Inteligência Artificial

Redes Neurais Artificiais

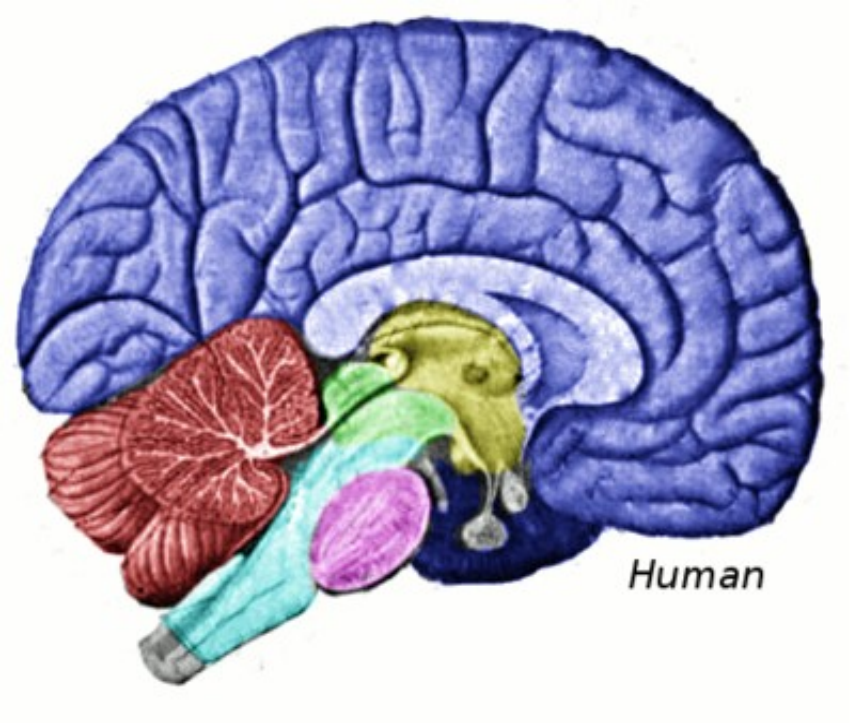
Sumário

- Introdução
  - Neurônios Biológicos
- Neurônios Artificiais
  - Modelo MCP
  - Funções de Ativação
- Aprendizado
  - Supervisionado
  - Não Supervisionado
  - Por Reforço
- Redes Perceptron
  - Algoritmo de aprendizado do Perceptron
- Redes MLP
  - Algoritmo *Backpropagation*

Redes Neurais Artificiais
Neurônios

Introdução

Cérebro

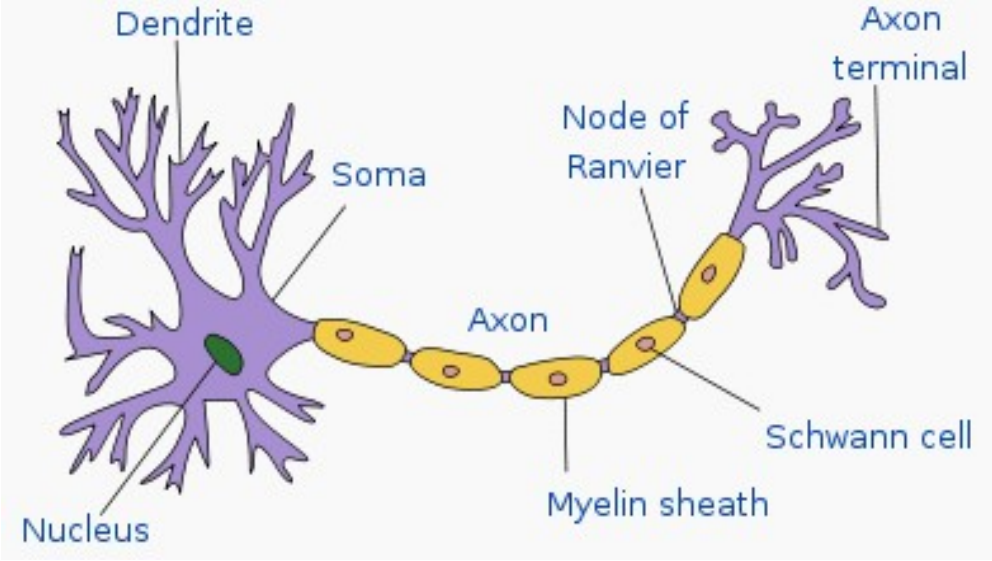


- "Inventado" há vários milhões de anos
- Cerca de 10^{11} Neurônios
  - Cada um com cerca de 10^{3} conexões sinápticas
  - Responsável por grande parte do comportamento racional

Computador

- Inventado há menos de 100 anos.
- Cerca de 10^8 transistores
  - Cada um com apenas 1 entrada!

