Redes Neurais

Prof. Aurora Pozo

Obs: slides baseados em

Prof. Marcílio Souto e Prof. Marley Vellasco

CONTEÚDO

- **Introdução**
  - Motivação, Objetivo, Definição, Características Básicas e Histórico
- **Conceitos Básicos**
  - Neurônio Artificial, Modos de Interconexão
- **Processamento Neural**
  - *Recall e Learning*
- **Regras de Aprendizado**
  - Regra de Hebb, Perceptron, Back Propagation, Hopfield e Competitive Learning

MOTIVAÇÃO

Constatação que o cérebro processa informações de forma diferente dos computadores convencionais

CÉREBRO	COMPUTADOR
velocidade 1 milhão de vezes mais lenta que qualquer gate digital	processamento extremamente rápido e preciso na execução de sequência de instruções

Processamento altamente paralelo
(10^{11} neurônios com 10^{4} conexões cada)

Problema dos 100 Passos

Neurônio: 2ms

Processador: 2ns

Processador é 10^6 mais rápido que o neurônio

Cérebro reage a um estímulo entre 0,2 e 1 seg.

O cérebro responde em 100 passos

MOTIVAÇÃO

- ***Observações:***
  - O cérebro tem ~ ***10 bilhões*** de neurônios.
  - Cada neurônio tem ~ ***1.000*** a ***10.000*** conexões

    ↓
  - ***60 trilhões*** de conexões - ***10^{14} sinapses***!

    ↓
  - Cada pessoa pode dedicar ***100.00 conexões*** para armazenar cada segundo de experiência

    (65 anos ⇒ 2.000.000.000 de segundos!)
  - Durante os 2 primeiros anos de vida, ***1.000.000 de sinapses*** são formadas por segundo!!

Objetivo

- **Estudar a teoria e a implementação de sistemas *massivamente paralelos, que* possam processar informação com *eficiência comparável ao cérebro.***

Definição

- **Redes Neurais Artificiais são sistemas inspirados nos *neurônios biológicos e* na *estrutura massivamente paralela do* cérebro, com capacidade de adquirir, armazenar e utilizar conhecimento experimental.**

Aquisição de Conhecimento: Aprendizado

Treinamento efetuado através da apresentação de exemplos

↓

Existe uma variedade de algoritmos que estabelecem QUANDO e COMO os parâmetros da Rede Neural devem ser atualizados

↓

Substitui a programação necessária para a execução das tarefas nos computadores

APLICAÇÕES GERAIS

- ☑ Reconhecimento de Padrões
- ☑ Classificação de Padrões
- ☑ Correção de Padrões
- ☑ Previsão de Séries Temporais
- ☑ Aproximação de Funções
- ☑ Suporte à Decisão
- ☑ Extração de Informações

Características Básicas

- **Devido à similaridade com a estrutura do cérebro, as Redes Neurais exibem características similares ao do comportamento humano, tais como:**

Características Básicas

- **<u>Procura Paralela e Endereçamento pelo Conteúdo:</u>**

O cérebro **não possui endereço de memória e não procura a** informação **seqüencialmente**

Aprendizado

- A rede ***aprende por experiência, não*** necessitando explicitar os algoritmos para executar uma determinada tarefa

Características Básicas

- **Associação**:

  A rede é capaz de fazer associações entre padrões diferentes

  Ex: Pessoa → Nome

  Perfume → Pessoa

Características Básicas

- **Generalização**:

Redes Neurais são capazes de ***generalizar o seu conhecimento*** a partir de exemplos anteriores

→ Habilidade de lidar com ***ruídos e distorções***, respondendo corretamente a padrões novos.

Características Básicas

- **Abstração**:

  Capacidade de ***abstrair a essência de um conjunto de entradas***, isto é, a partir de padrões ruidosos, extrair a informação do padrão sem ruído.

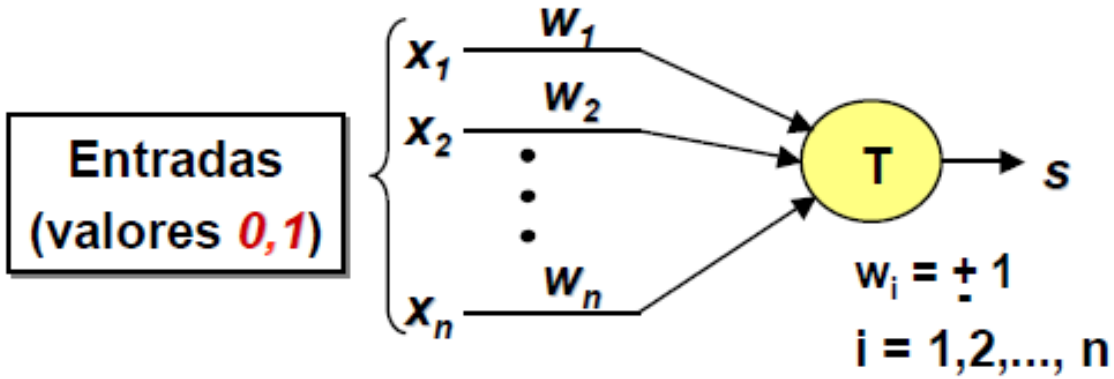
Características Básicas

- **Robustez e Degradação Gradual:**

A perda de um conjunto de elementos processadores não causa o mal funcionamento da rede neural

EVOLUÇÃO

- ***Modelo de McCulloch-Pitts:***



$$s^{k+1} = \begin{cases} 1 \text{ se } \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 \text{ se } \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$

EVOLUÇÃO

McCulloch & Pitts (Mathematical Bio-Physics, Vol. 5, 1943),
"A Logical Calculus of Ideas Immanent in Nervous Activity"

	Von Neumann	Marvin Minsky	Frank Rosenblatt
	Machine Intelligence	***Macroscopic Intelligence***	***Microscopic Intelligence***
1945	Digital Computers		
1950		Black-Box AI (LISP)	Perceptron, Adaline
1960	Mainframes	Theorem Solver	
1970	Vax 780 (Time Sharing)	Expert Systems	
1980	Workstations, PCs	Commercialization of E.S.	Rumelhart, Hopfield
1990	Desktop Supercomputers		Commercialization of N.N.

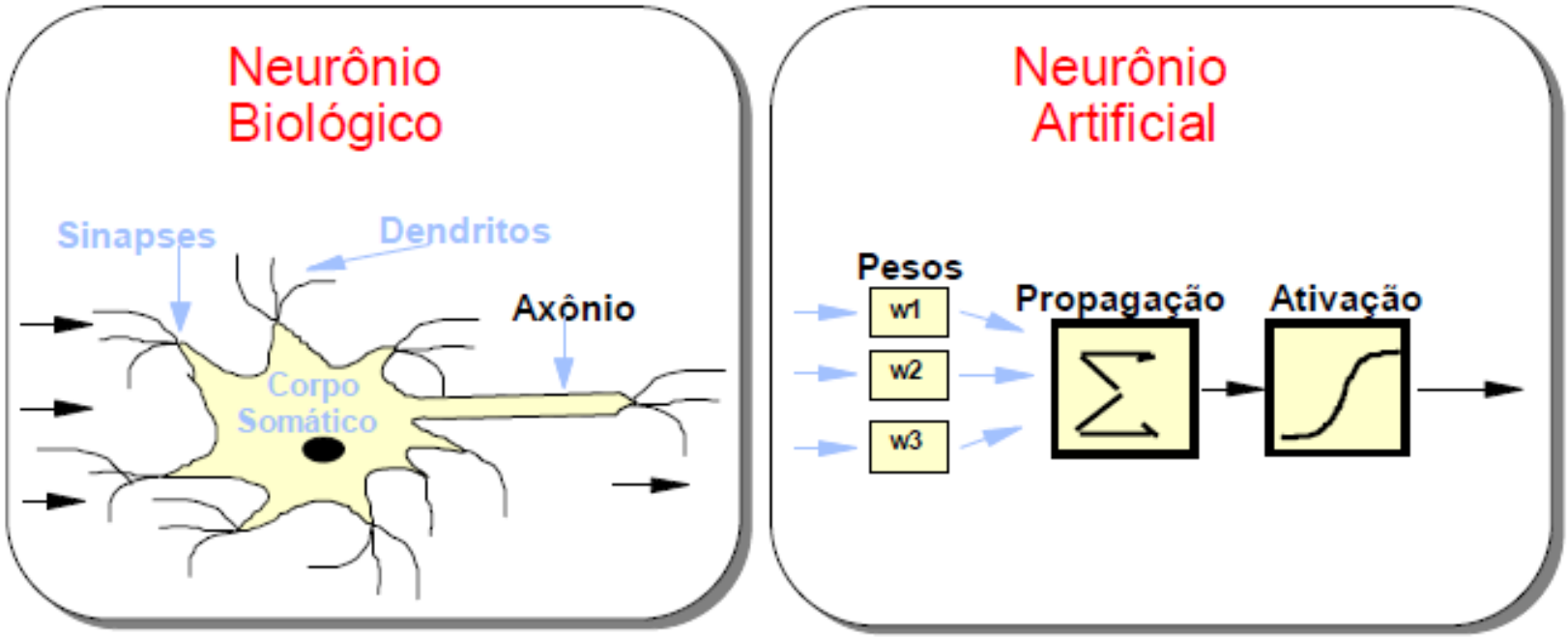
HISTÓRICO

- McCulloch & Pitts (1943):
  - modelo computacional para o neurônio artificial. Não possuía capacidade de aprendizado
- Hebb (1949):
  - modelo de aprendizado (***Hebbian Learning Rule)***
- Rosenblatt (1957):
  - Perceptron, com grande sucesso em certas aplicações e problemas em outras aparentemente similares
- Minsly & Papert (Perceptrons 1969):
  - prova matemática de que as redes Perceptron são incapazes de solucionar problemas simples tipo OU-EXCLUSIVO
- Rumelhart (início da década de 80):
  - novos modelos que superaram os problemas dos Perceptrons.

CONCEITOS BÁSICOS

- ***Neurônio Artificial***
  - (Elemento Processador)
- ***Estruturas de Interconexão***
  - FeedForward de 1 camada
  - FeedForward de Múltiplas Camadas
  - Recorrente (com realimentação)

Elemento Processador

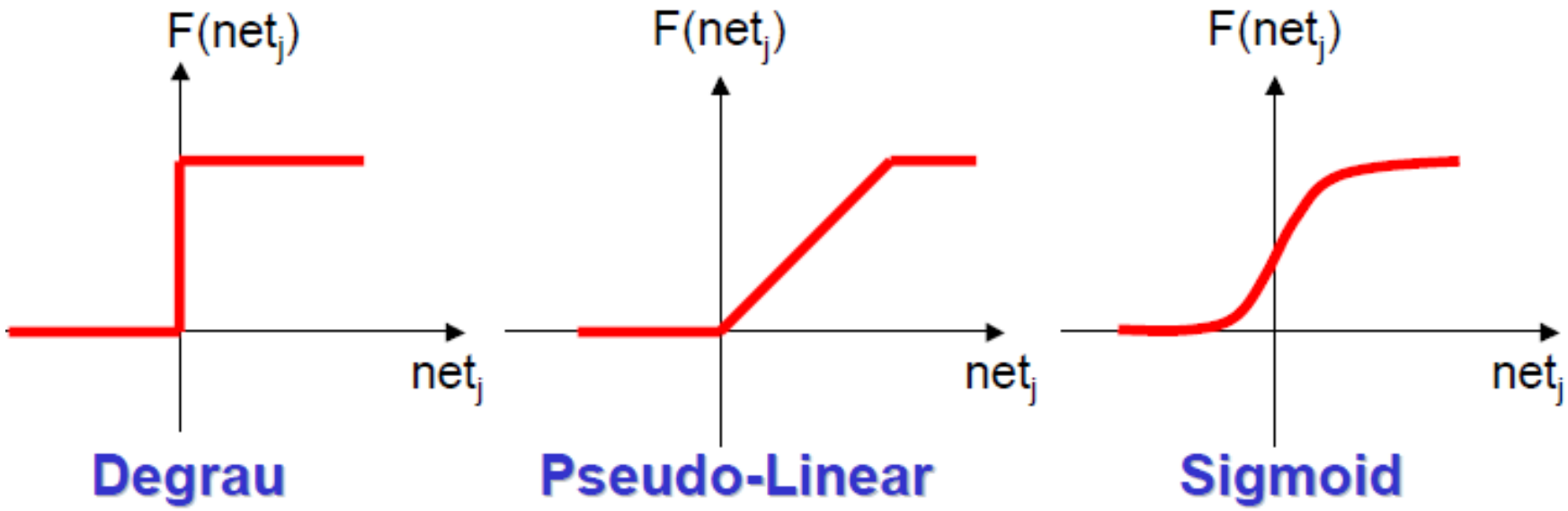


Elementos Básicos

- ***Estado de Ativação*** ➔ s_j
- ***Conexões entre Processadores***
  - a cada conexão existe um peso sináptico que determina o efeito da entrada sobre o processador ➔ w_{ij}
- ***Função de Ativação***
  - determina o novo valor do Estado de Ativação do processador ➔ $s_j = F(net_j)$

Funções de Ativação

É a função que determina o nível de ativação do Neurônio Artificial - $s_j = F(net_j)$



Tipos de Processadores

Input → Recebe os dados de entrada

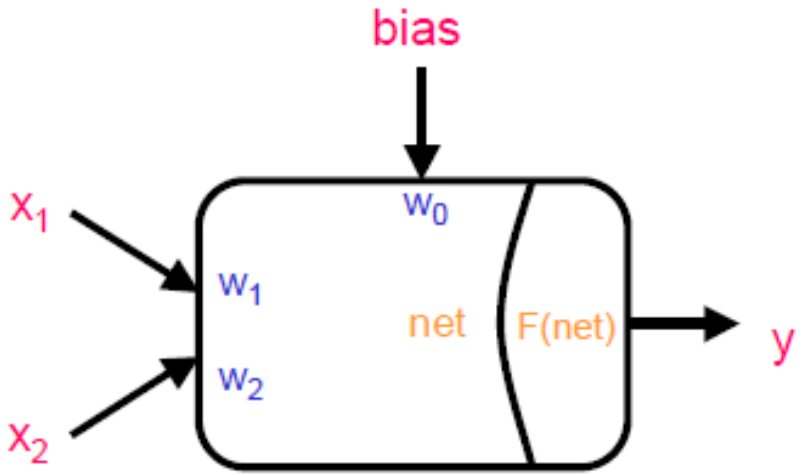
Output → Apresenta os dados de saída

Hidden → As suas entradas e saídas permanecem dentro do sistema

Neurônio Artificial

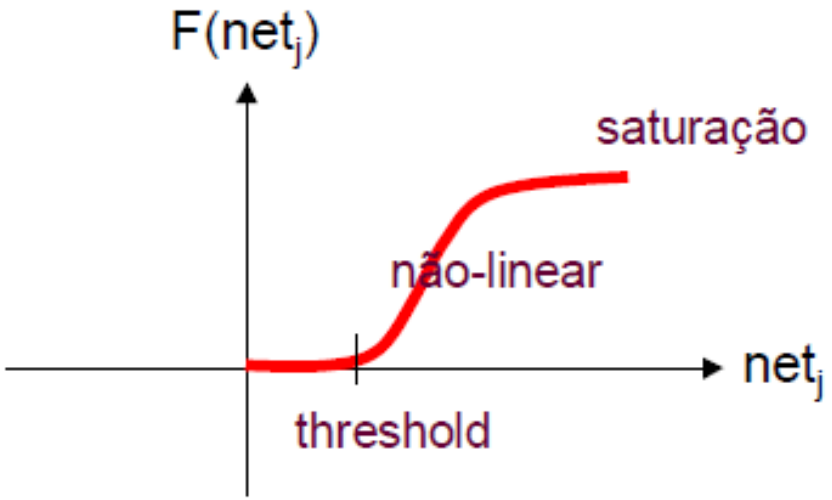
3 pontos importantes:

