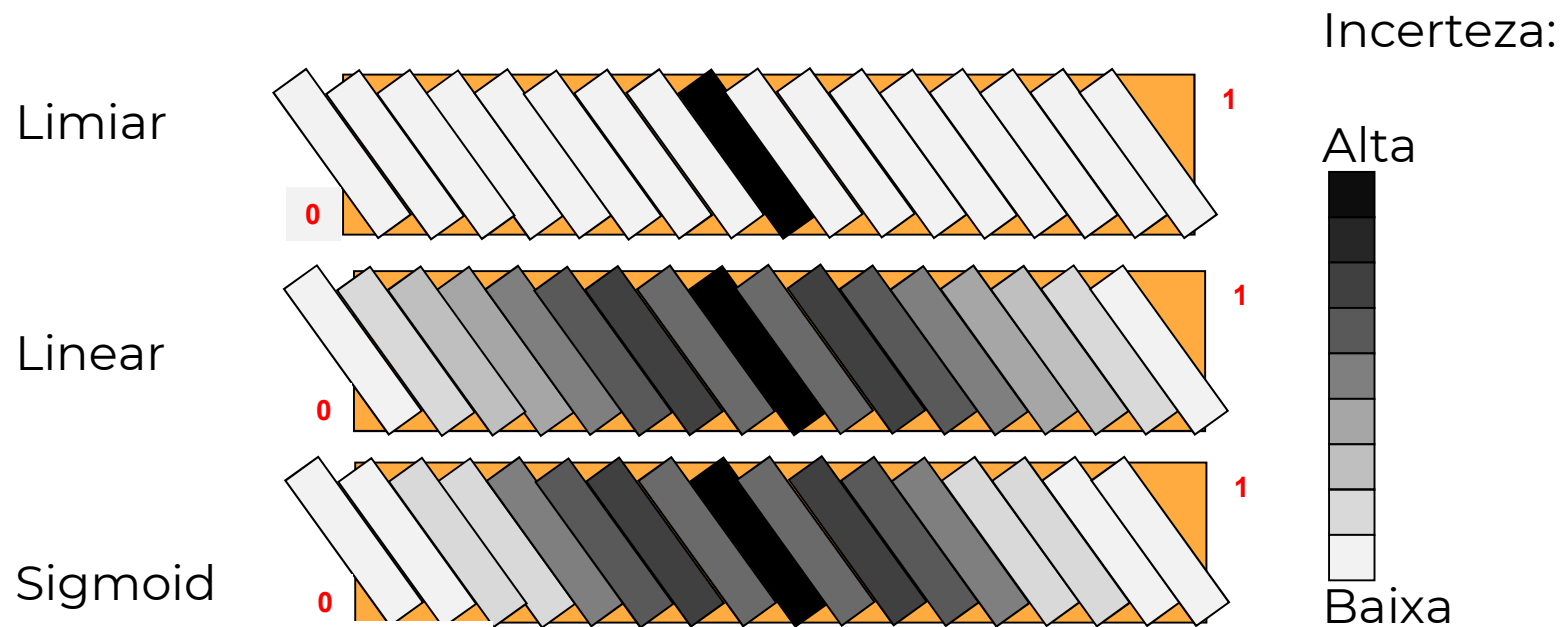
Funções de ativação



Funções de ativação linear

- Redes com função de ativação linear são mais fáceis de treinar
 - Derivada é uma constante
- Generalizam bem, mas não permitem aprender funções complexas
 - Ainda são usadas na camada de saída
 - Usando manipulação de matrizes, é possível mostrar que rede com várias camadas pode ser reduzida a uma camada
- Funções diferenciáveis mais usadas são sigmoide e tangente hiperbólica
 - Até o início dos anos 1990 era a função default, substituída depois pela tanh
 - Mais fácil de treinar e em geral produzia melhores resultados

Funções de ativação e fronteiras de decisão



Algoritmo de treinamento

Iniciar todas as conexões com valores aleatórios $\in [a,b]$

Repita

erro = 0;

Para cada par de treinamento (X, y)

Para cada camada $k := 1$ a N

Para cada neurônio $j := 1$ a M_k

Calcular a saída $f_{kj}(\text{net})$

Se $k = N$

Então Calcular soma dos erros dos neurônios da camada;