Neurônio Biológico



- Dendritos recebem impulsos nervosos
- Corpo celular processa
- Axônio transmite para os próximos neurônios
- Unidade básica
  - Processamento paralelo e assíncrono

Aplicações de RNAs

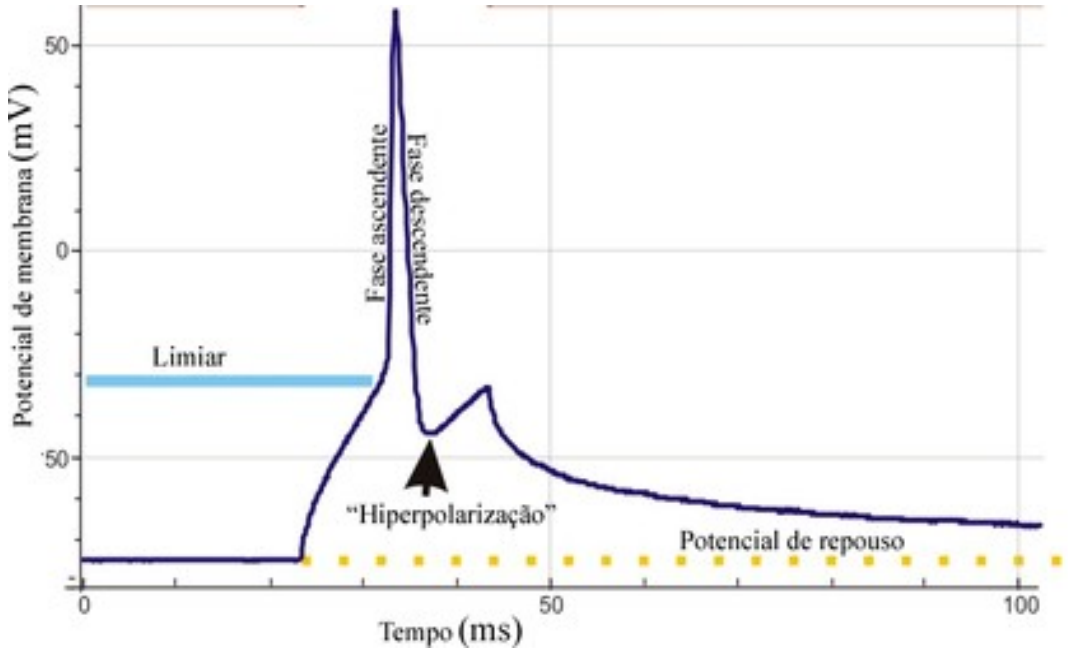
- Classificação de Padrões
  - Reconhecimento de caracteres
  - Identificação de Erros em Linhas de Produção
  - Diagnóstico de Doenças
  - etc.
- Modelagem de Sistemas
  - Modelagem de Sistemas complexos
  - Previsão do tempo
  - Previsão de Séries Temporais
  - etc.