- ***Thresholding***
- ***Não-Linearidade***
- ***Saturação***



$$net = w_0 + w_1x_1 + w_2x_2$$

$$F(net) = \frac{1}{1 + e^{-net}} \quad \text{(sigmoid)}$$

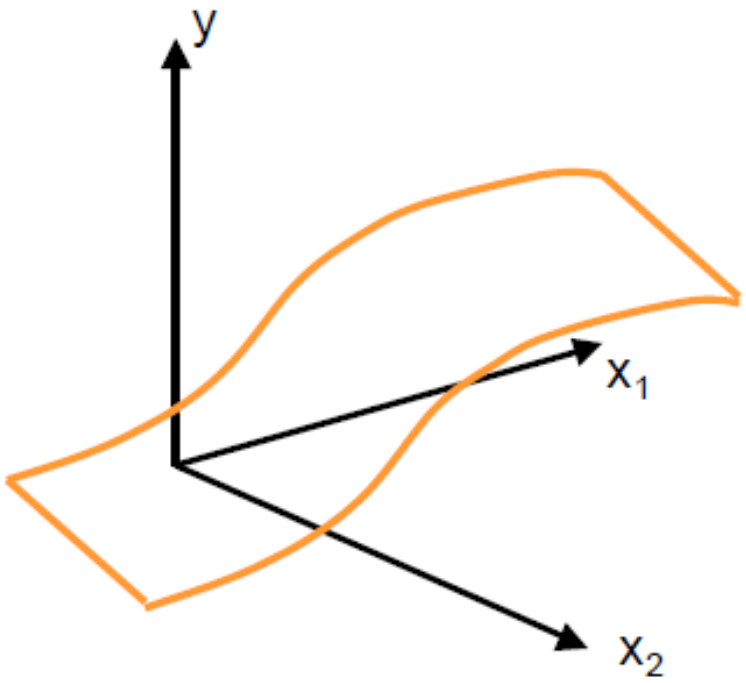


Neurônio Artificial

Em função das equações de net e F(net):

$$y = F(net) = \frac{1}{1 + e^{-(w_0 + x_1 w_1 + x_2 w_2)}}$$

Fórmula matemática representada pelo neurônio artificial



Exemplos

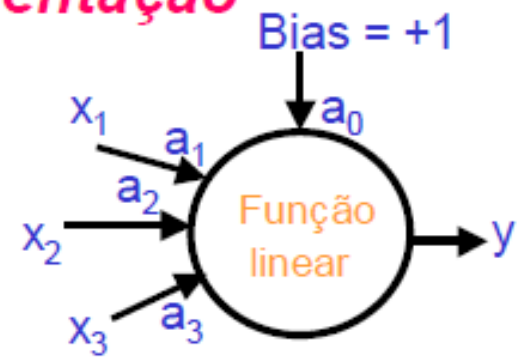
Regressão Linear:

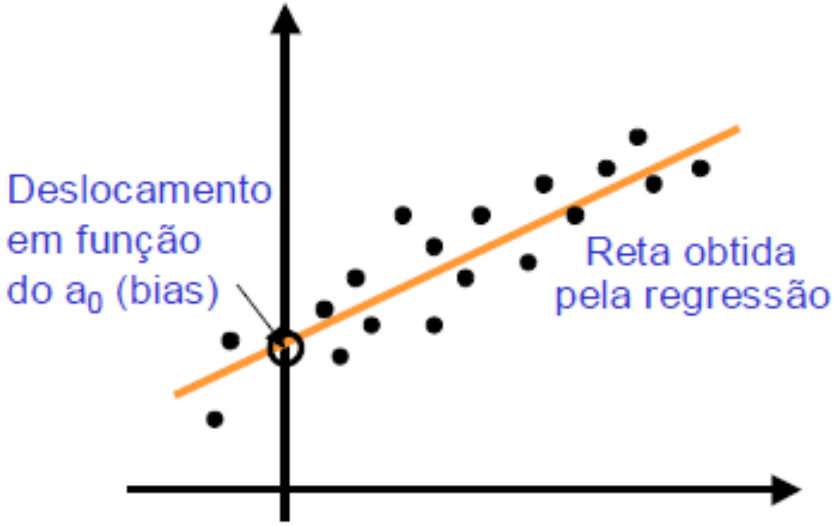
$$y = a_0 + a_1x_1 + a_2x_2 + a_3x_3$$

Variáveis explicativas

Acha a **reta** com **erro mínimo** que passe pelos **pontos existentes** (padrões de treinamento)

Representação Neural



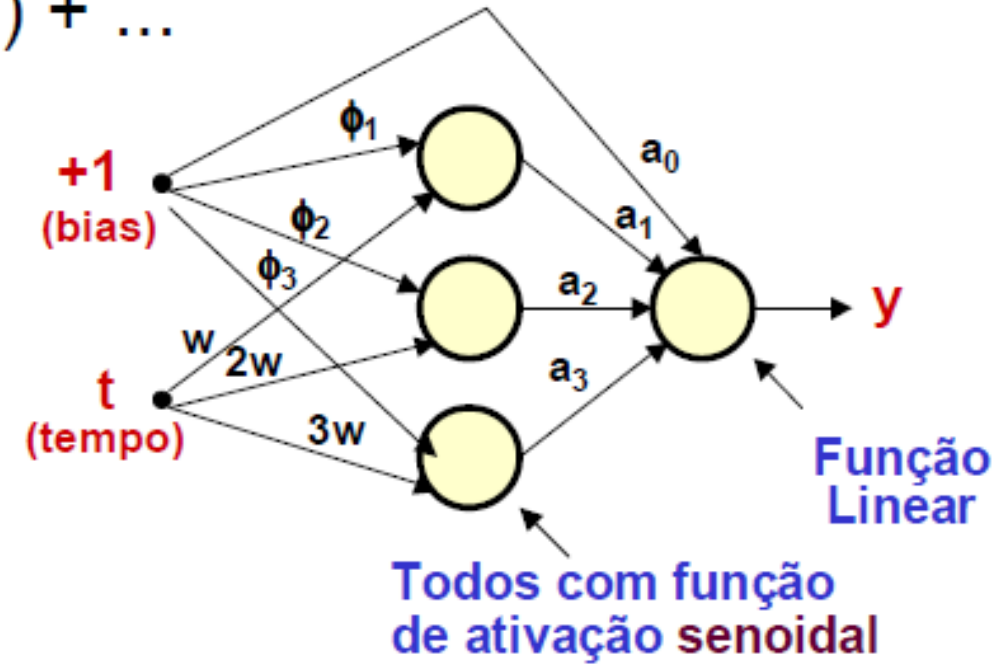


Exemplos

Transformada de Fourier:

$$y = a_0 + a_1 sen(wt + \phi_1) + a_2 sen(2wt + \phi_2) + a_3 sen(3wt + \phi_3) + \ldots$$

Representação Neural



CONCEITOS BÁSICOS

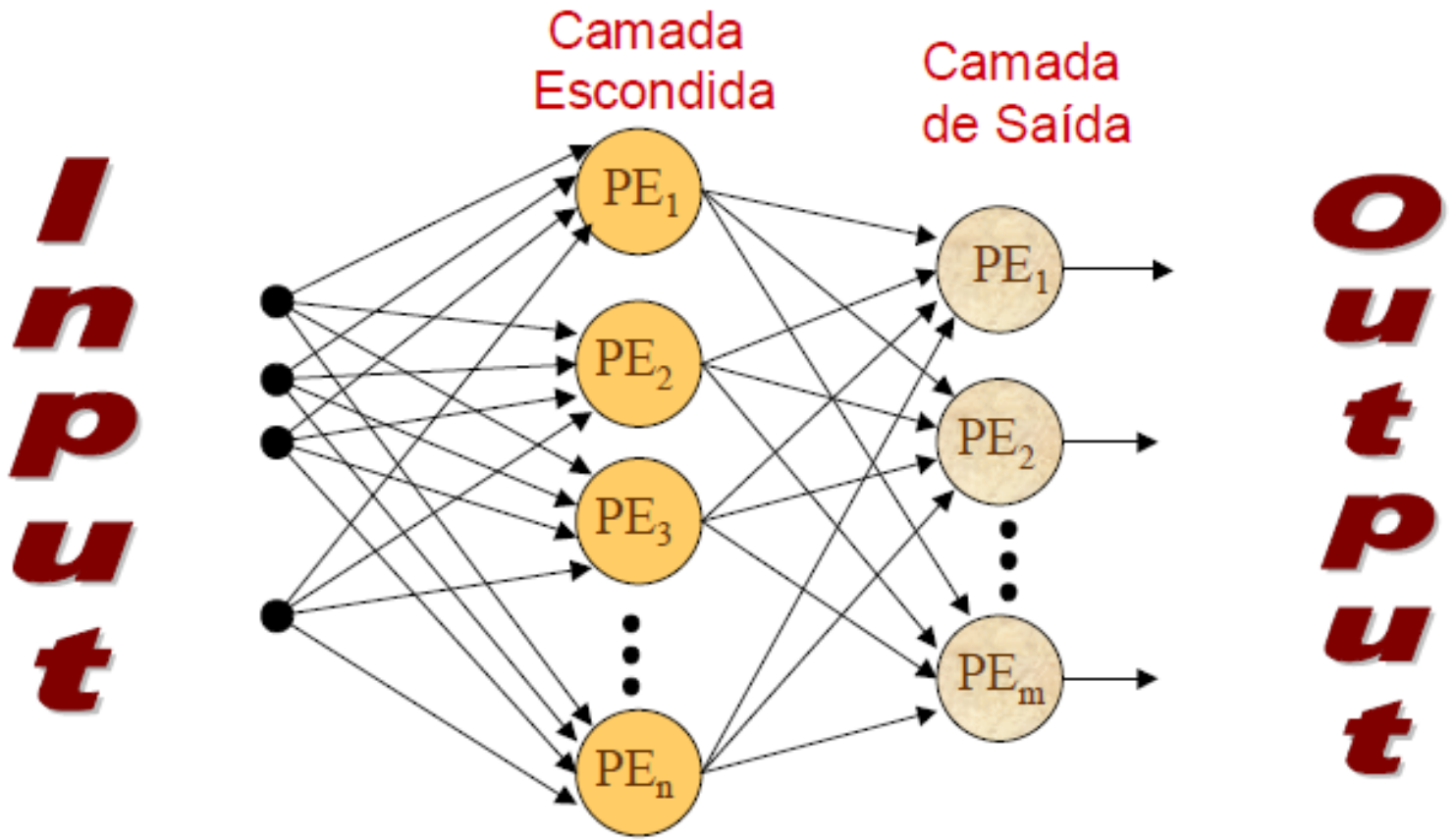
- **Neurônio Artificial**
  - (Elemento Processador)
- **Estruturas de Interconexão**
  - FeedForward de 1 camada
  - FeedForward de Múltiplas Camadas
  - Recorrente (com realimentação)

Topologias das Redes

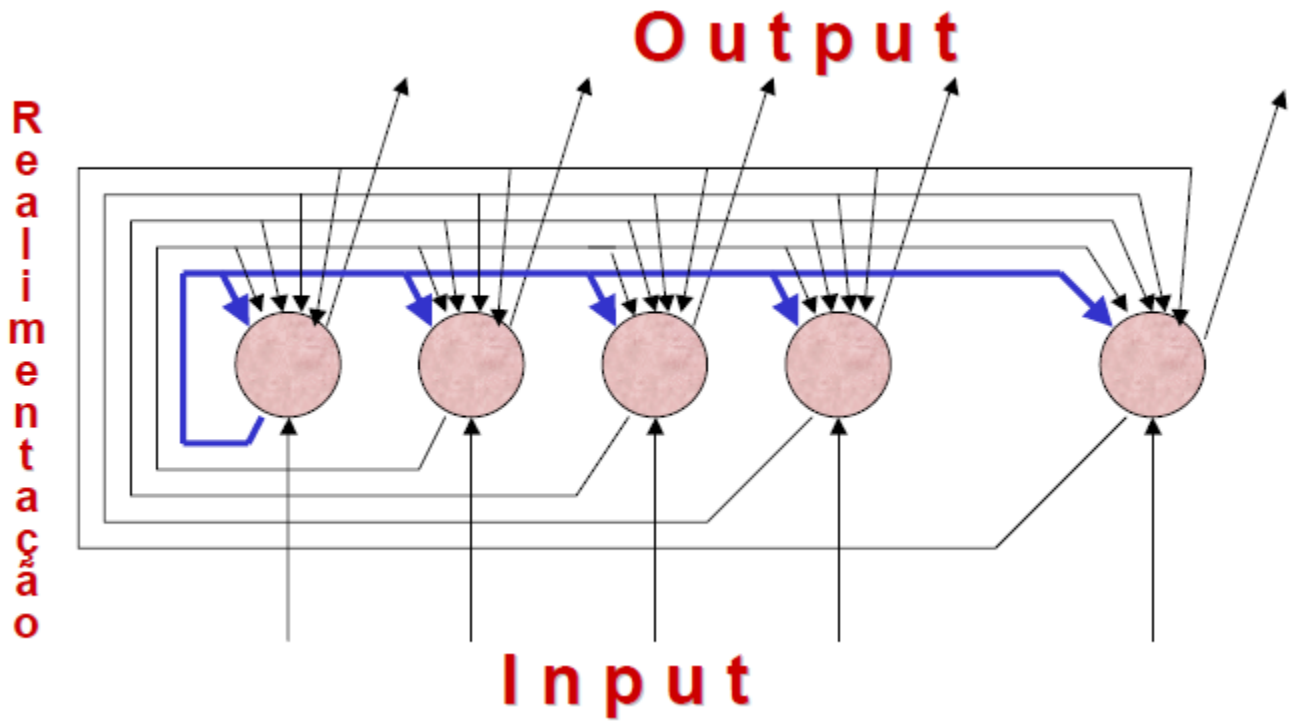
- **Neurais**
- ***Redes Feed-Forward:***
  - redes de uma ou mais camadas de processadores, cujo ***fluxo de dados é*** sempre em ***uma única direção, isto é, não*** existe realimentação.
- ***Redes Recorrentes:***
  - redes com conexões entre processadores da mesma camada e/ou com processadores das camadas anteriores (***realimentação).***

Redes Feed-Forward

Rede de Múltiplas Camadas



Redes Recorrentes



Processamento Neural

O processamento de uma Rede Neural pode ser dividido em duas fases:

Recall → Processo de cálculo da saída da rede, dado um certo padrão de entrada - *Recuperação da Informação*

Learning → Processo de atualização dos pesos sinápticos para a aquisição do conhecimento - *Aquisição da Informação*