Se erro $> \varepsilon$

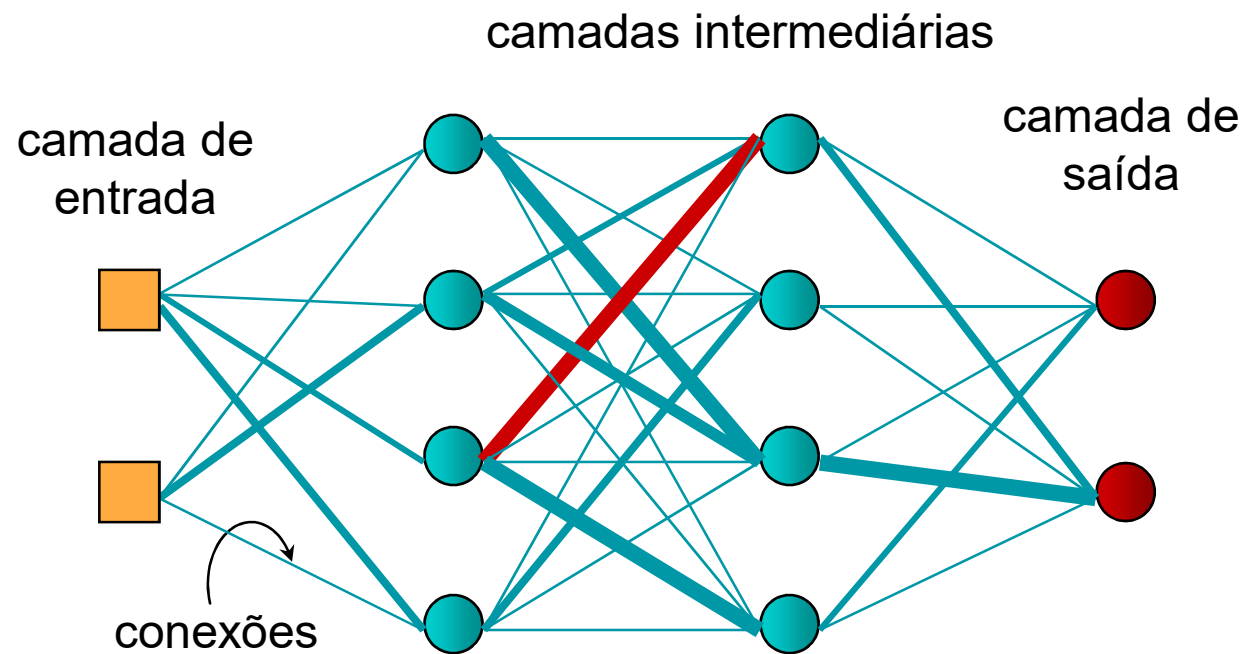
Então Para cada camada $k := N$ a 1

Para cada neurônio $j := 1$ a M_k

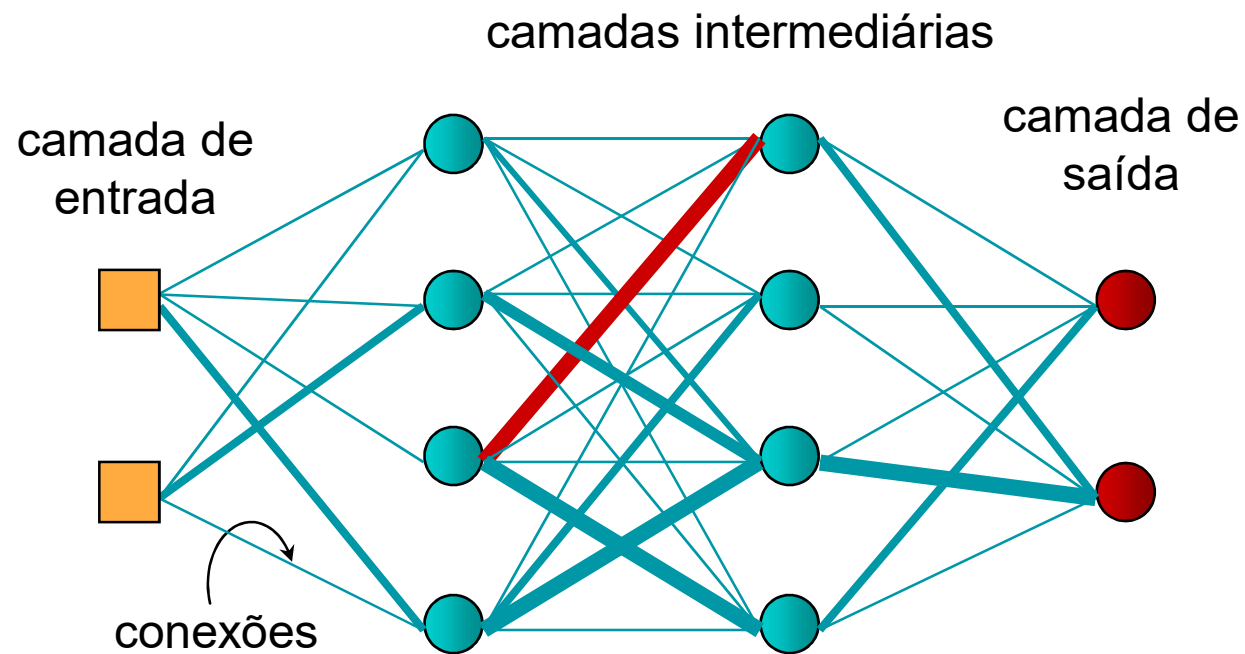
Atualizar pesos;

Até erro $< \varepsilon$ (ou número máximo de ciclos)