Redes Neurais Artificiais
Neurônios

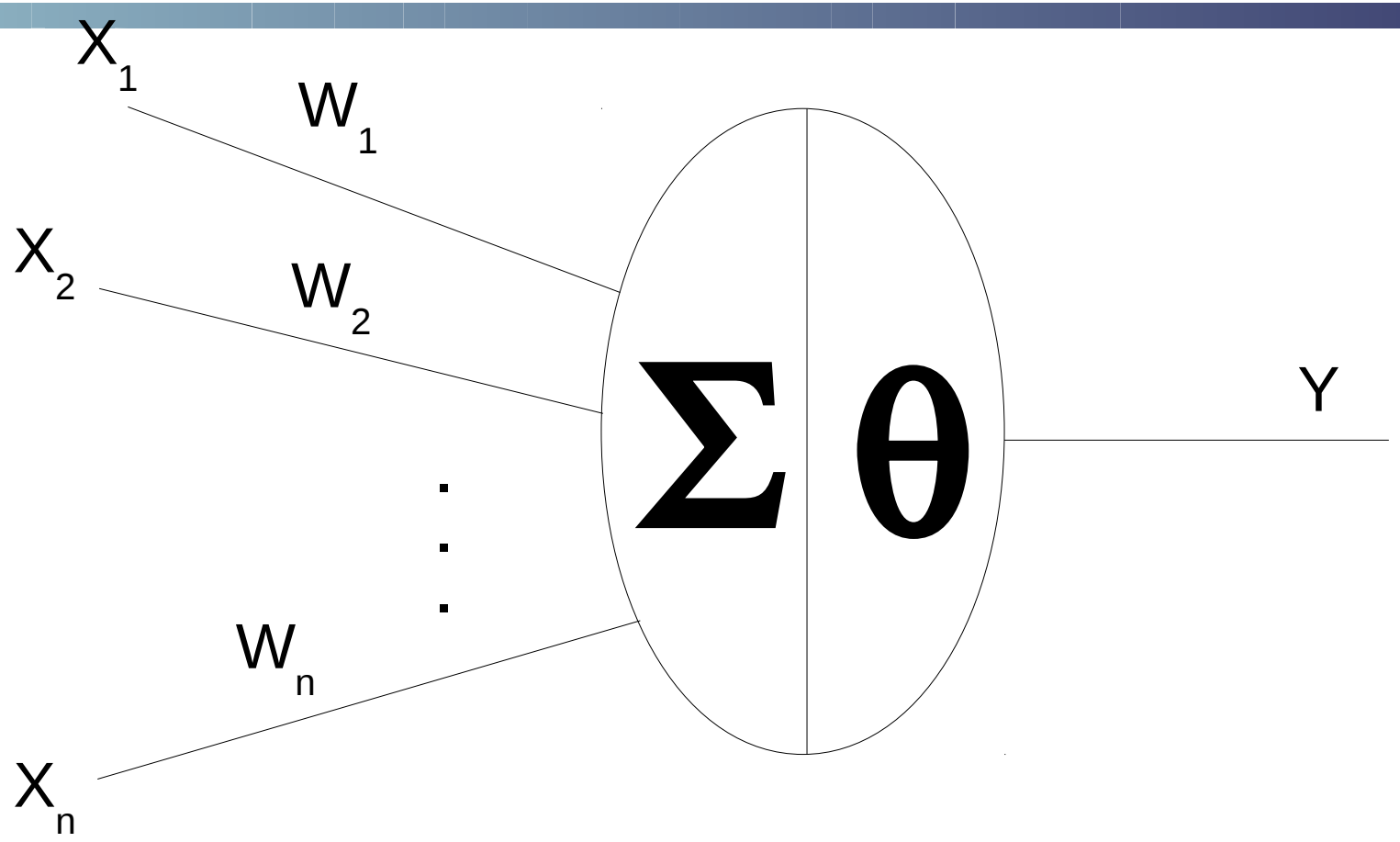
Neurônios Artificiais

Neurônio Biológico Funcionamento

- Um neurônio biológico “dispara” quando a soma dos impulsos que ele recebe ultra-passa o seu limiar