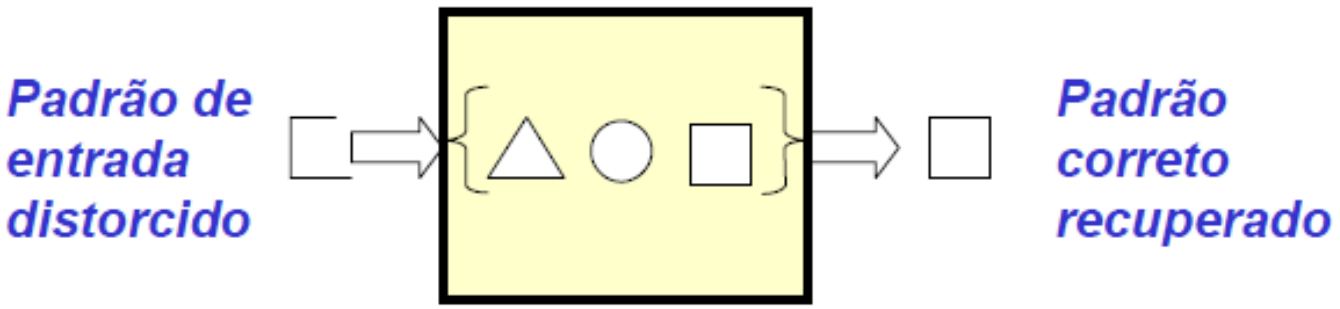
Recuperação de Dados

- **Assumindo que um conjunto de padrões tenha sido *armazenado,* a Rede Neural pode executar as seguintes tarefas:**

Recuperação de Dados

Autoassociação:

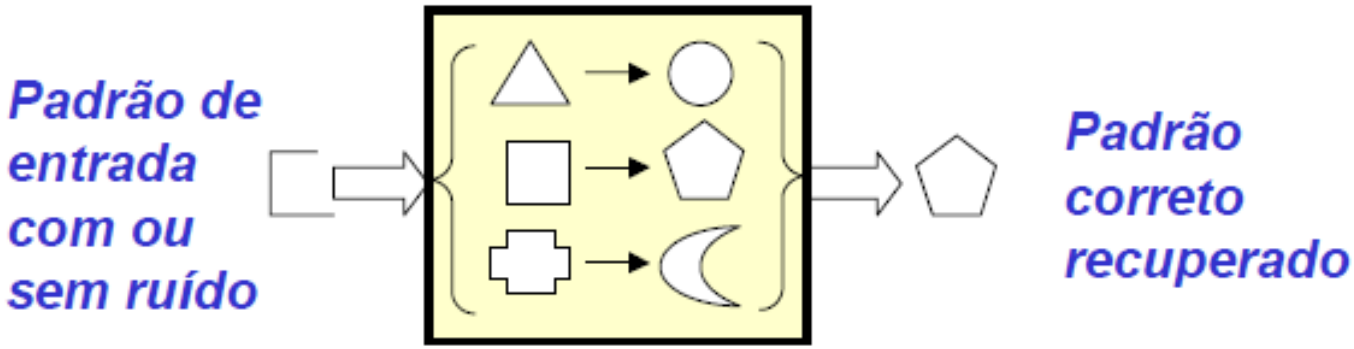
- A Rede Neural recupera o padrão armazenado mais semelhante ao padrão de entrada apresentado. ➡ ***Recuperação de padrões ruidosos***



Recuperação de Dados

Heteroassociação:

- A Rede Neural armazena a associação entre um par de padrões. ➡ ***Recuperação de um padrão diferente do da entrada.***

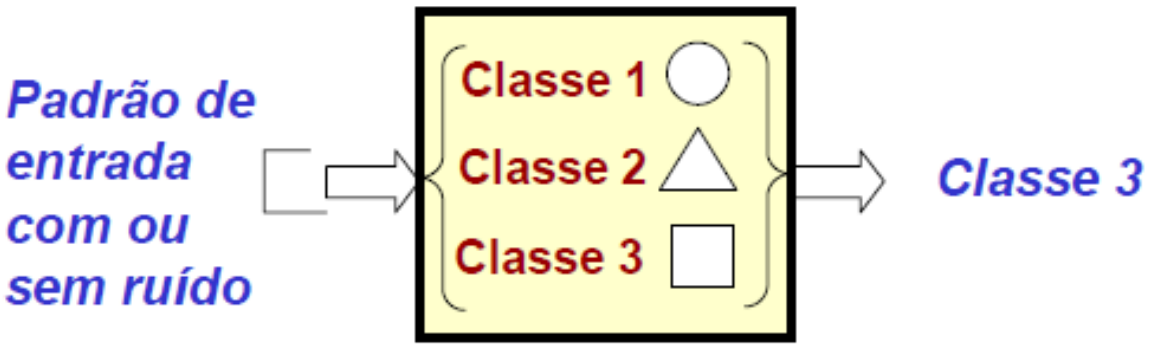


Recuperação de Dados

Classificação:

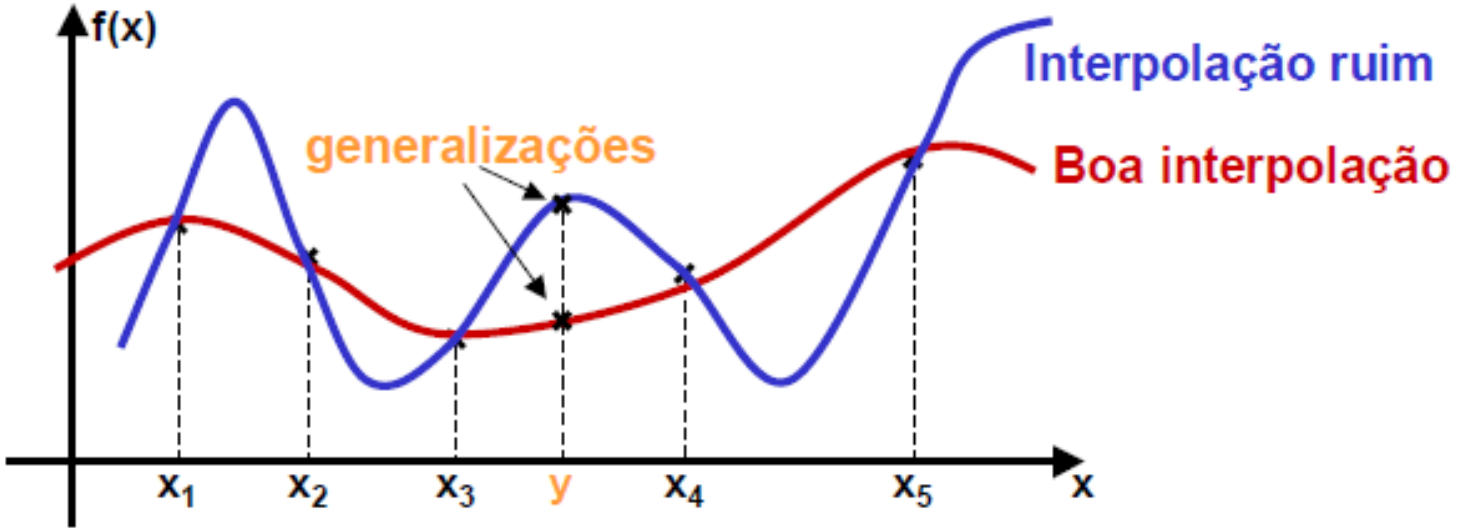
- A Rede Neural responde com a informação relativa à ***classe*** a qual o padrão de entrada pertence. ➡ ***Caso especial de Heteroassociação***

Ex: Padrões de entrada divididos em 3 classes distintas.



GENERALIZAÇÃO

- A Rede Neural responde corretamente a um padrão de entrada fora do conjunto de treinamento ➡ ***Interpola corretamente os novos pontos apresentados***



x_i ➔ pontos do conjunto de treinamento
y ➔ novo ponto para generalização

Aprendizado

- Processo pelo qual os parâmetros livres - *pesos sinápticos* - de uma rede neural são adaptados através de um processo contínuo de estimulação pelo ambiente.

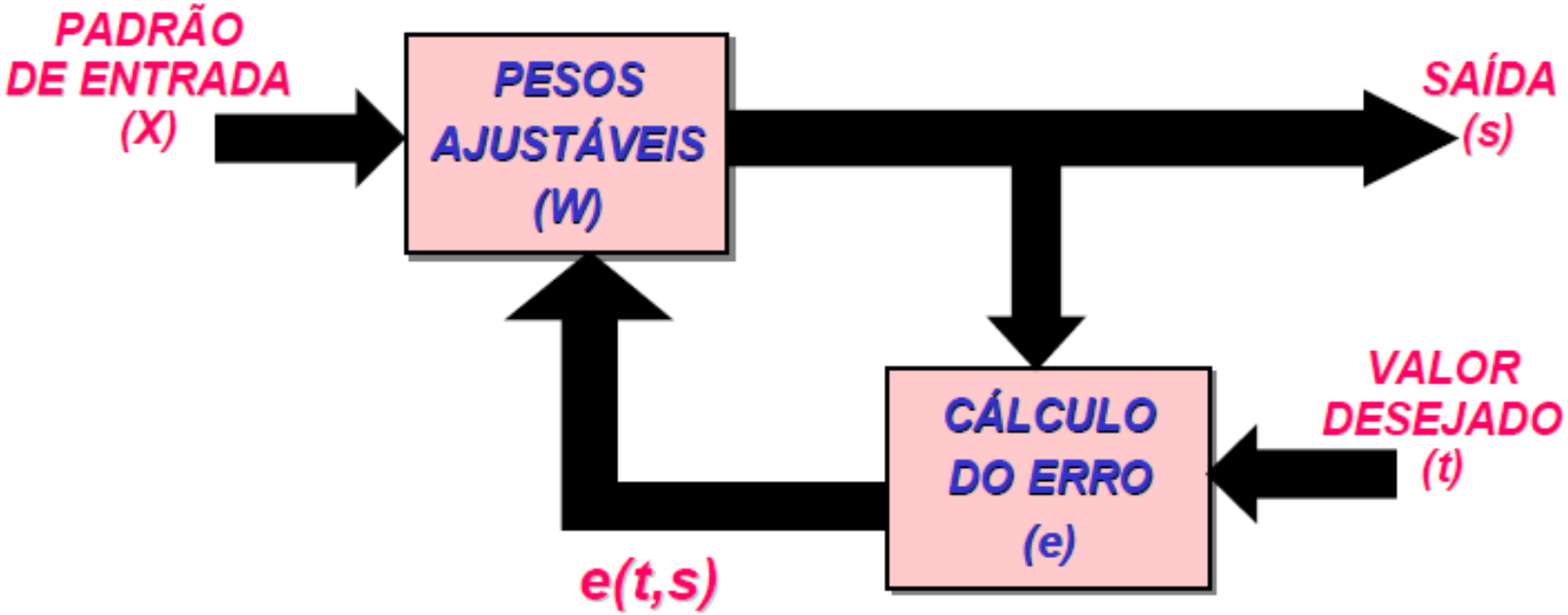
- Existem 3 tipos básicos de aprendizado:
  - ☑ Treinamento Supervisionado;
  - ☑ Treinamento Não-Supervisionado;
  - ☑ Treinamento em *"Batch"*.

Treinamento Supervisionado

A rede é treinada através do fornecimento dos valores de entrada e seus respectivos valores da saída desejada *("training pair")*.

Geralmente efetuado através do processo de minimização do erro calculado na saída.

Treinamento Supervisionado

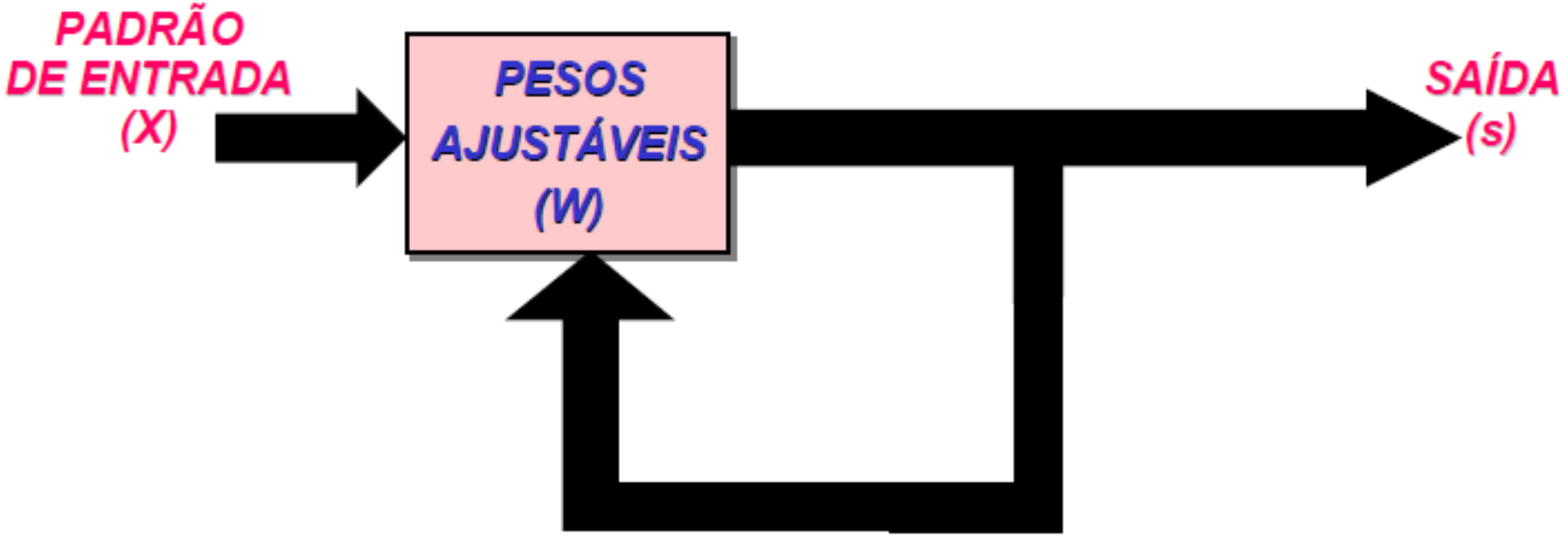


Treinamento Não-Supervisionado

"Self-Organization" → Não requer o valor desejado de saída da rede. O sistema extrai as características do conjunto de padrões, agrupando-os em classes inerentes aos dados.