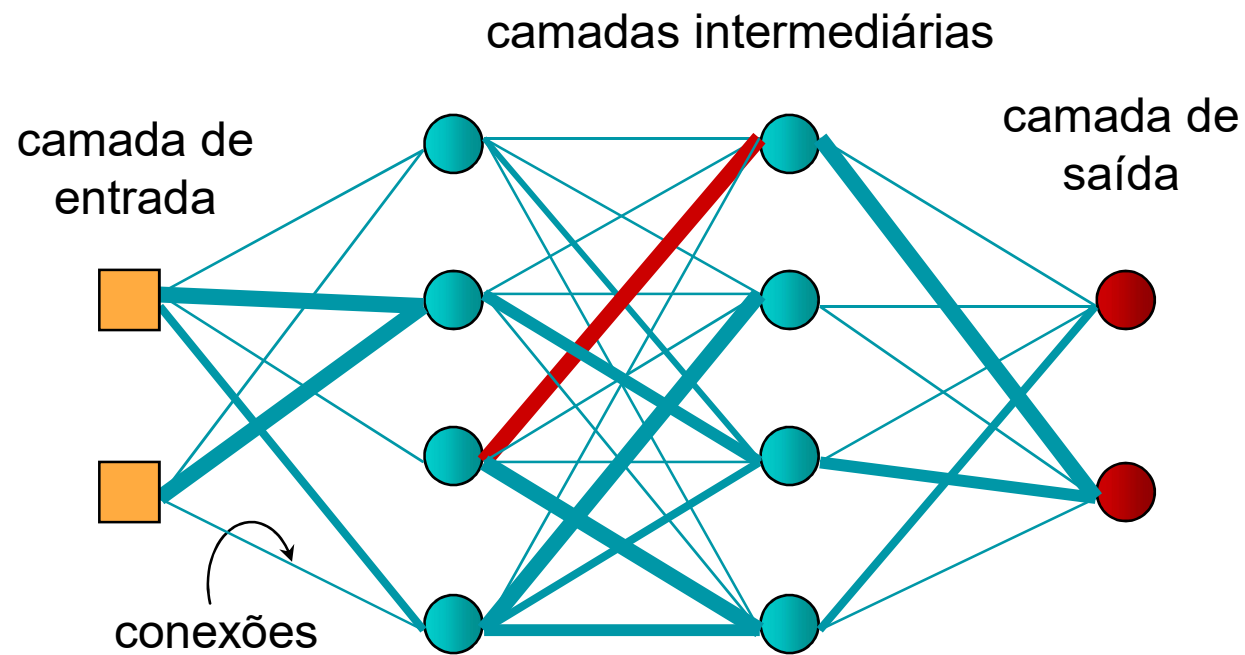
Treinamento modificando pesos



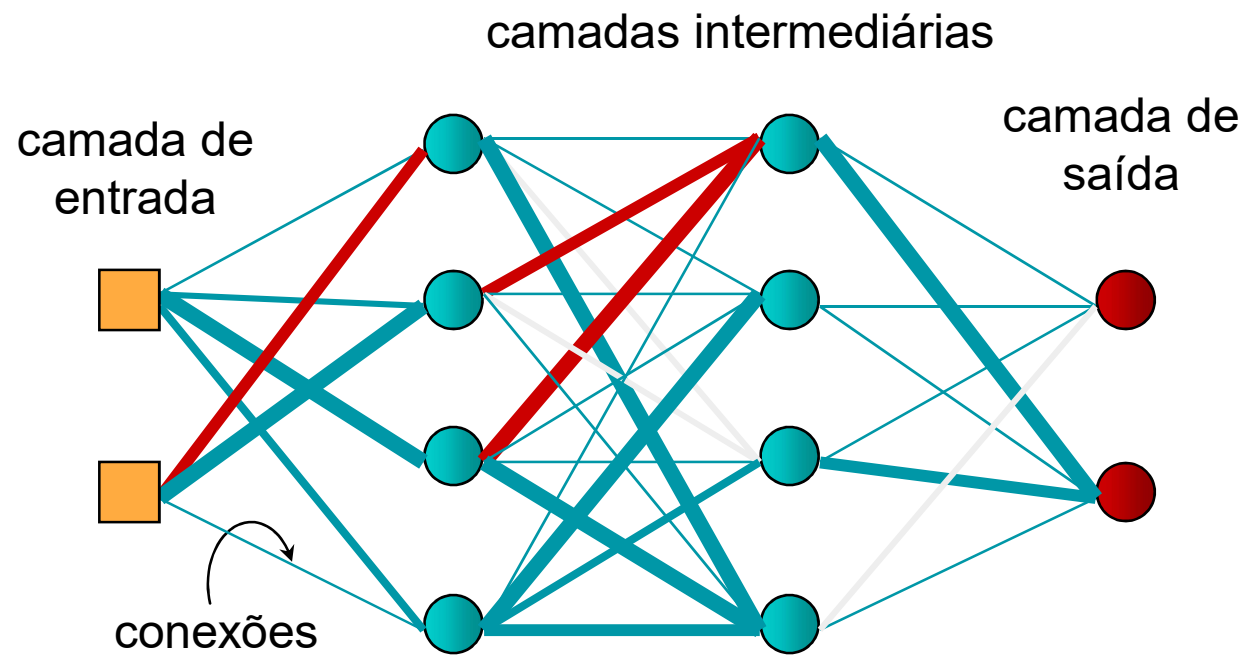
Treinamento modificando pesos