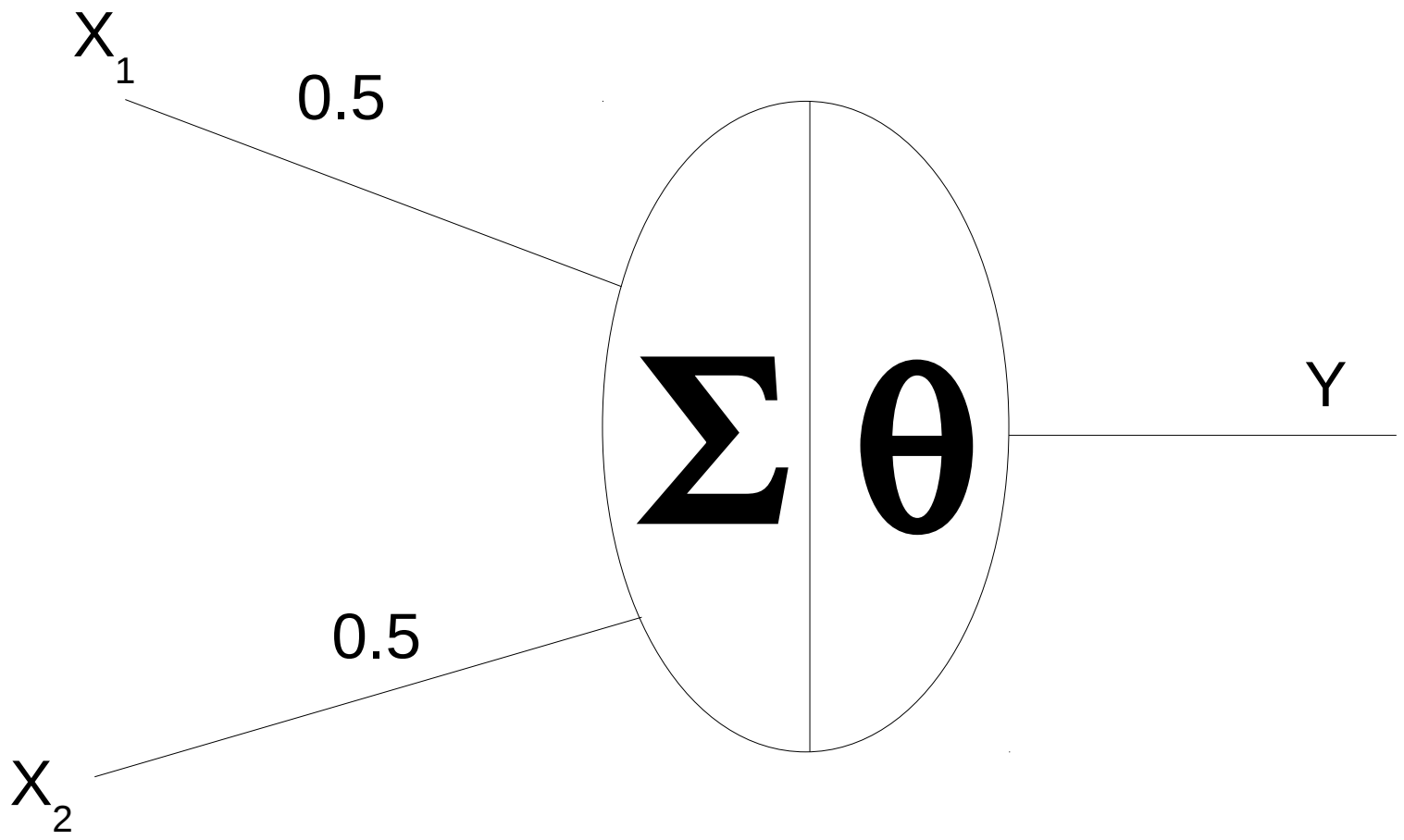


Modelo MCP do Neurônio



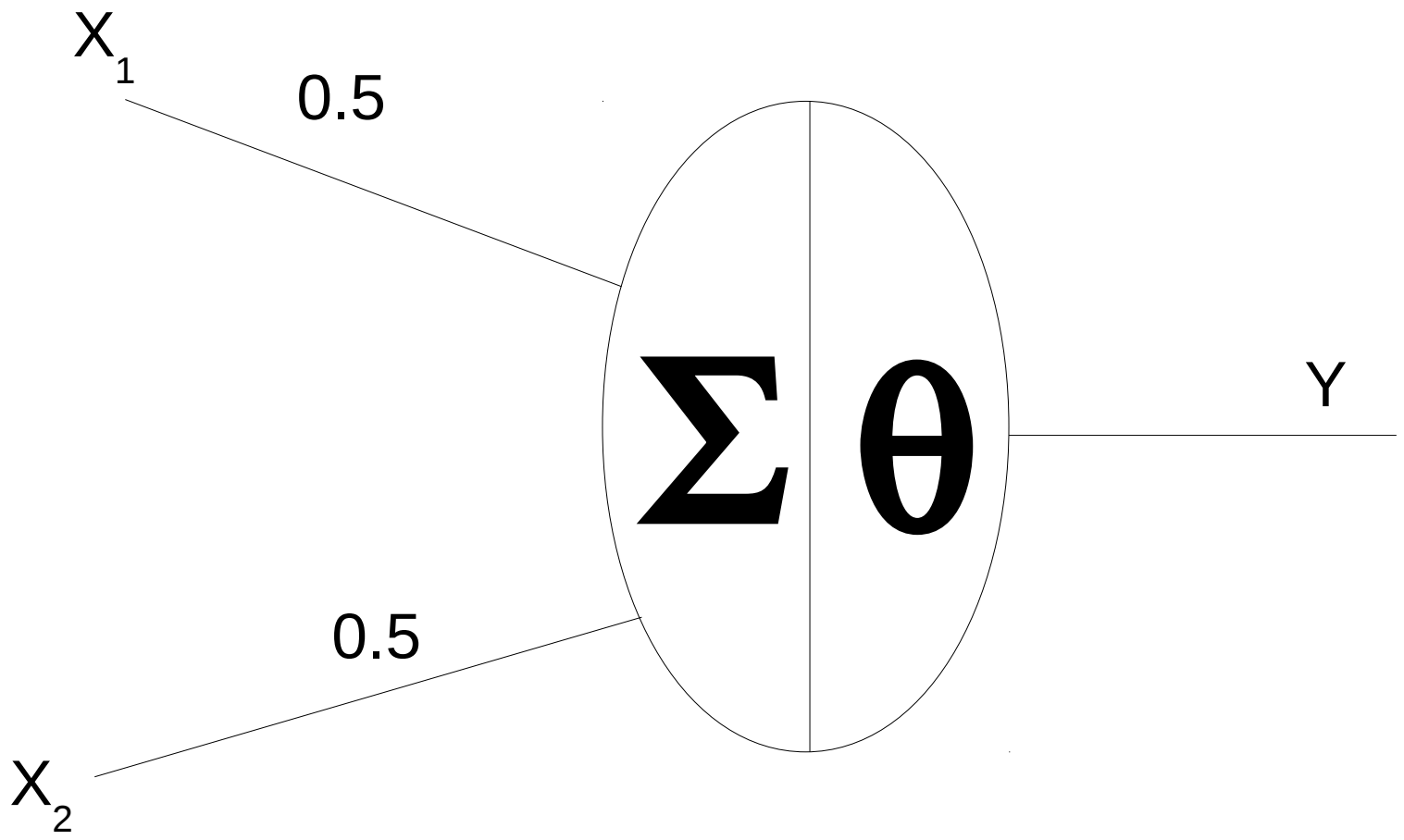
- Y estará ativa quando $\sum_{i=1}^{n} x_i w_i \geq \theta$