Aplicado a problemas de *Clusterização*

Treinamento Não-Supervisionado



Treinamento em “Batch”

Os valores dos pesos sinápticos são estabelecidos a priori, em um único passo. ➡ Também chamado de Gravação - ***“Recording”***

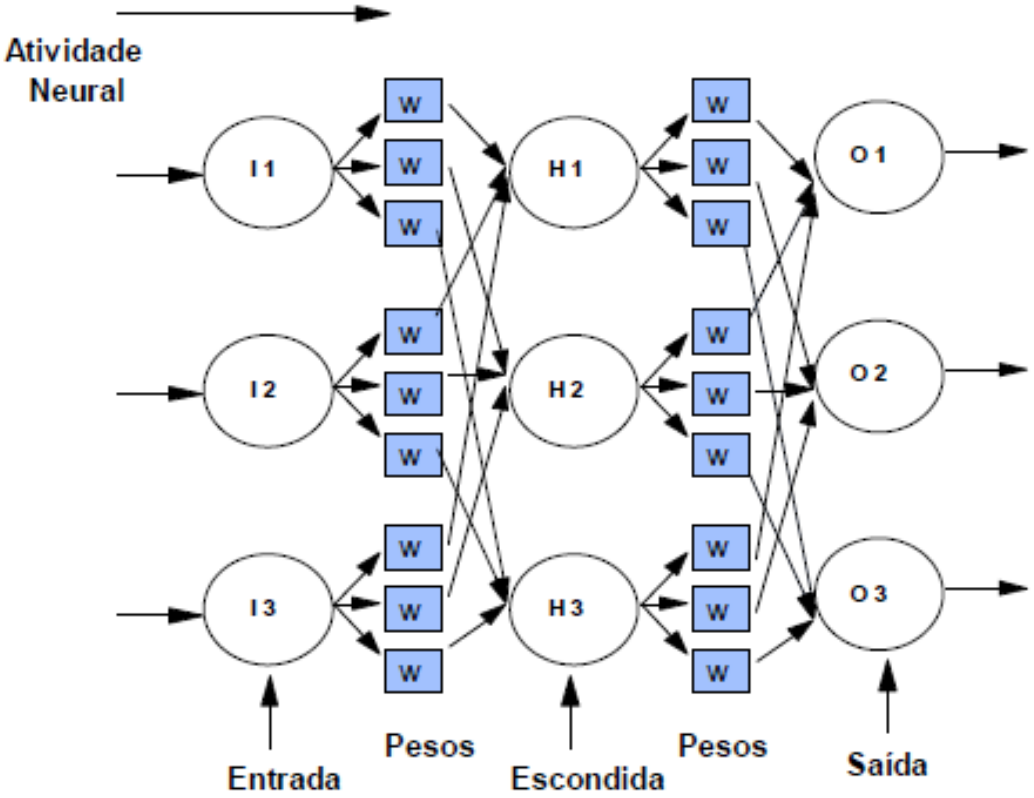
Exemplos de Treinamento Supervisionado

❶ Reconhecimento de Caracteres

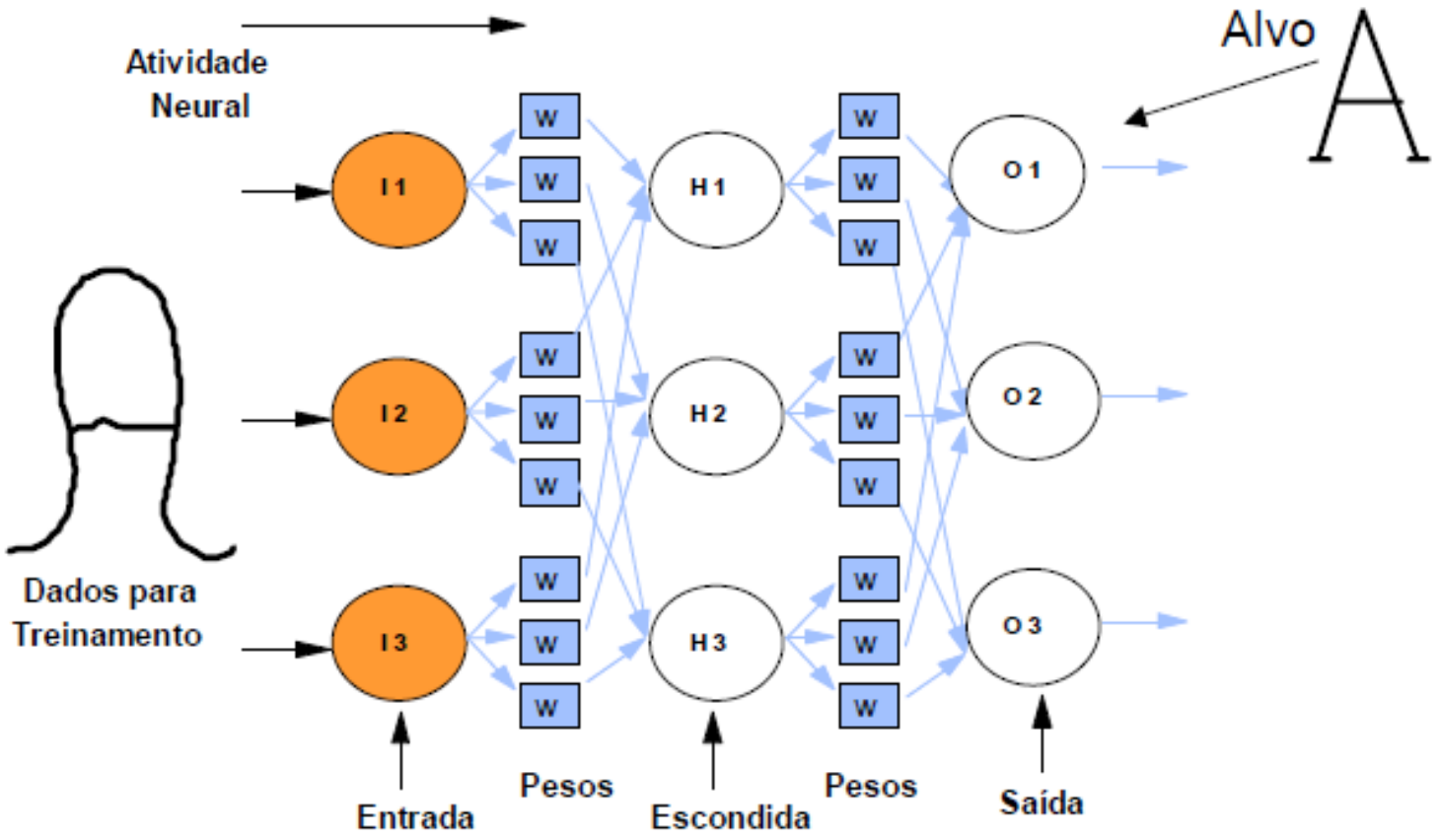
❷ Previsão de Séries Temporais

Reconhecimento de Caracteres

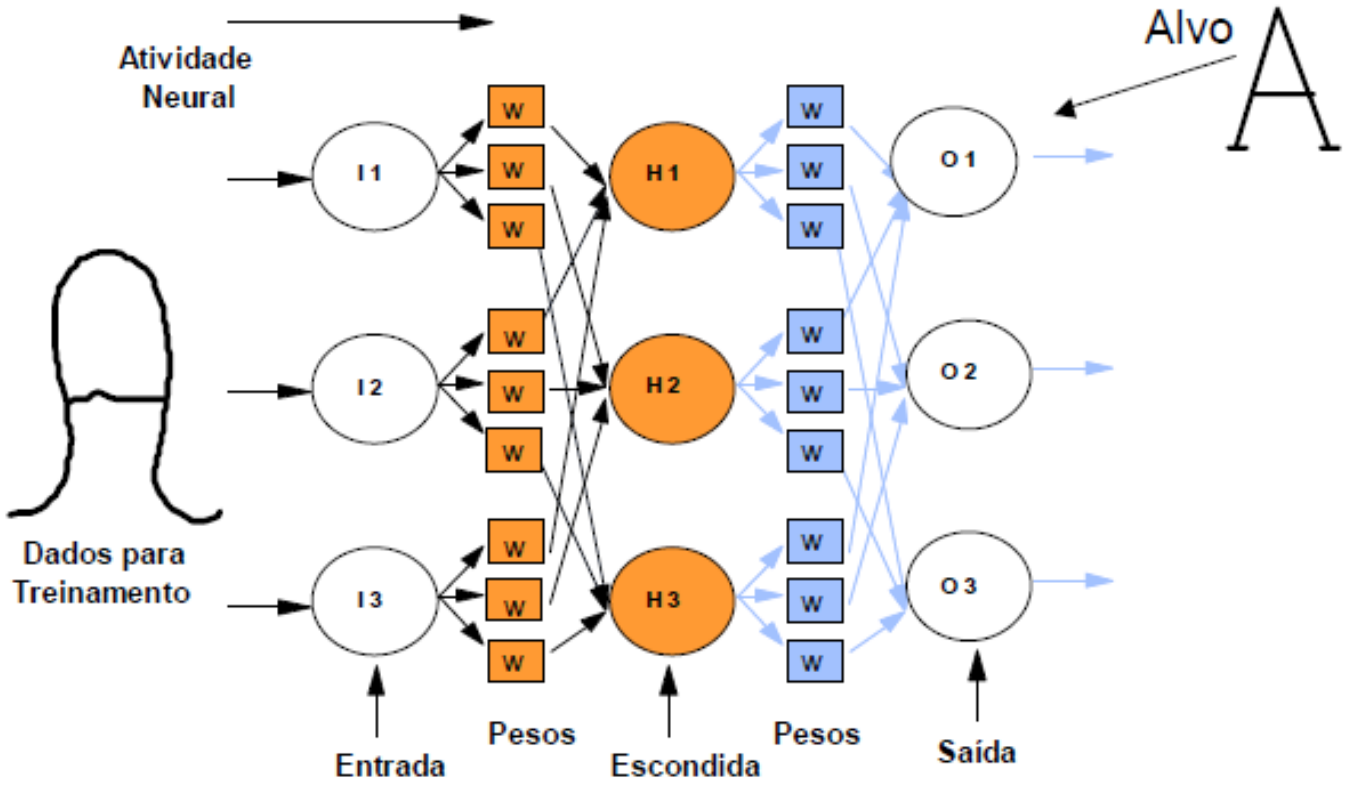
Estrutura da Rede Neural