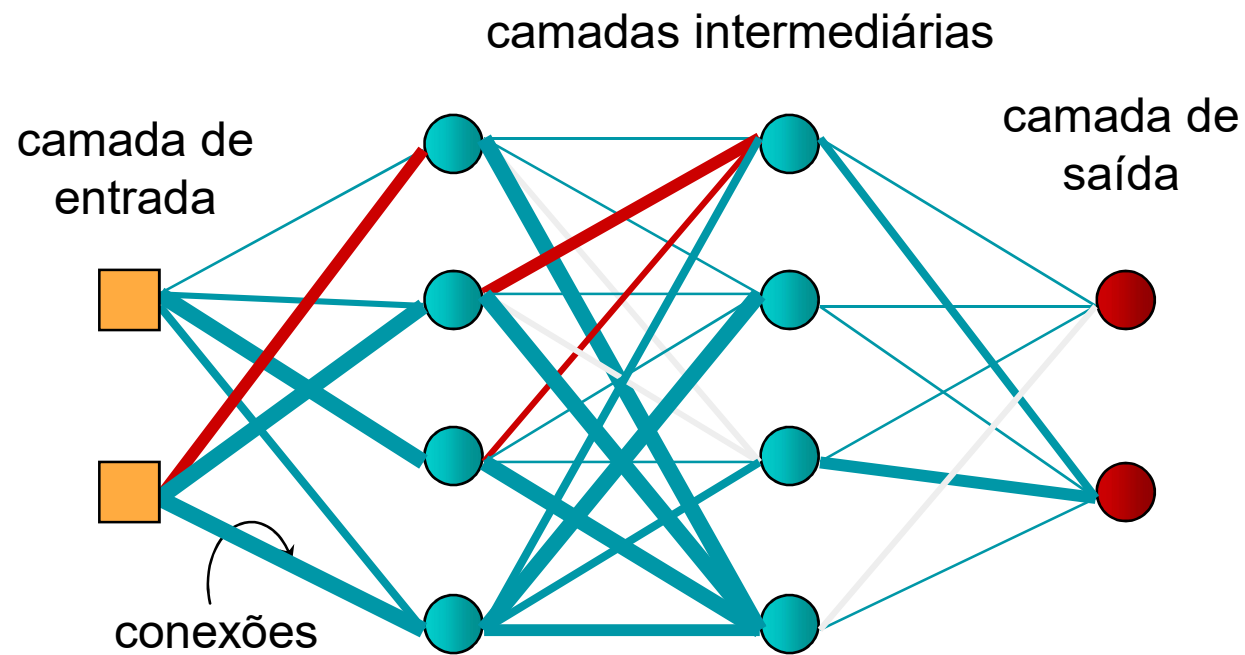
Treinamento modificando pesos



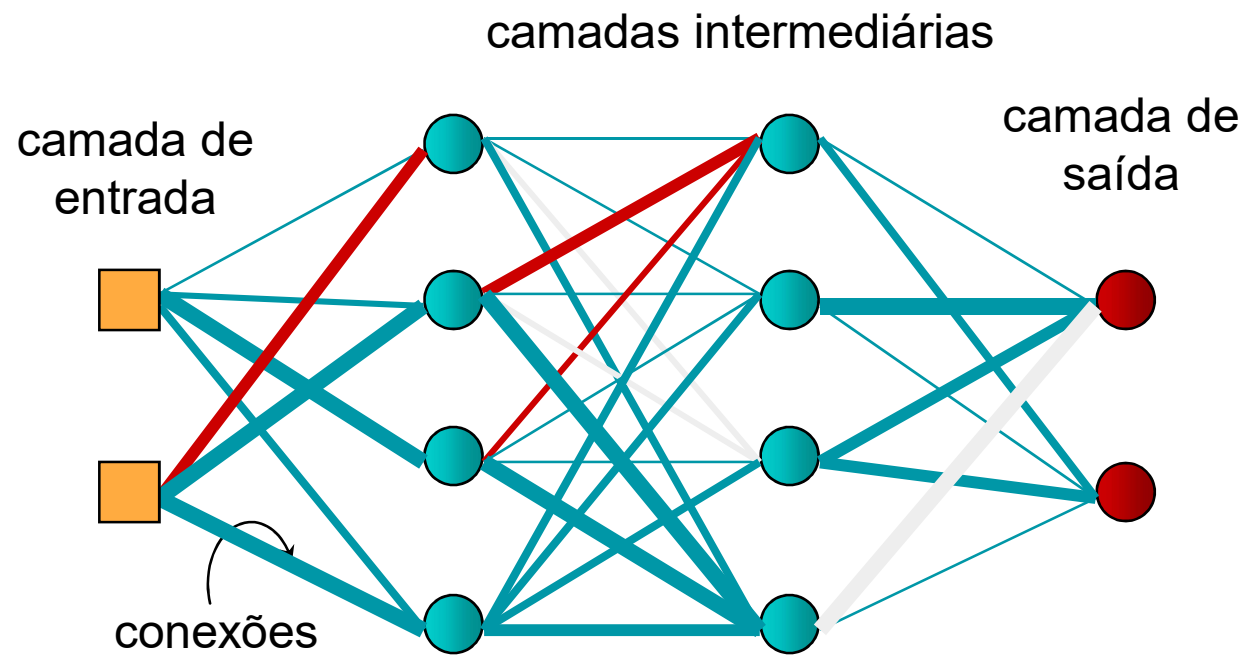
Treinamento modificando pesos



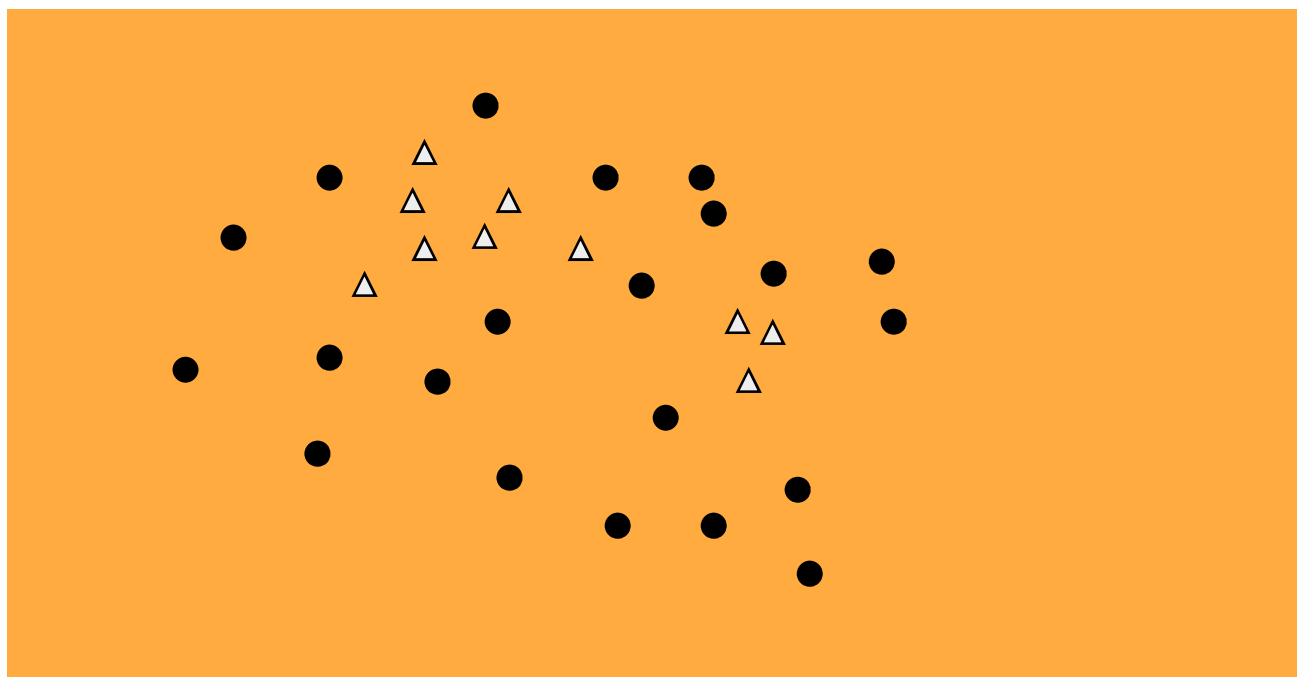