Exemplo: Porta AND



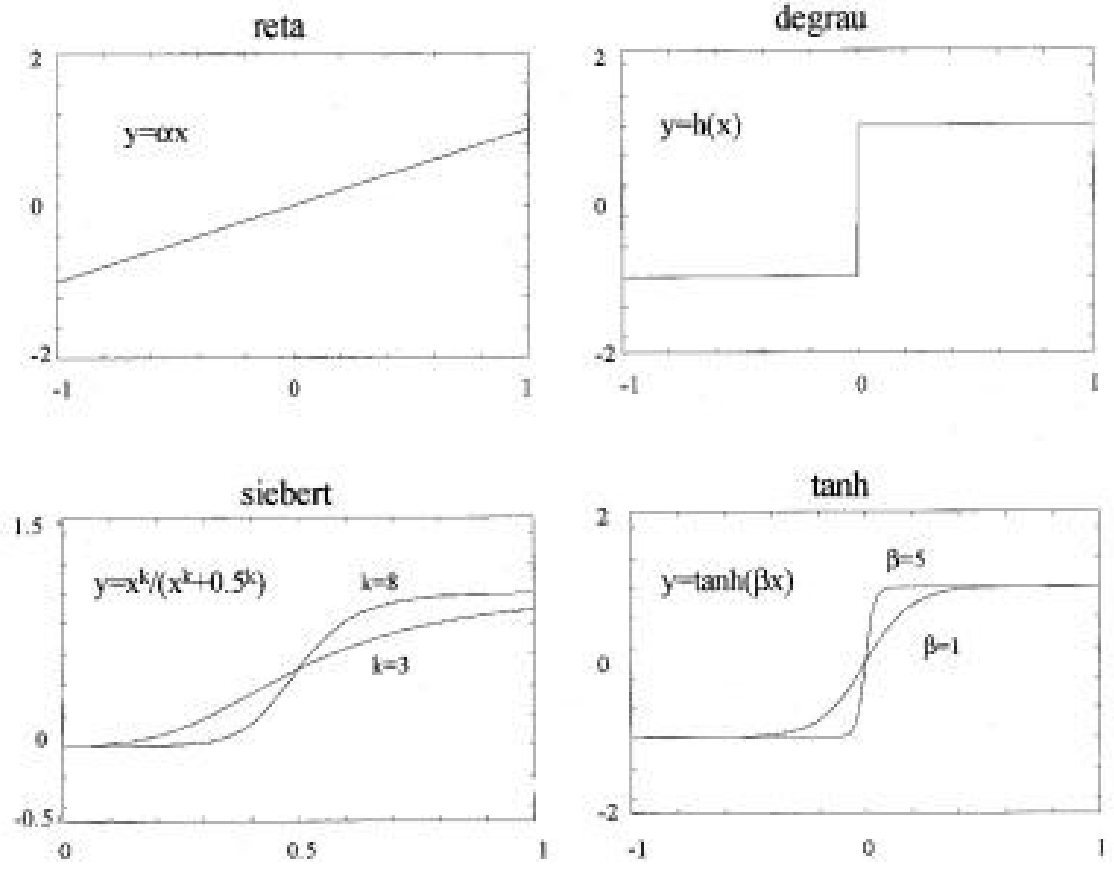
$$\theta = 1.0$$

Exemplo: Porta OR



$$\theta = 0.5$$

Funções de Ativação
Exemplos



Redes Neurais Artificiais
Neurônios

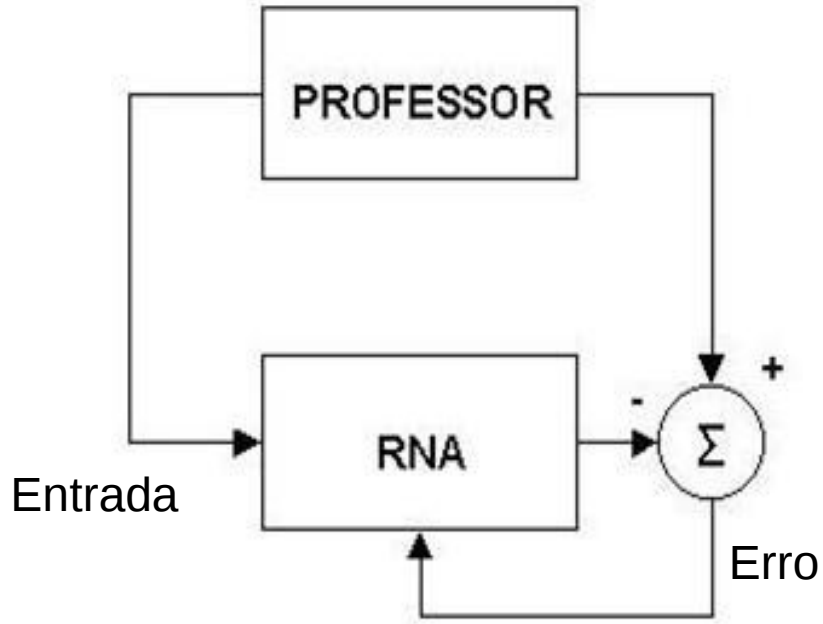
Aprendizado

O que significa “Aprender”?

- Aprendizado Conexionista
  - Não utiliza abordagem Simbólica como na IA clássica
  - Tenta-se determinar a *intensidade* das conexões entre neurônios
    - A *intensidade* é dada pelo peso das sinapses

- Para a rede neural *aprender*, atualiza-se seus pesos (***W***) até obter-se o resultado desejado