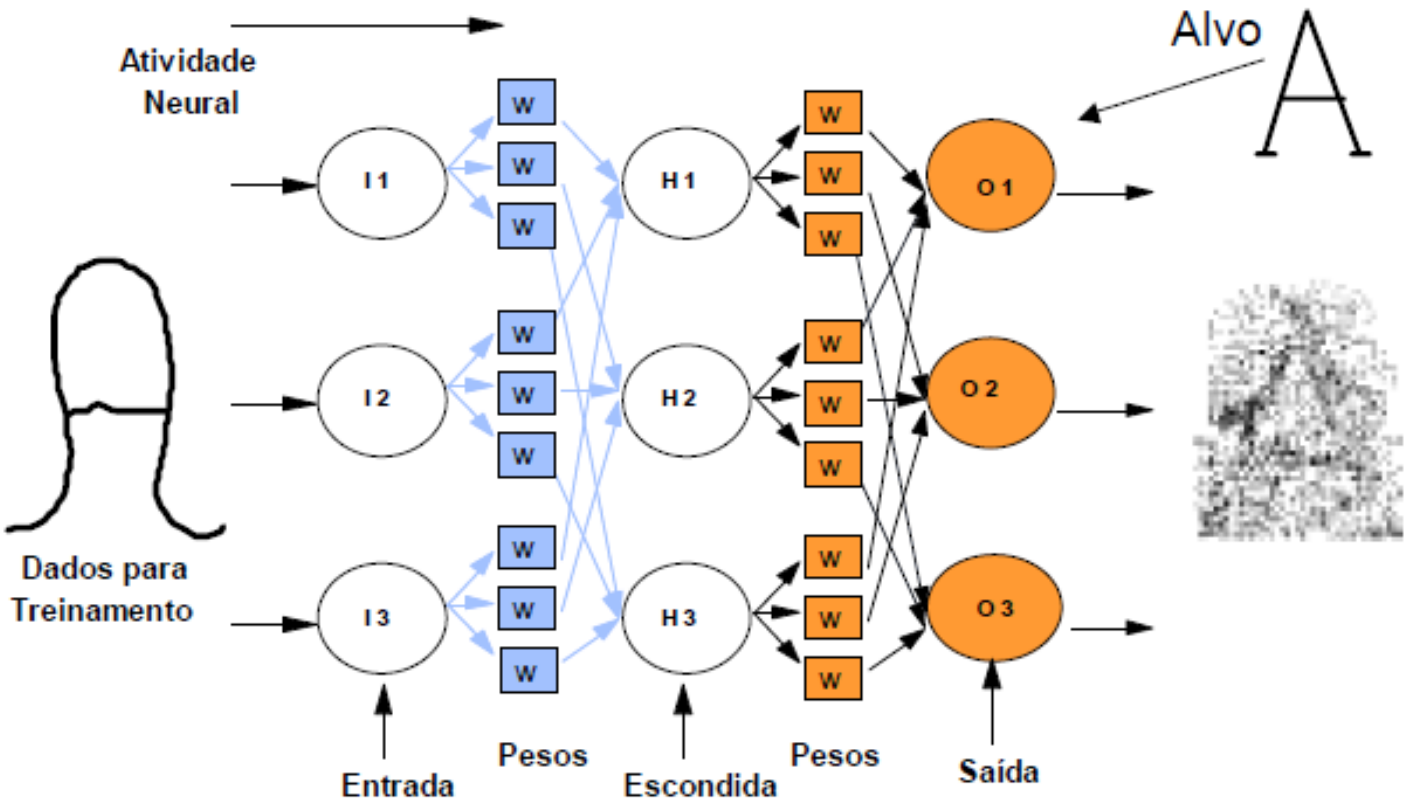
Processo de Aprendizado



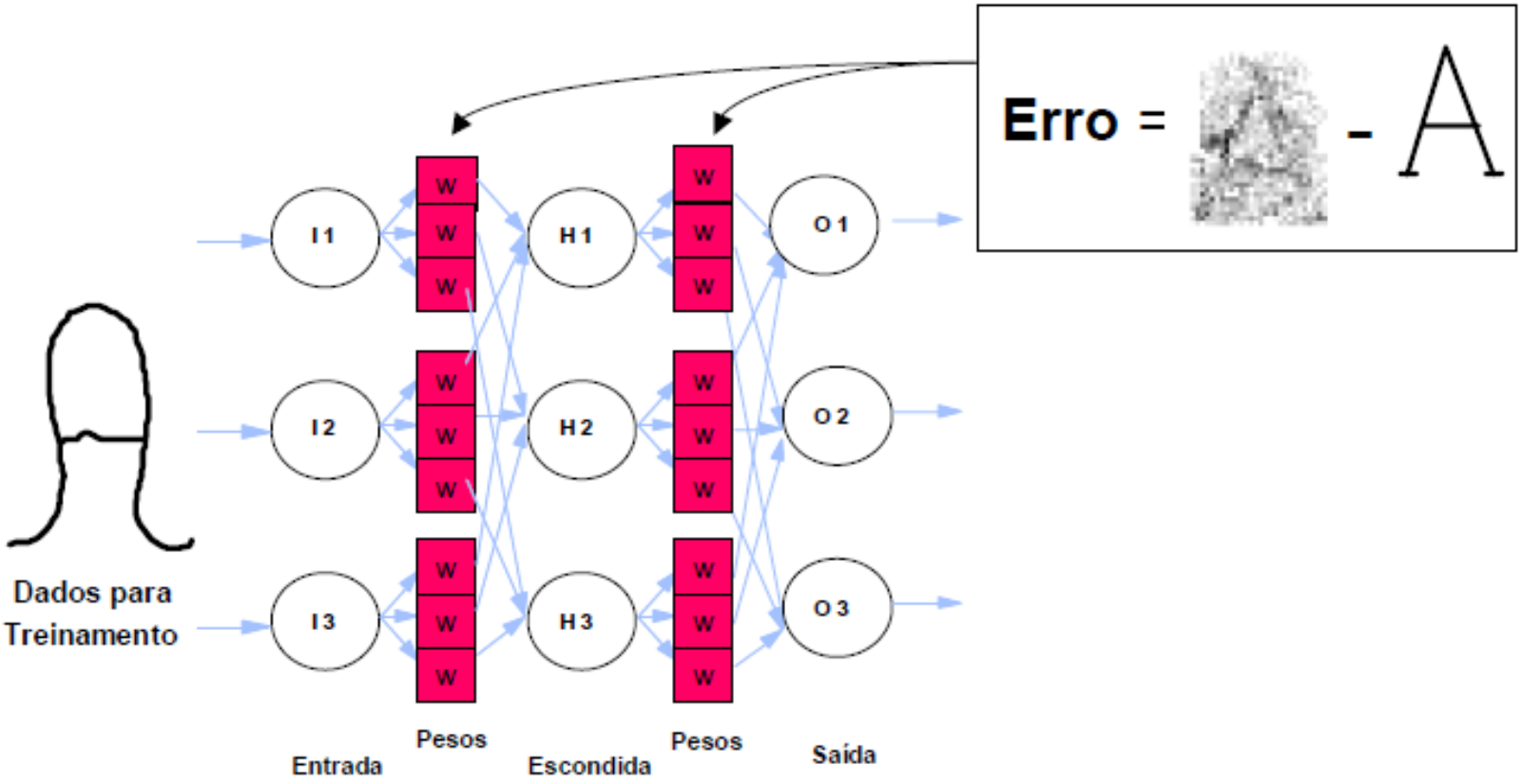
Processo de Aprendizado



Processo de Aprendizado

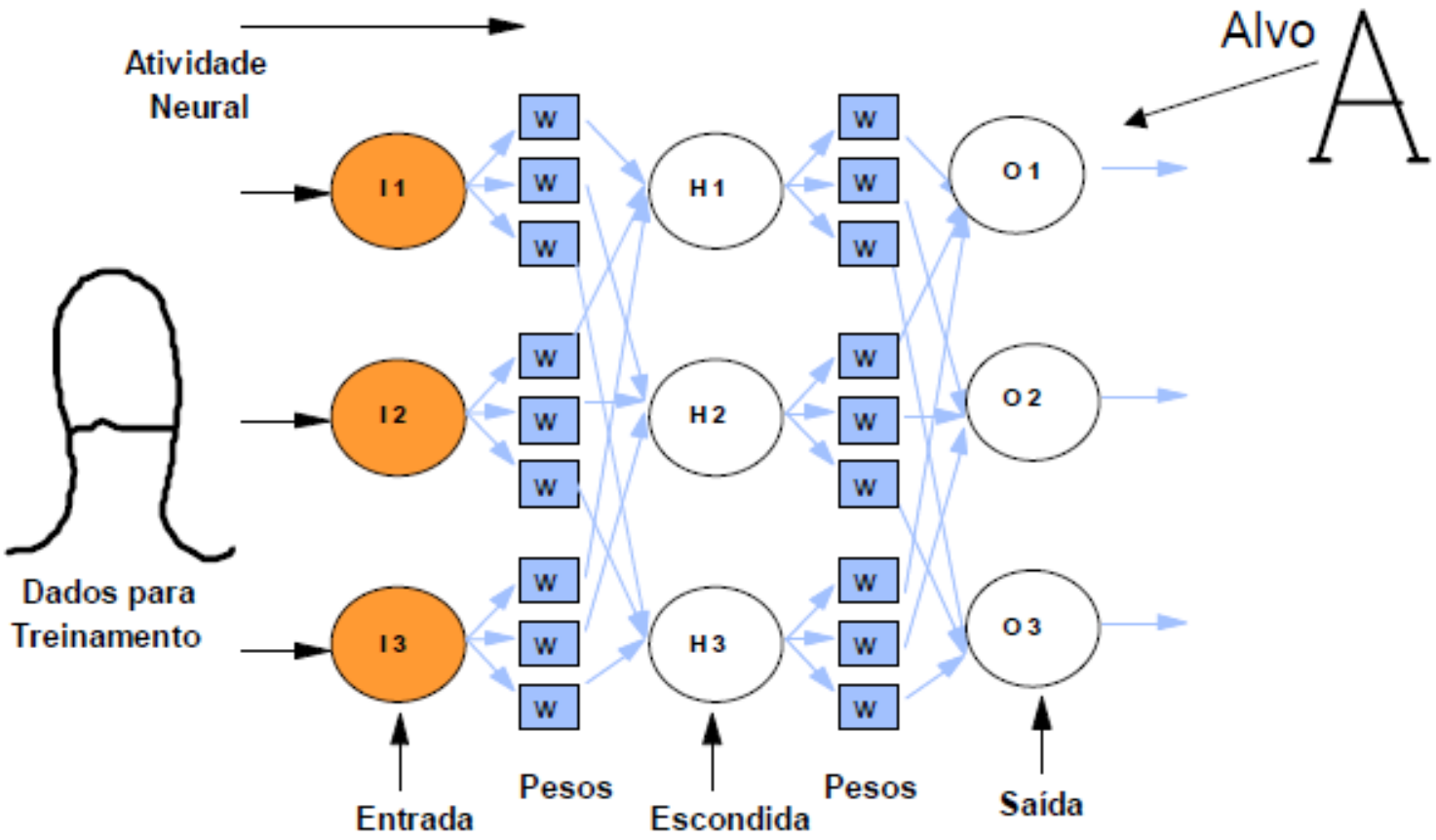


Processo de Aprendizado

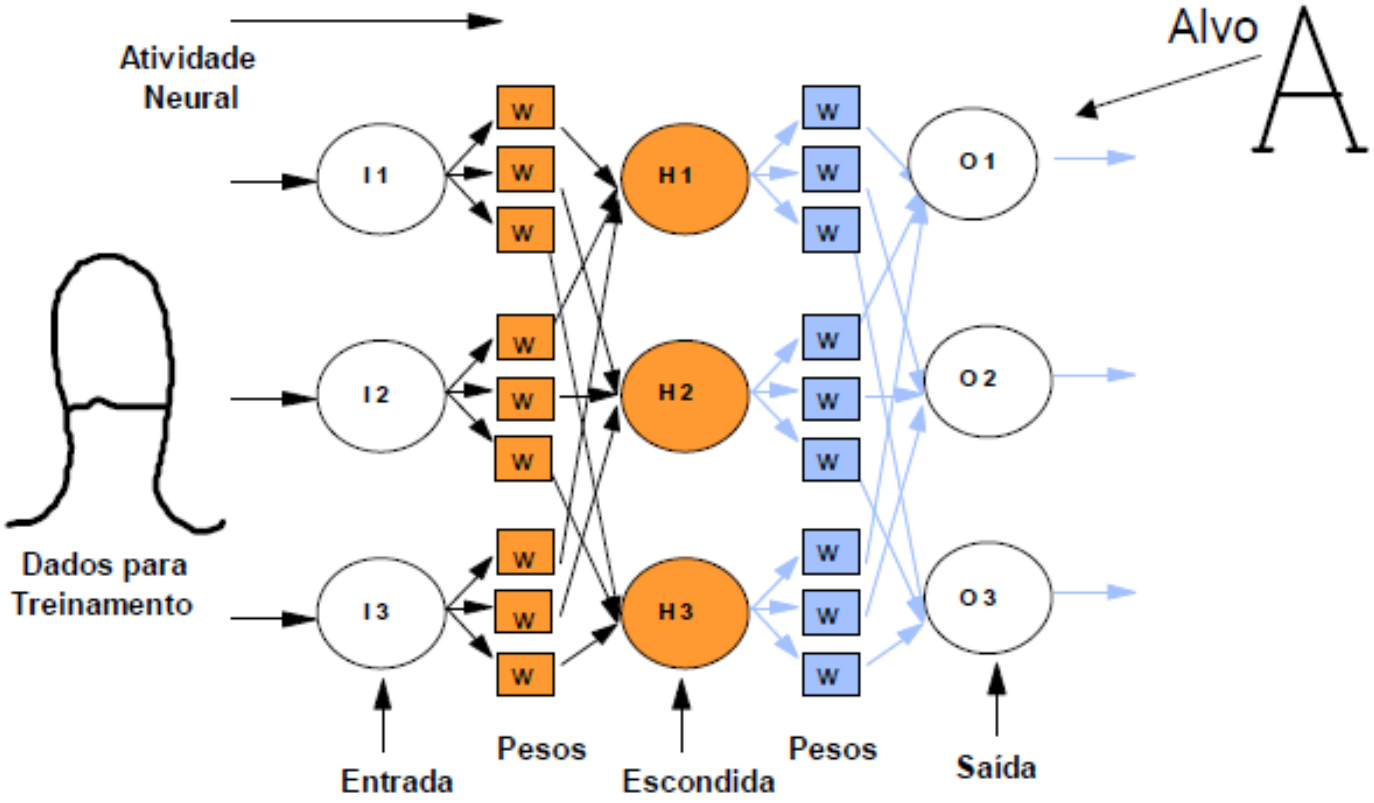


Atualização dos pesos em função do erro

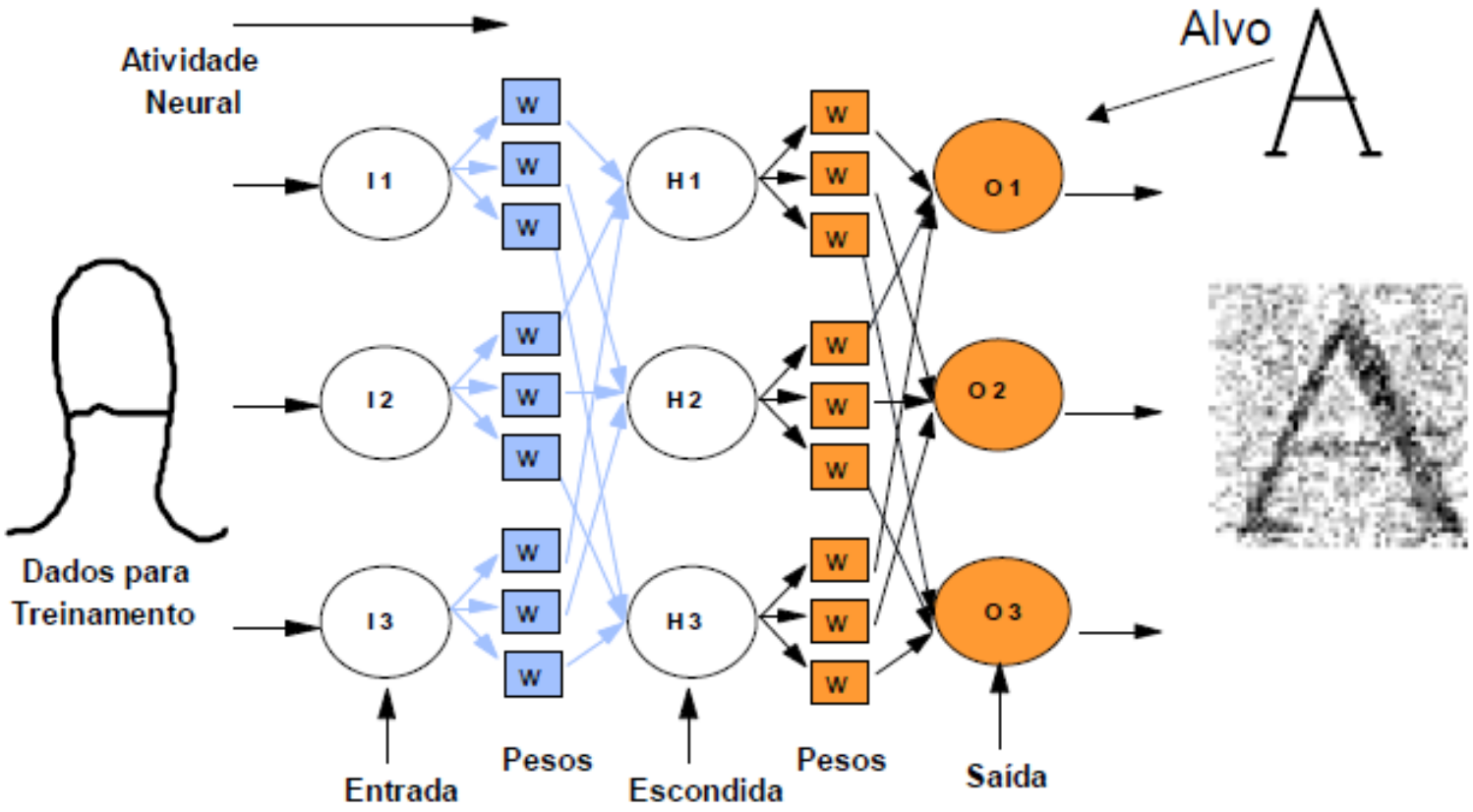
Processo de Aprendizado



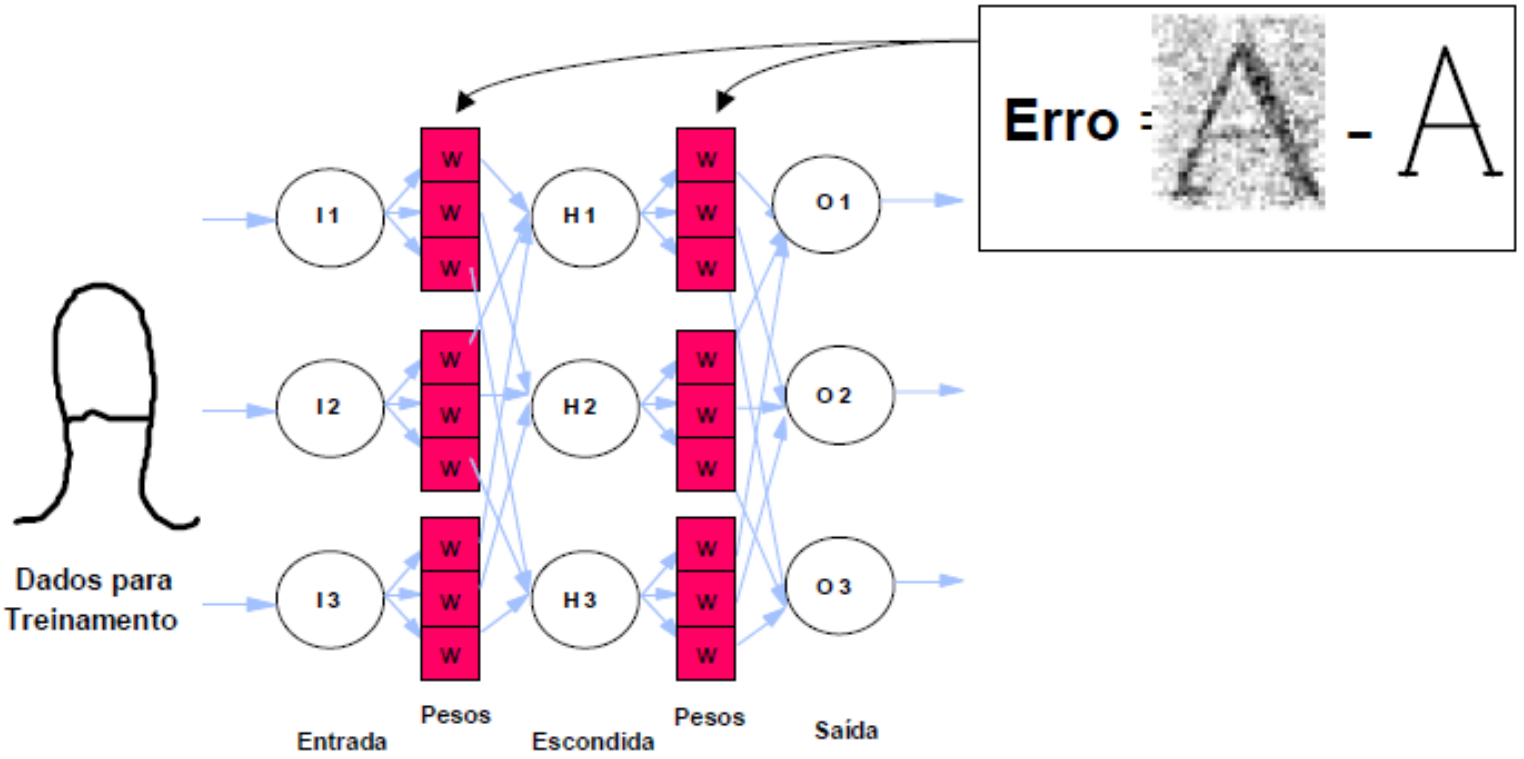
Processo de Aprendizado



Processo de Aprendizado