Treinamento modificando pesos



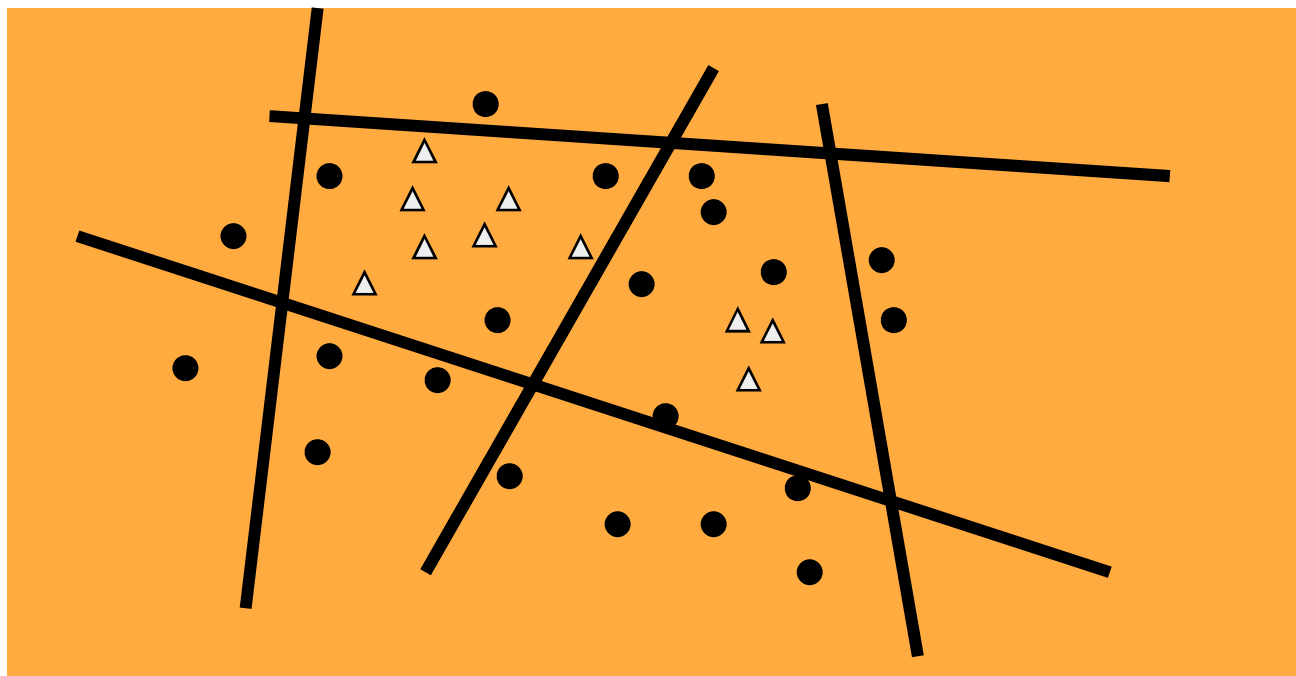
Treinamento modificando pesos



Treinamento modificando fronteiras