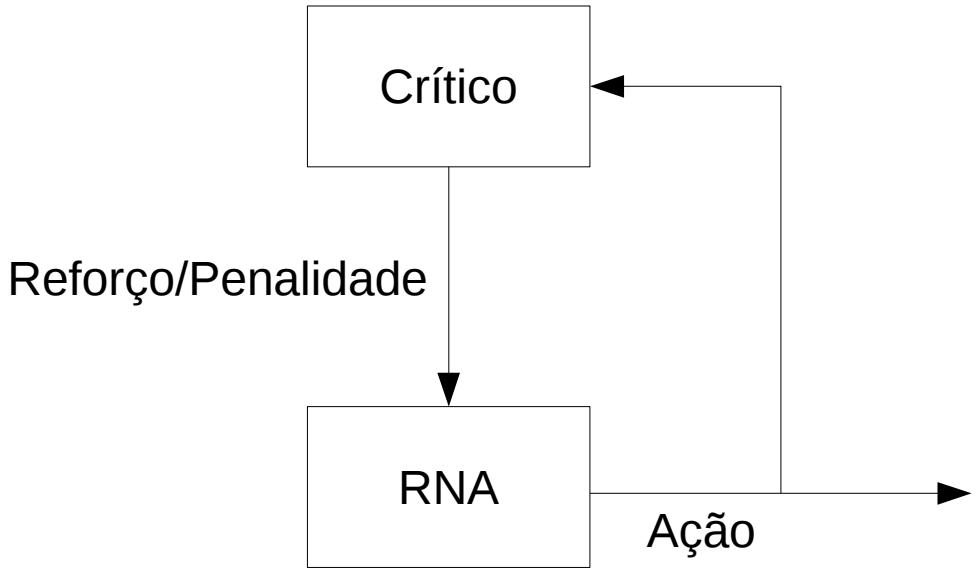
Aprendizado Supervisionado



Aprendizado Não Supervisionado

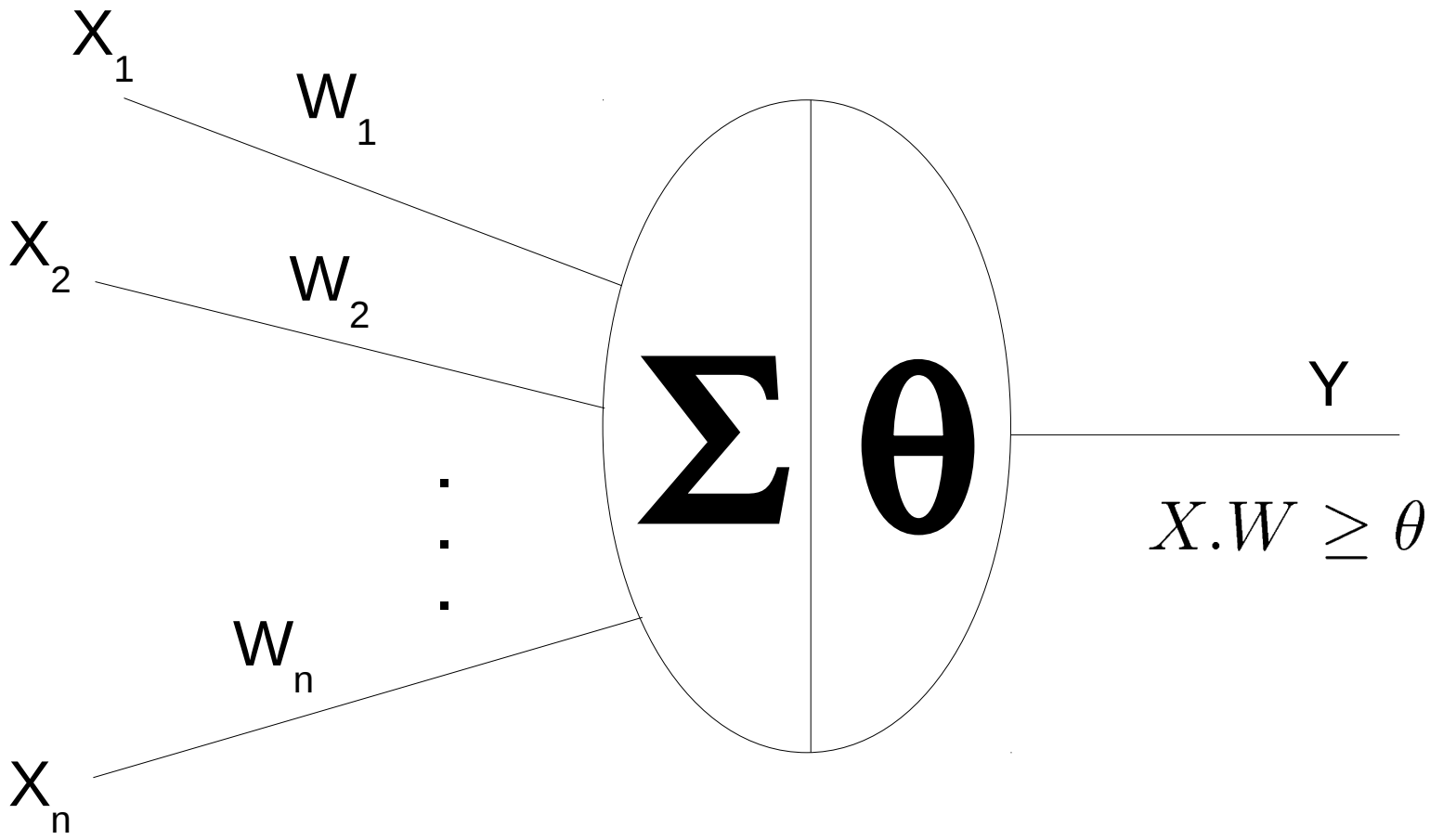
- Muitos sistemas biológicos funcionam como aprendizado não-supervisionado
  - Primeiros estágios da Visão e Audição
    - Identificação de retas, ângulos, etc...
- O ajuste do peso não é feito com base no erro
  - Não existe professor para dizer *quão perto* está a solução
  - O ajuste é feito com base na *utilização* da sinapse
    - Quanto mais ativada uma entrada, mais ela é importante e maior o seu peso