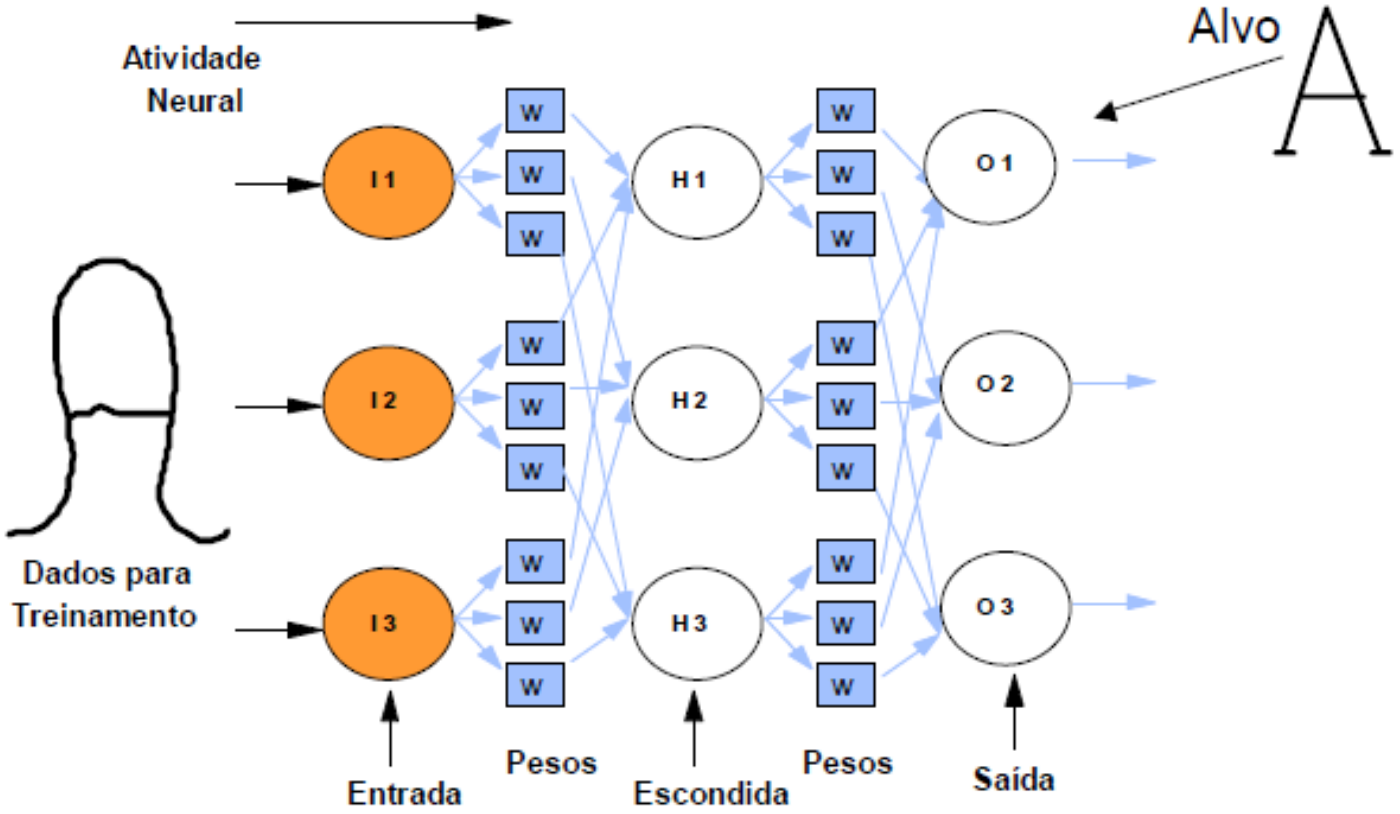


Processo de Aprendizado

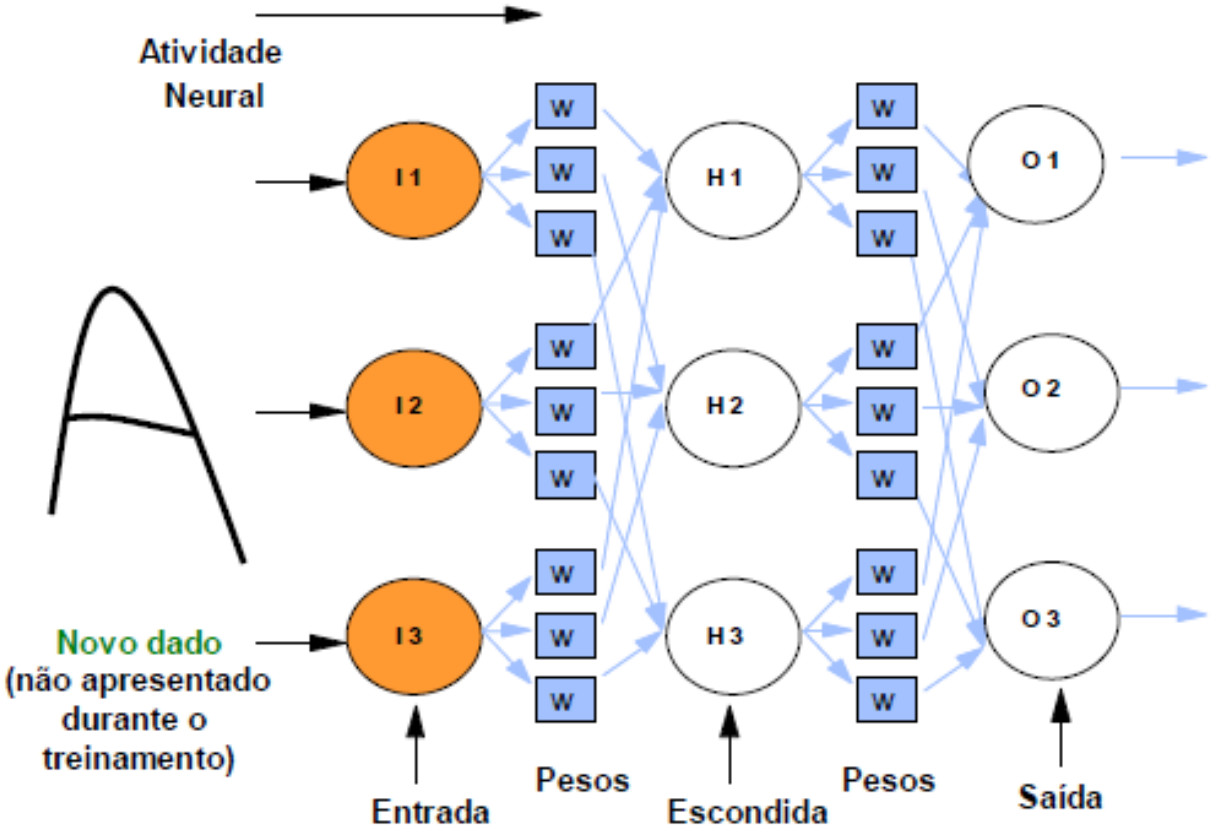


Atualização dos pesos em função do erro

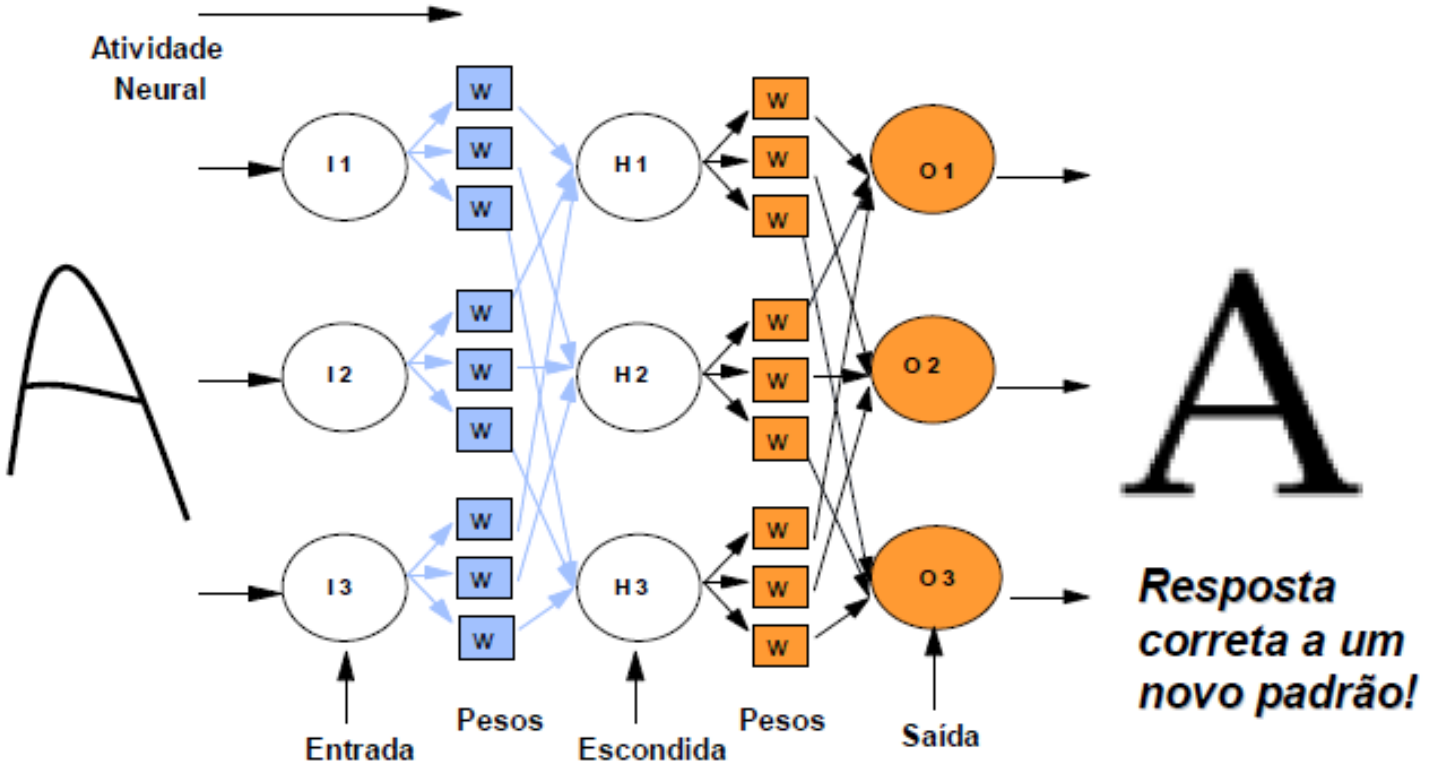
Processo de Aprendizado



Processo de Generalização



Processo de Generalização



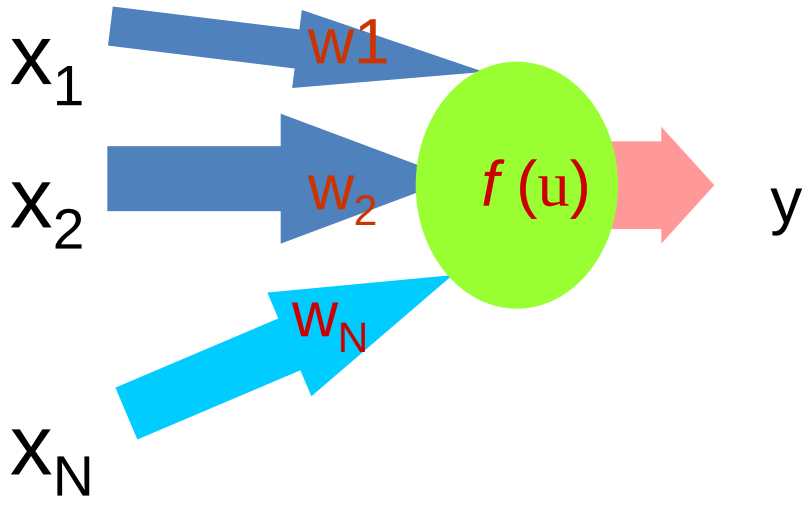
Redes Neurais Artificiais

Perceptrons e Adalines

Unidades de Processamento

- Função: receber entradas, computar função sobre entradas e enviar resultado para as unidades seguintes
- Entrada total