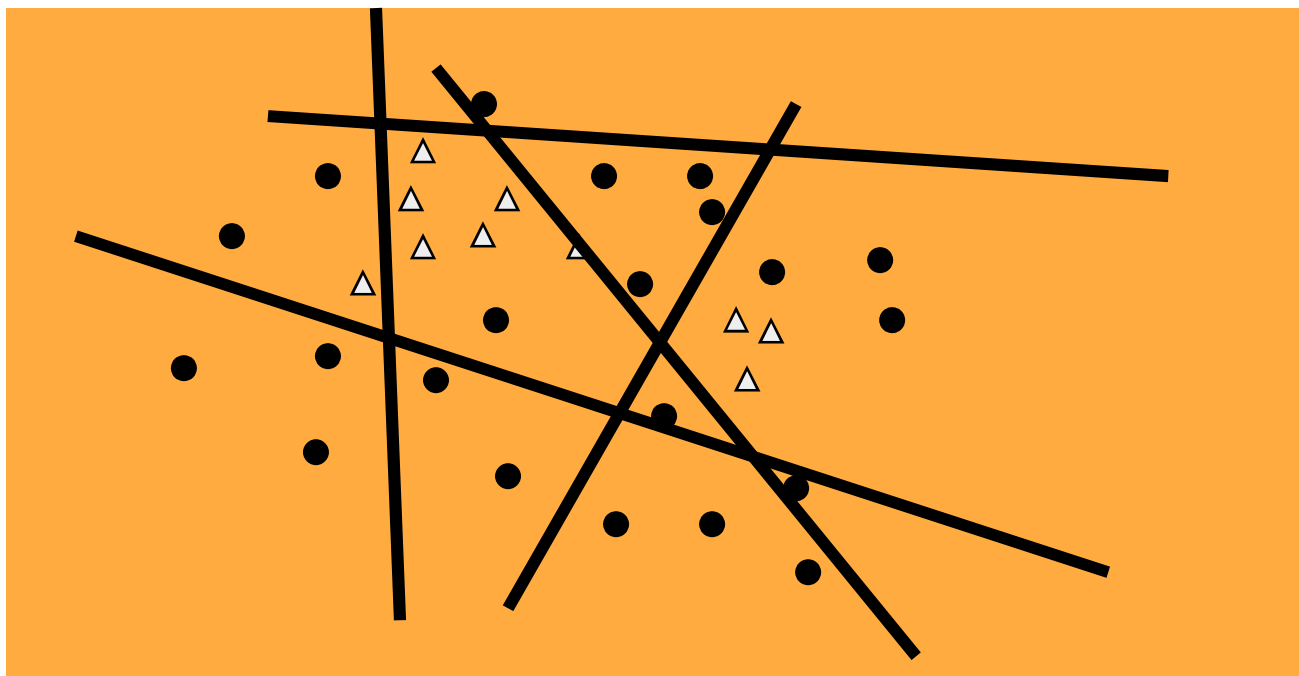
Treinamento modificando fronteiras



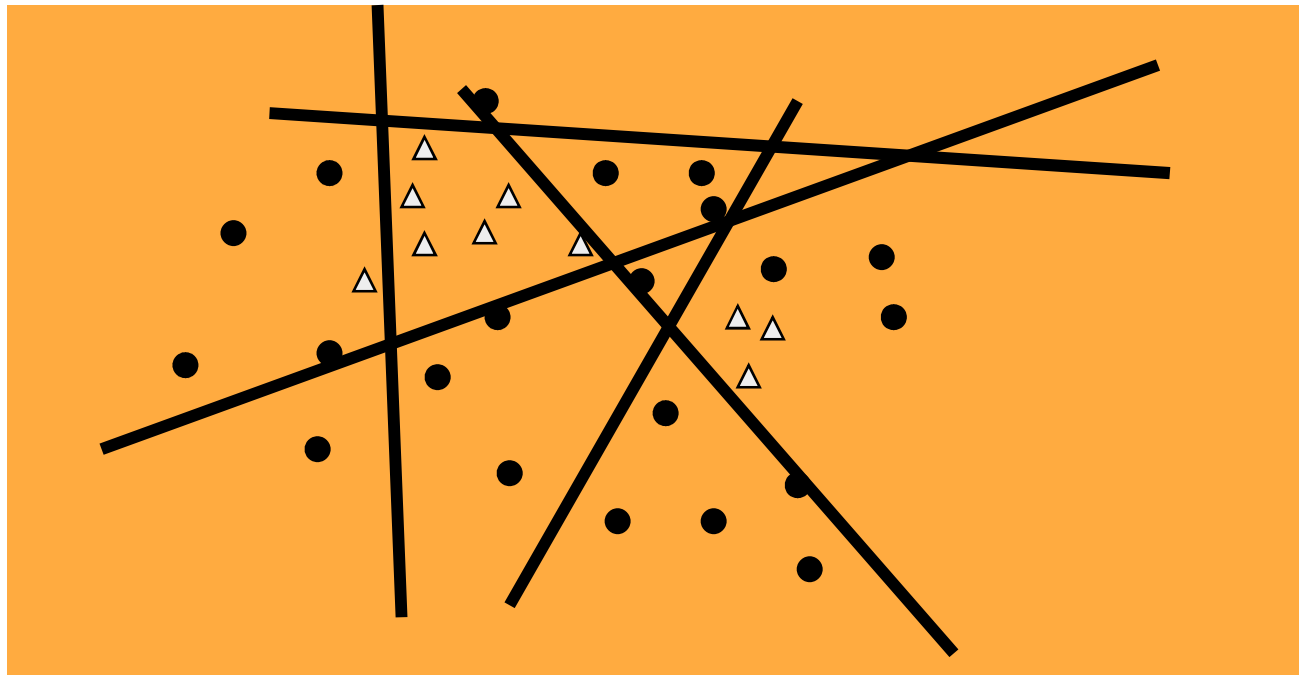
Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