Aprendizado Supervisionado



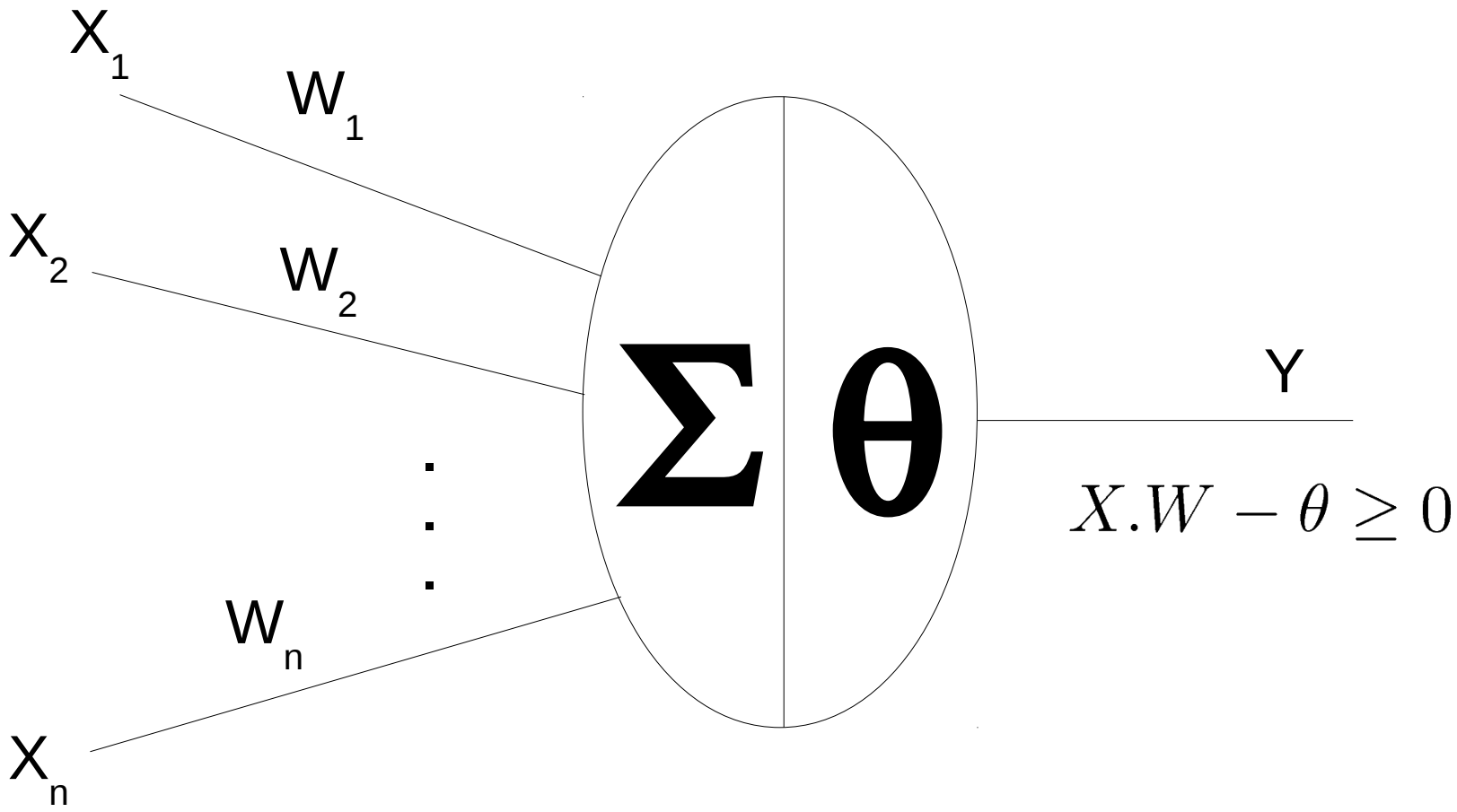
Perceptron

- Usa o modelo MCP



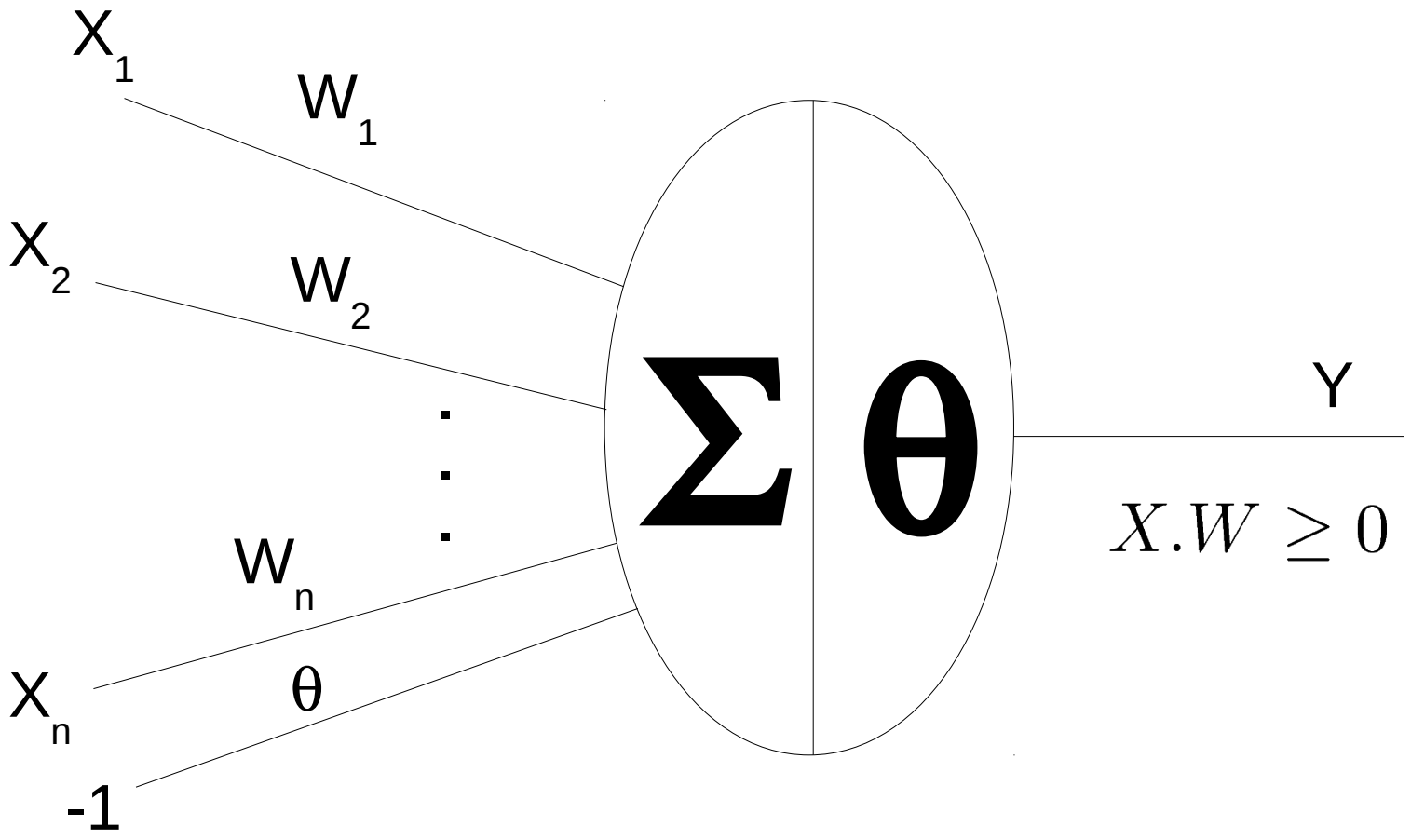
Perceptron

- Usa o modelo MCP



Perceptron

- Usa o modelo MCP



Atualização de Pesos

- Regra para atualizar pesos no Perceptron

$$W(t+1) = W(t) + \eta e X(t)$$

  - Onde η, a taxa de aprendizagem, é uma constante;
  - *e* é o erro (valor desejado – saída da rede);
  - X é o vetor de entrada e W é o vetor de pesos.

Algoritmo de Treinamento do Perceptron

- Inicializar a taxa de aprendizado e o vetor de pesos
- Repetir
  - Para cada par do conjunto de treinamento
    - Atualizar o peso para cada um dos nós da rede, utilizando a regra

$$W(t+1) = W(t) + \eta e X(t)$$

- Até $e = 0$ para todos os elementos do conjunto de treinamento e todos os nós da rede

Exemplos

- Executar o algoritmo do perceptron para treinar uma RNA de apenas um nó para desempenhar a função AND;
- Executar o algoritmo do perceptron para treinar uma RNA de apenas um nó para desempenhar a função OR;