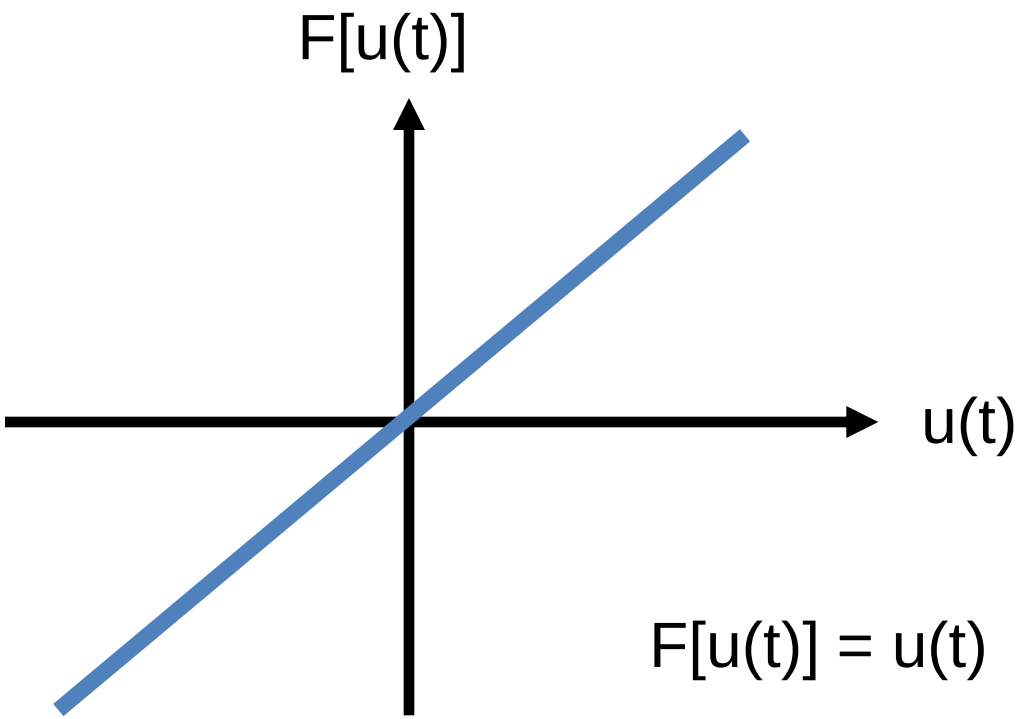
$$u = \sum_{j=1}^{N} x_j w_j$$



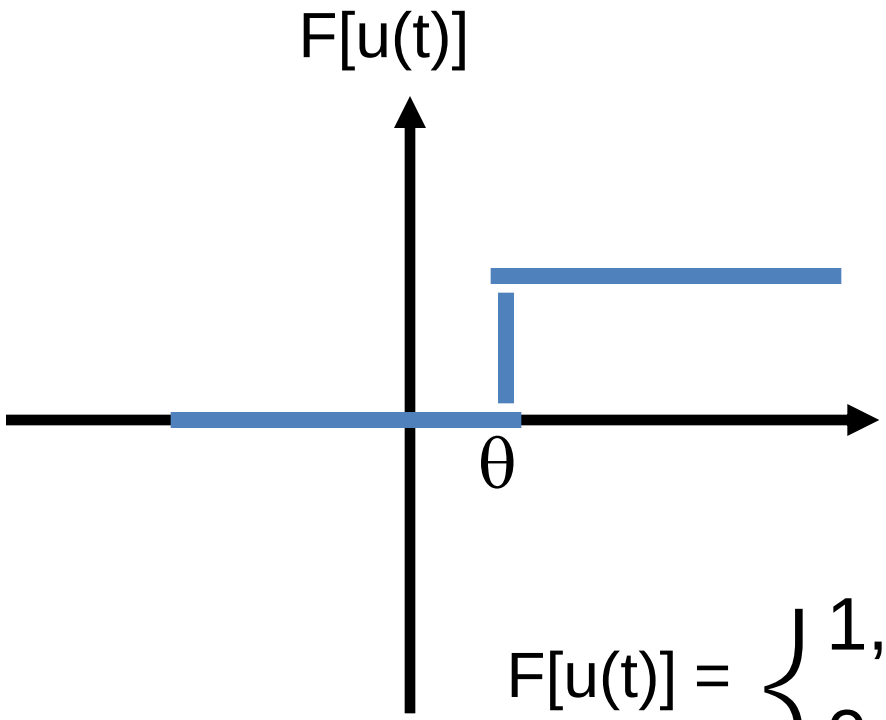
Funções de ativação

- Funções de ativação mais comuns
  - $a(t) = u(t)$ (linear)
  - $a(t) = \begin{cases} 1, \text{ se } u(t) \geq \theta \\ 0, \text{ se } u(t) < \theta \end{cases}$ (threshold ou limiar)
  - $a(t) = 1/(1 + e^{-\lambda u(t)})$ (sigmoide logística)
  - $a(t) = \dfrac{(1 - e^{-\lambda u(t)})}{(1 + e^{-\lambda u(t)})}$ (tangente hiperbólica)

Função linear

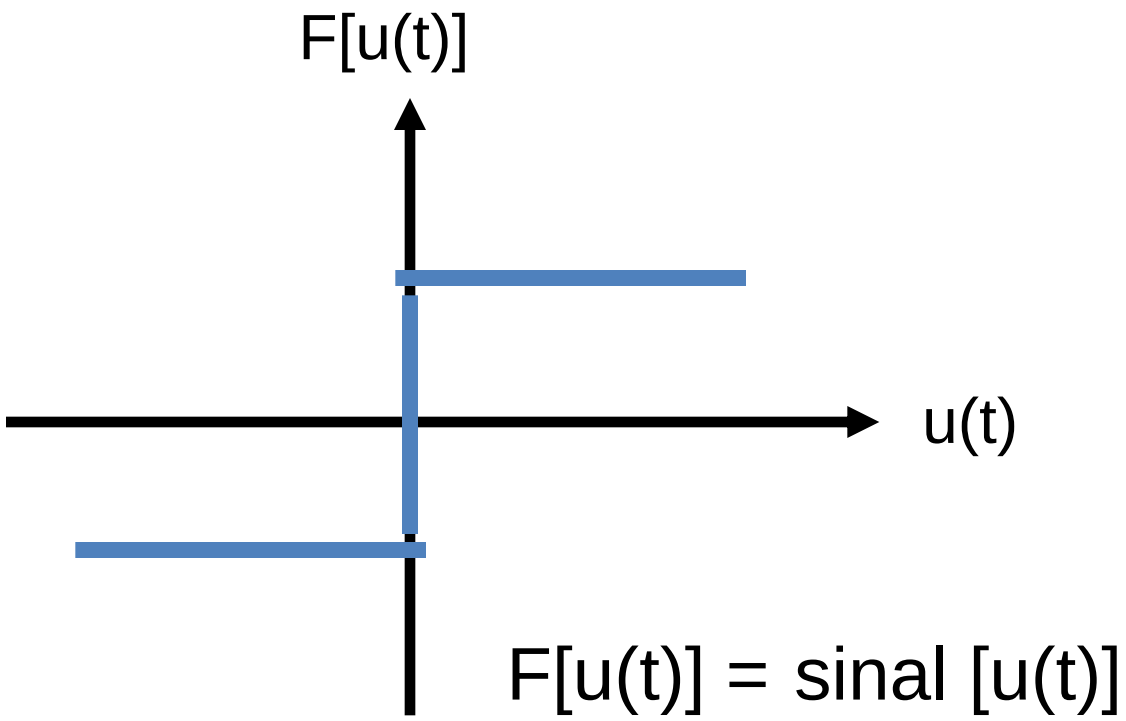


Função limiar unipolar

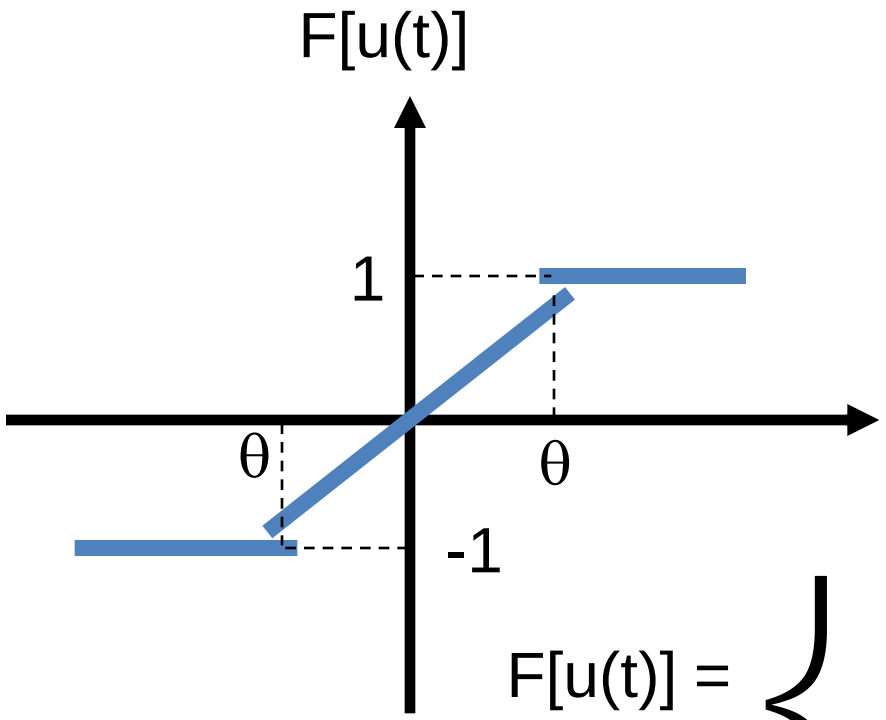


$$F[u(t)] = \begin{cases} 1, & \text{se } u(t) \geq \theta \\ 0, & \text{se } u(t) < \theta \end{cases}$$

Função limiar bipolar

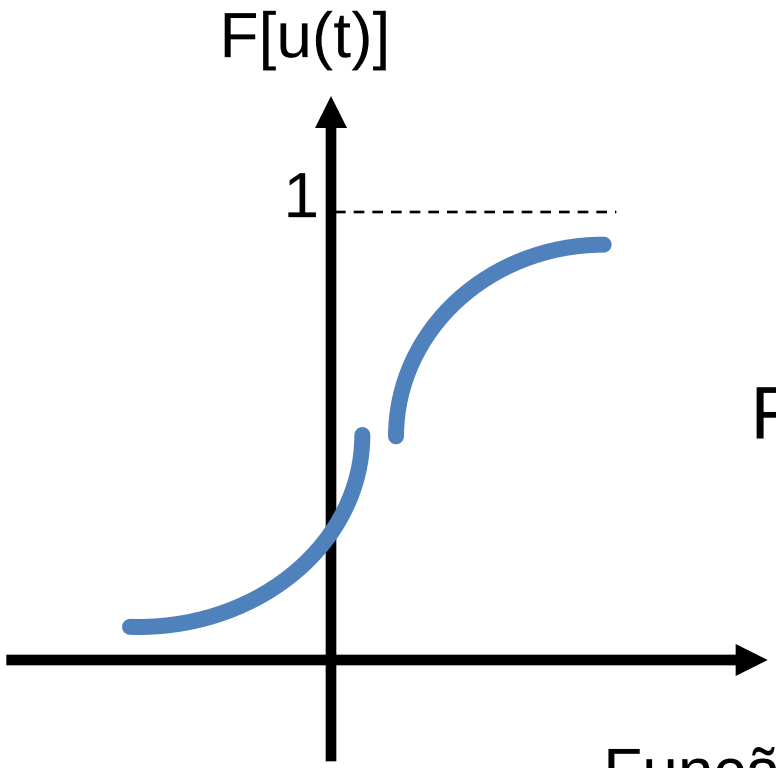


Função *piecewise* linear



$$F[u(t)] = \begin{cases} +1, & \text{se } u(t) > \theta \\ -\theta \leq u(t) \leq \theta & \\ -1, & \text{se } u(t) < -\theta \end{cases}$$

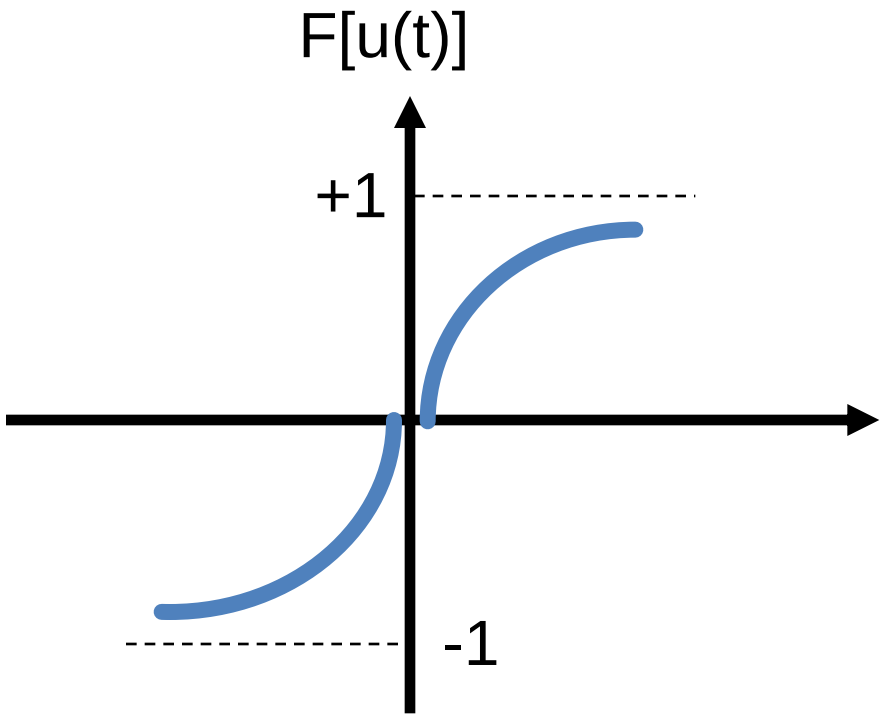
Função sigmoide logística (bipolar)



$$F[u(t)] = 1/(1 + e^{-\lambda u(t)})$$

Função de limiar diferenciável

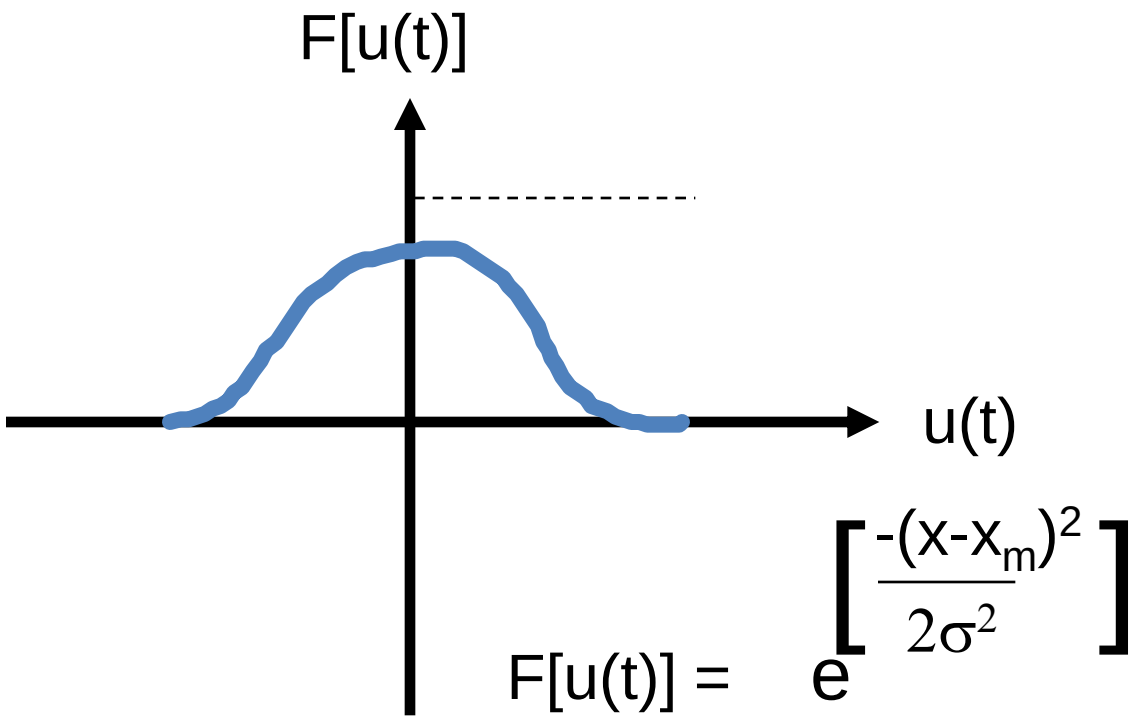
Função sigmoid bipolar



$$F[u(t)] = \frac{(1 - e^{-\lambda u(t)})}{(1 + e^{-\lambda u(t)})}$$

Função de limiar diferenciável

Função Gaussiana



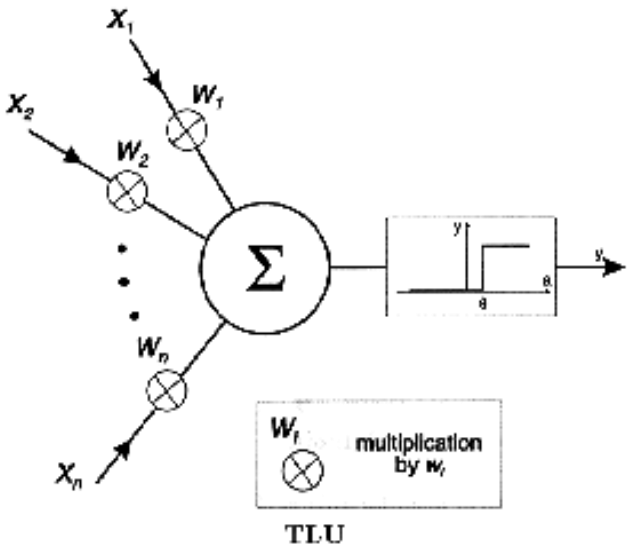
Perceptrons e Adalines

Regras Simples de Aprendizado

Perceptrons e Adalines

- Redes de uma única camada
  - Unidades Lineares com Threshold (TLUs)
    - Perceptron => Rosenblatt, 1958
  - Unidades Lineares (ULs)
    - Adaline => Widrow e Hoff, 1960
- Problemas de classificação
  - Dado um conjunto pré-especificado de entradas, uma certa entrada pertence ou não a este conjunto?