Treinamento modificando fronteiras



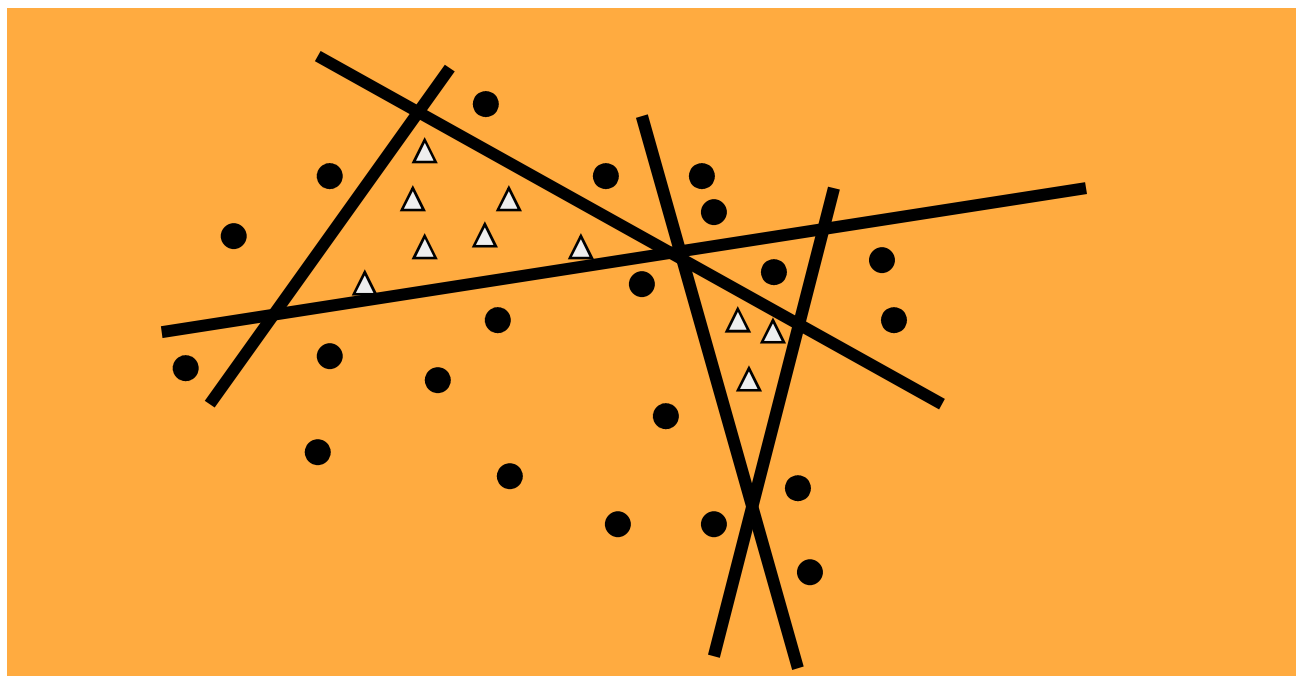
Treinamento modificando fronteiras



Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



Treinamento modificando fronteiras



Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização

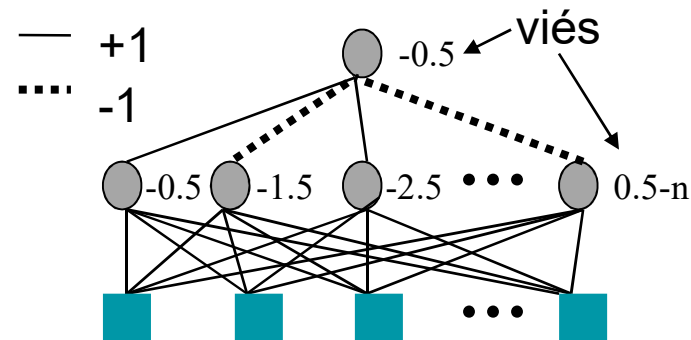


Exercício

Dada a rede abaixo, que recebe como entrada um vetor binário de n bits e gera como saída um valor binário:

- Indicar a função implementada pela rede abaixo:
- Explicar papel de cada neurônio no processamento da função

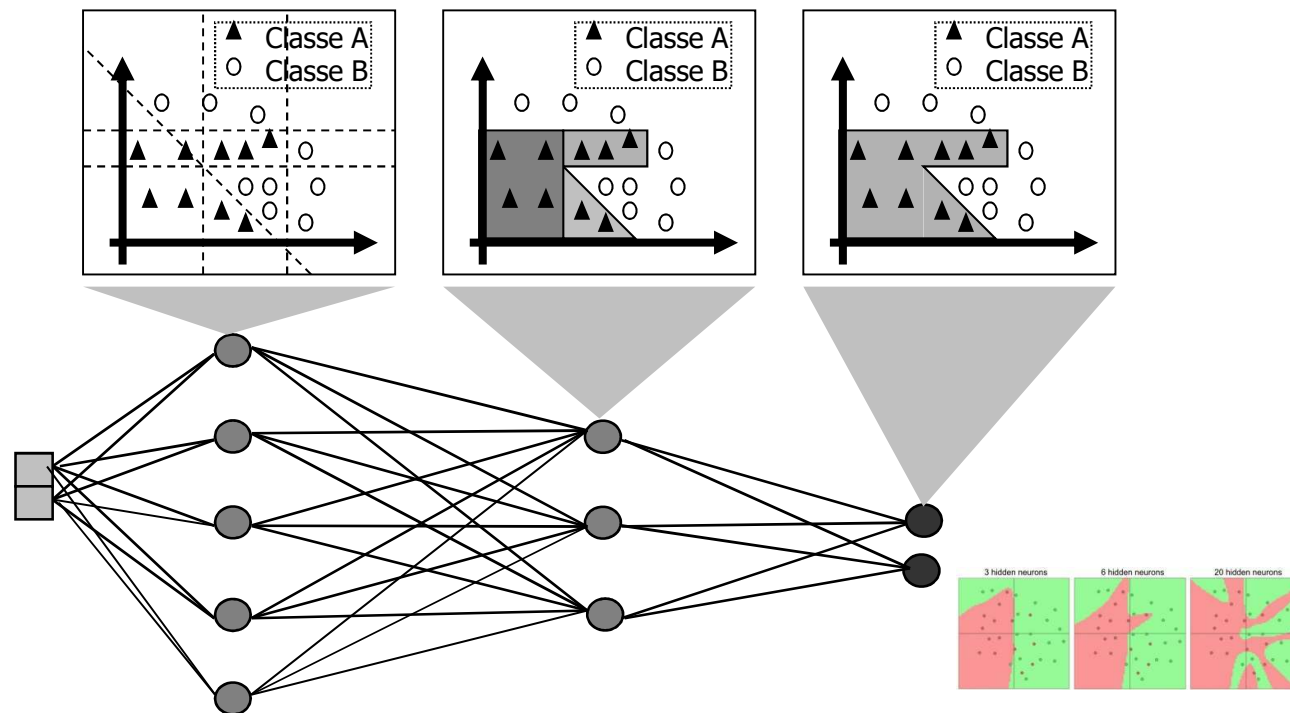
Considerar função de ativação limiar (threshold) entrada/saída binária



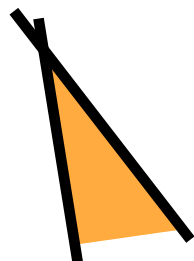
Exercício

- Paridade
 - Uma das limitações do Perceptron levantadas por Minsky e Papert
- Problema difícil
 - Padrões mais semelhantes requerem respostas diferentes
 - Usa n unidades intermediárias para detectar paridade em vetores com n bits