TLU - Threshold Linear Unit



TLU

$$a = \sum_{i=1}^{N} x\ w_i \qquad y = \begin{cases} 1, & \text{se } a \geq \theta \\ 0, & \text{se } a < \theta \end{cases}$$

Algoritmo de aprendizado do Perceptron (1/2)

1. Inicializar η e o vetor de pesos w
2. Repetir
   - Para cada par do conjunto de treinamento (x,t)
   - Atualizar o vetor de pesos para cada um dos nós da rede segundo a regra $w_i(t+1) = w_i(t) + \eta(t - o)x_i$
3. Até $o=t$ para todos os vetores

Algoritmo de aprendizado do Perceptron (2/2)

- Teorema da convergência (Minsky e Papert, 1969)
  - O algoritmo converge dentro de um número finito de passos para um vetor de pesos que classifica corretamente todo o conjunto de treinamento
    - Dado que o conjunto de treinamento é linearmente separável

Gradiente Descendente e a Regra Delta

- Widrow e Hoff (1960) para Adalines
- Uso do gradiente descendente para buscar o vetor de pesos que melhor se ajuste ao conjunto de treinamento
  - *Independentemente* do conjunto de treinamento ser *linearmente separável*

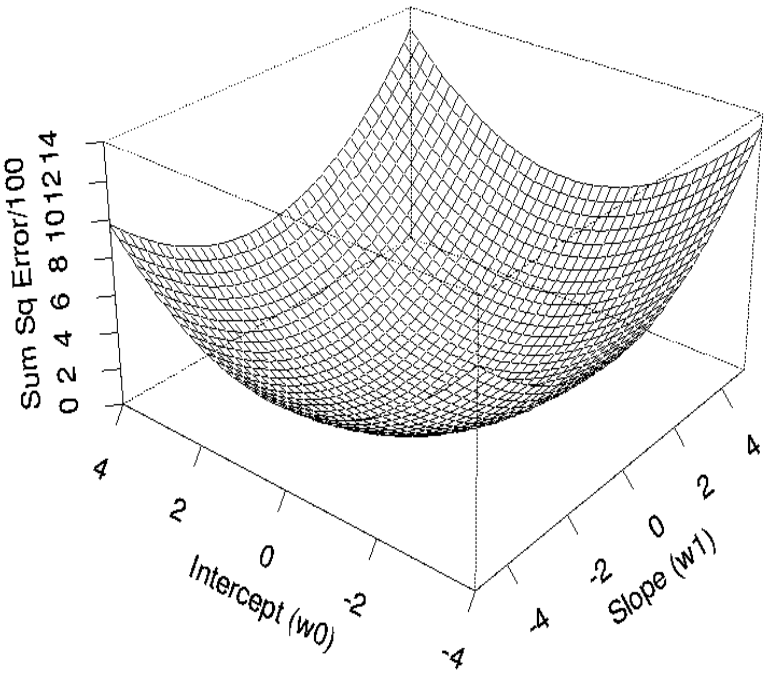
Regra Delta

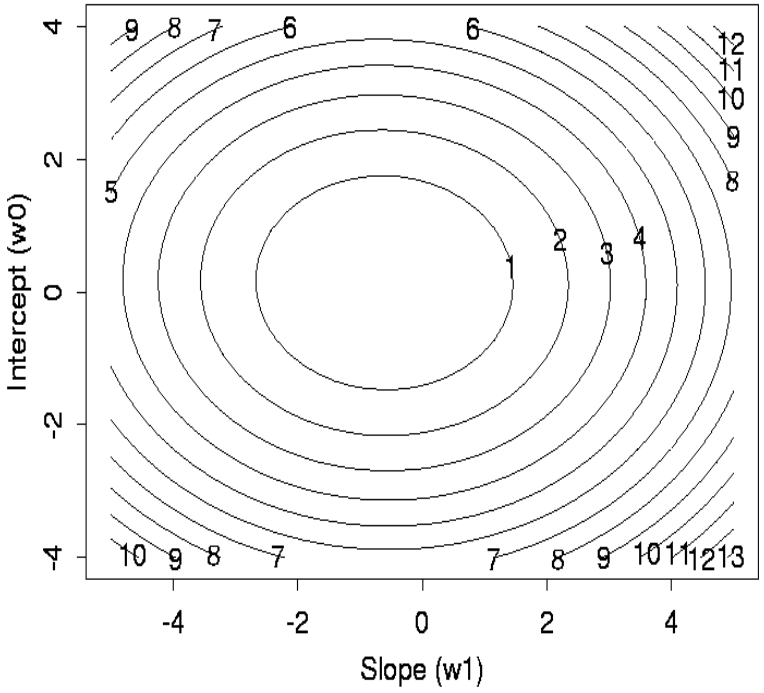
- Considere a tarefa de treinar um perceptron sem limiar (unthresholded), i.e., uma Unidade Linear (UL):
  - $y=w.x$ (1)
- Especifique uma medida de erro (função de custo) para o treinamento
  - $E(w)=1/2 \sum_{d \in D} (t_d - o_d)^2$ (2)

Função de Custo (1/2)

- $E(w)=1/2 \sum_{d \in D} (t_d - o_d)^2$, em que
  - D é o conjunto de treinamento
  - t_d é a saída desejada para o exemplo *d*
  - o_d é a saída da UL para o exemplo *d*
- Caracterizamos E como *uma função dos pesos* *w* e *não da saída linear* *o*
  - $o=w.x$
  - D é fixo durante o treinamento

Função de Custo (2/2)





Minimização do custo (1/3)

- A função E é uma medidade objetiva do erro preditivo para uma escolha especifíca de vetor de pesos
- Objetivo: encontrar um vetor w que minimize E
- Solução: usar técnicas de gradiente descendente

Minimização do custo (2/3)

1. Escolha valores iniciais arbitrários para os pesos
2. Calcule o gradiente da função de custo com respeito a cada peso
3. Mude os pesos tal que haja um deslocamento pequeno na direção de $-G$

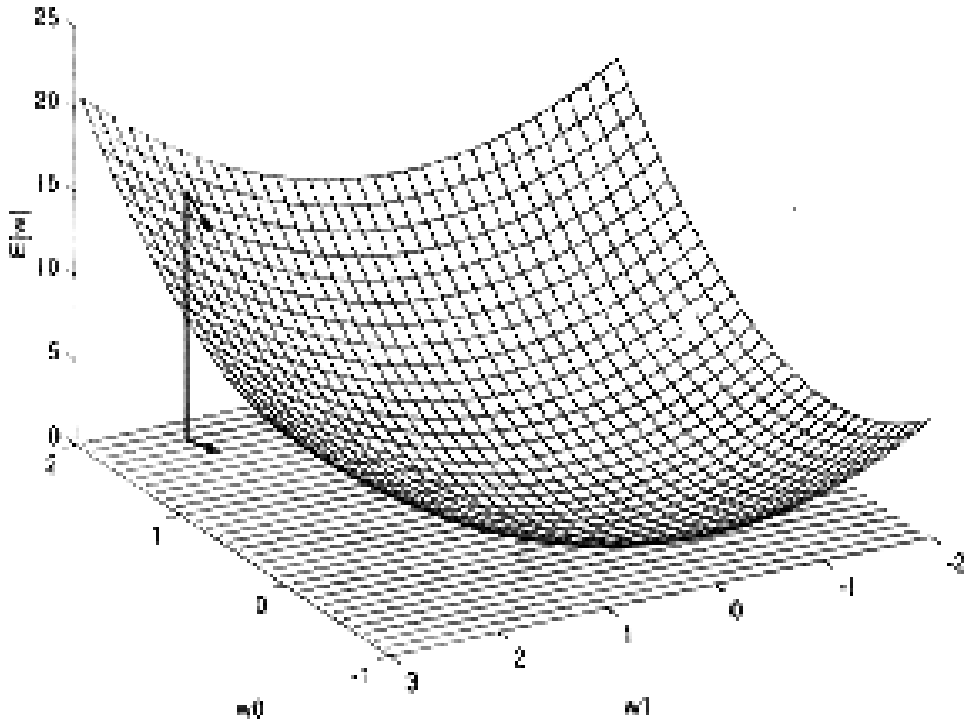
    $-G$ => Maior taxa de diminuição do erro

1. Repita passos 2 e 3 até que erro se aproxime de zero

Como este algoritmo funciona?

Minimização do custo (3/3)

- Uma busca basedada no gradiente descente determina o vetor de peso que minimiza E, começando com um vetor arbitrario inicial
- Modique o vetor repetidamente, em passos pequenos
- Em cada paso, o vetor é alterado na direção que produz a descida mais acentuada (steepest descent) ao longo da superfíie do erro
- Continue até que um mínimo global seja encontrado



Derivação da regra do gradiente descendente (1/3)

- Como podemos calcular a direção da descida mais acentuada ao longo da superfície do erro?
  - Calcule a derivada de E com respeito a cada componente do vetor *w*
  - Gradiente E com respeito *w*

$$\Delta E(w) \equiv [\partial E/\partial w_0, \partial E/\partial w_1, \ldots, \partial E/\partial w_0] \quad (3)$$

    - Vetor de derivadas parciais
    - Especifica a direção que produz o aumento mais acentuado em E

Derivação da regra do gradiente descendente (2/3)

- Visto que o gradiente especifica a direção de maior crescimento de E, a regra do gradiente é:
  - $w \leftarrow w + \Delta w$, onde
  - $\Delta w = -\eta \Delta E(w)$ (4), onde
    - η é uma constante positiva - taxa de aprendizado
    - O sinal negativo força o deslocamento do vetor de peso na direção que E decresce
  - $w_i(t+1) \leftarrow w_i(t) + \Delta w_i$
  - $\Delta w_i = -\eta \, (\partial E / \partial w_i)$ (5)

Derivação da regra do gradiente descendente (3/3)

- O vetor de derivadas $\partial E/\partial w_i$ que forma o gradiente pode ser obtido derivando E:
- $\partial E/\partial w_i = \partial/\partial w_i \; 1/2 \sum_{d \in D} (t_d - o_d)^2$
- $= 1/2 \sum_{d \in D} \partial/\partial w_i \; (t_d - o_d)^2$
- $= 1/2 \sum_{d \in D} 2 \; (t_d - o_d) \; \partial/\partial wi \; (t_d - o_d)$
- $= \sum_{d \in D} (t_d - o_d) \; \partial/\partial wi \; (t_d - w.x_d)$
- $\partial E/\partial w_i = \sum_{d \in D} (t_d - o_d)(-x_{id})$ (6), onde
  - x_{id} denot um único componente de entrada x_i do exemplo de treinamento d
- $\Delta w_i \; (t+1) \leftarrow \Delta w_i \; (t) + \eta \; (t_d - o_d) x_i$ (6) em (5) (7)

Algoritmo de Widrow e Hoff

- Inicializar η e o vetor de pesos w
- Repetir
- Inicializar Δw_i com zero
- Para cada par do conjunto de treinamento (x,t)
  - Calcule a saída o
  - Para cada peso w_i das LUs
    - Calcule $\Delta w_i \leftarrow \Delta w_i + \eta (t - o)x_i$
- Para cada peso w_i das LUs
  - $w_i \leftarrow w_i + \Delta w_i$
- Até que uma condição de término seja satisfeita

Observações (1/2)

- A estratégia do gradiente descendente pode ser aplicada sempre que:

1. O espaço de hipóteses contém hipóteses parametrizadas contínuas (e.g. os pesos *w* de uma LU)
2. O erro pode ser diferenciável com relação a estes parâmetros das hipóteses

Observações (2/2)

- Dificuldades prática da estratégia do gradiente descendente:

1. Convergência para um mínimo local pode ser muita lenta (várias iterações)
2. Se existir vários mínimos locais na superfície de erro, não haverá garantia que o algoritmo encontrará o mínimo

Aproximação estocástica

- Gradiente descendente incremental ou gradiente descendente estocástico

  – Aproxima a busca do gradiente descendente atualizando os pesos a cada exemplo individual

    - $\Delta w_i = \eta (t - o) x_i$

Gradiente descendente estocástico

- Inicializar η e o vetor de pesos w
- Repetir
- Inicializar Δw_i com zero
- Para cada par do conjunto de treinamento (x,t)
  - Calcule a saída o
  - Para cada peso w_i das LUs
    - Calcule $w_i \leftarrow w_i + \eta (t - o)x_i$
    - {Para cada peso w_i das Lus $w_i \leftarrow w_i + \Delta w_i$ }
- Até que uma condição de término seja satisfeita

Gradiente descendente Padrão X Estocástico

- O erro de cada exemplo é acumulado antes da atualização dos pesos X Os pesos são atualizados após a apresentação de cada exemplo

- O gradiente padrão requer mais computação por cada passo de atualização dos pesos
  - Um taxa maior de aprendizagem pode ser utilizada

- Em problemas com vários mínimos locais, o gradiente estocástico pode evitar a queda em um desses mínimos

Perceptrons/Adalines (1/2)

- Poder Computacional
  - Representam uma superfície de decisão através de um hiperplano
    - *o*=1 para exemplos situados em um lado
    - *o*=0 para exemplos situados no outro lado
    - Exemplos *linearmente separáveis*
  - Apenas funções *linearmente separáveis* (Minsky e Papert, 1969)

Perceptrons/Adalines (2/2)

- Poder Computacional
  - Podem representar todas as funções Booleanas primitivas (AND, OR, NAND e NOR)
    - Não podem representar o XOR
  - Qualquer função booleana pode ser representada por um perceptron de *duas camadas*
    - Forma normal disjuntiva
- Regressão Linear

Regra do Perceptron x Regra Delta (1/3)

- Atualização dos pesos

1. baseada no erro da saída da rede após a aplicação do limiar (*thresholded output*)
2. baseada no erro da saída da rede sem a aplicação do limiar (*unthresholded output*)

Regra do Perceptron x Regra Delta (2/3)

- Convergência

1. Converge dentro de um *número finito de passos* para um vetor de pesos que classifica *corretamente* todo o conjunto de treinamento

   – Dado que o conjunto de treinamento seja *linearmente separável*

Regra do Perceptron x Regra Delta (3/3)

- Convergência

1. Converge apenas assintoticamente para um vetor de pesos com um erro mínimo, possivelmente requererendo um *número ilimitado de pasos*

    - *Independentemente* de o conjunto de treinamento ser *linearmente separável*