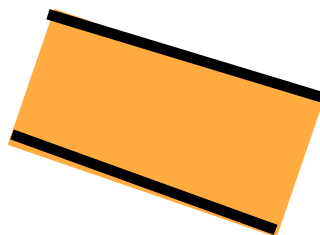
MLPs como classificadores



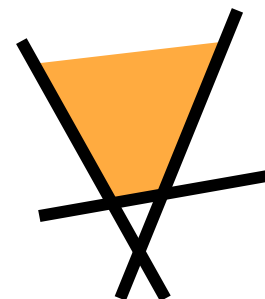
Regiões convexas



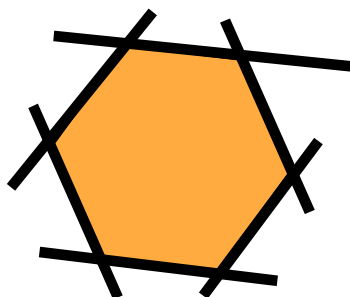
Aberta



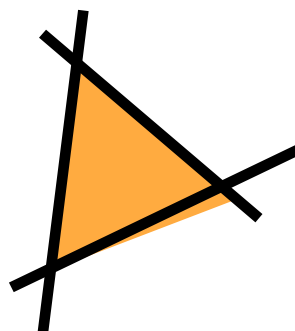
Aberta



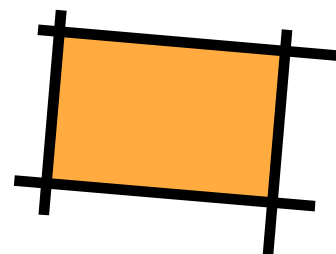
Aberta



Fechada

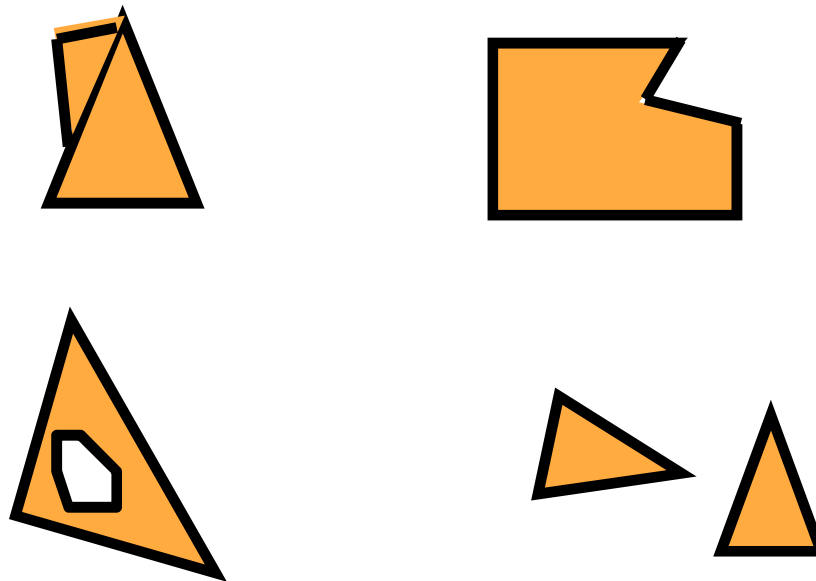


Fechada



Fechada

Combinações de regiões convexas

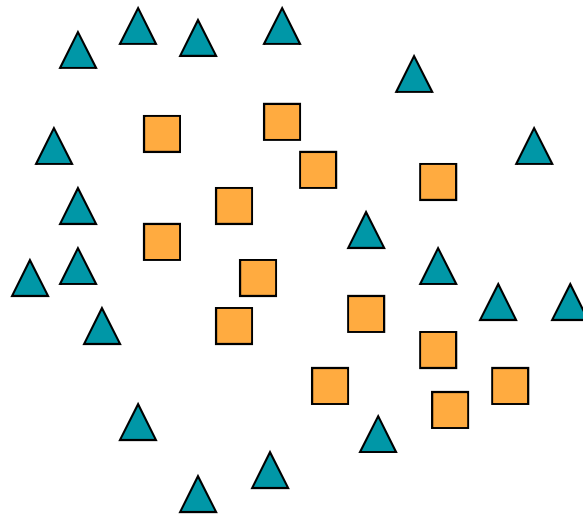


Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização



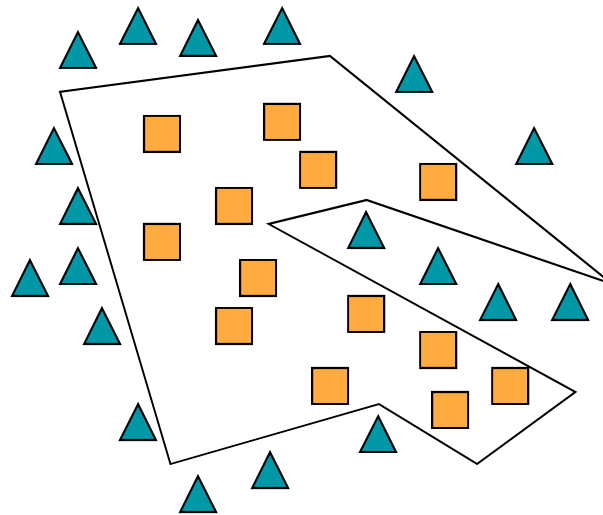
Combinações de regiões convexas

- Encontrar fronteiras de decisão que separem os dados abaixo:












Combinações de regiões convexas

- Encontrar fronteiras de decisão que separem os dados abaixo:



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)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric 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		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Copyright © 2020. Todos os direitos reservados ao CeMEAI-USP. Proibida a cópia e reprodução sem autorização







Função Sigmoide e Tangente Hiperbólica

- Problema das funções sigmoide e tangente hiperbólica é a saturação nos extremos
 - Sensíveis a mudanças apenas para valores próximos ao meio
 - Dificuldade para treinar redes com muitas camadas
 - Problema do gradiente desaparecendo (vanishing gradient problem)
 - Camadas mais próximas da entrada não recebem informação gradiente útil
 - Erros retro-propagados para as camadas de trás decresce reduz drasticamente a cada nova camada adicionada
 - Causado pela derivada da função de ativação
 - Faz com que redes com muitas camadas não aprendam de forma eficiente

Função Rectified Linear Unit (ReLU)

- Nos últimos anos, funções sigmoide e tangente hiperbólica têm sido substituídas por funções de unidade linear retificada ReLU
 - ReLU pode evitar o problema de gradiente decrescente
- Atualmente utilizada em vários tipos de redes neurais
 - Mais fáceis de treinar
 - Por serem quase lineares, preservam propriedades que fazem modelos lineares mais fáceis de otimizar por métodos baseados no gradiente
 - Muitas vezes obtêm melhores resultados
 - Por preservarem muitas das propriedades que fazem com que modelos lineares tenham boa capacidade de generalização

Funções de ativação

Nome	Figura	Função	Derivada
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric 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		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Conclusão

- Redes Neurais
 - Sistema nervoso
 - Muito utilizadas em problemas reais
 - Várias arquiteturas e algoritmos de treinamento
 - Magia negra
 - Caixa preta

Fim do
apresentação