- Executar o algoritmo do perceptron para treinar uma RNA de apenas um nó para desempenhar a função XOR;

Exemplos

- Executar o algoritmo do perceptron para treinar uma RNA de apenas um nó para desempenhar a função AND;
- Executar o algoritmo do perceptron para treinar uma RNA de apenas um nó para desempenhar a função OR;
- Executar o algoritmo do perceptron para treinar uma RNA de apenas um nó para desempenhar a função XOR;
  - Não é possível! Não é linearmente separável...

Redes Neurais Artificiais

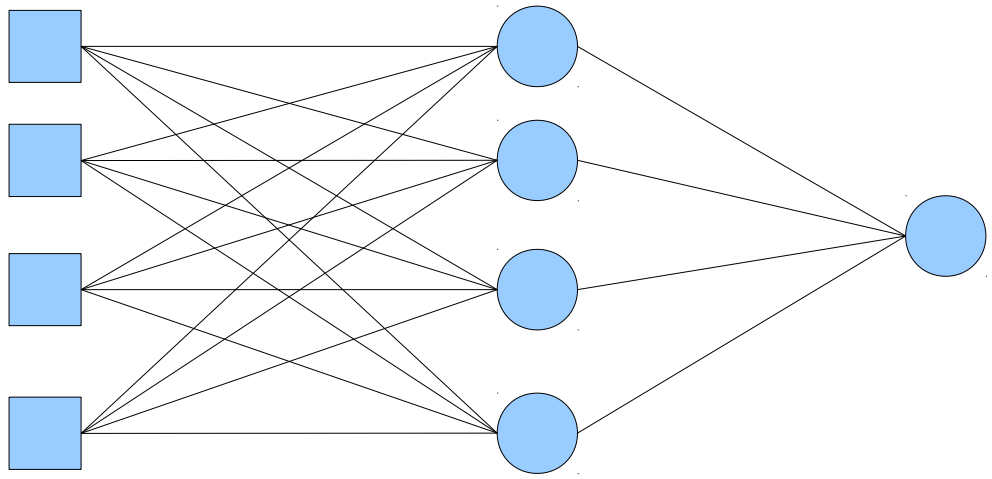
MLP

Multilayer Perceptron

Classes não linearmente separáveis

- Redes de apenas uma camada não são capazes de resolver problemas não linearmente separáveis
  - Para tal é preciso de redes de mais camadas
    - MLPs
  - MLPs de uma camada oculta podem implementar qualquer função contínua
    - Com duas ou mais camadas ocultas, pode-se implementar qualquer função
    - ATENÇÃO: Poder implementar qualquer função não significa que o algoritmo encontrará os parâmetros para implementá-la na prática!

Arquitetura de uma MLP



CAMADA DE ENTRADA

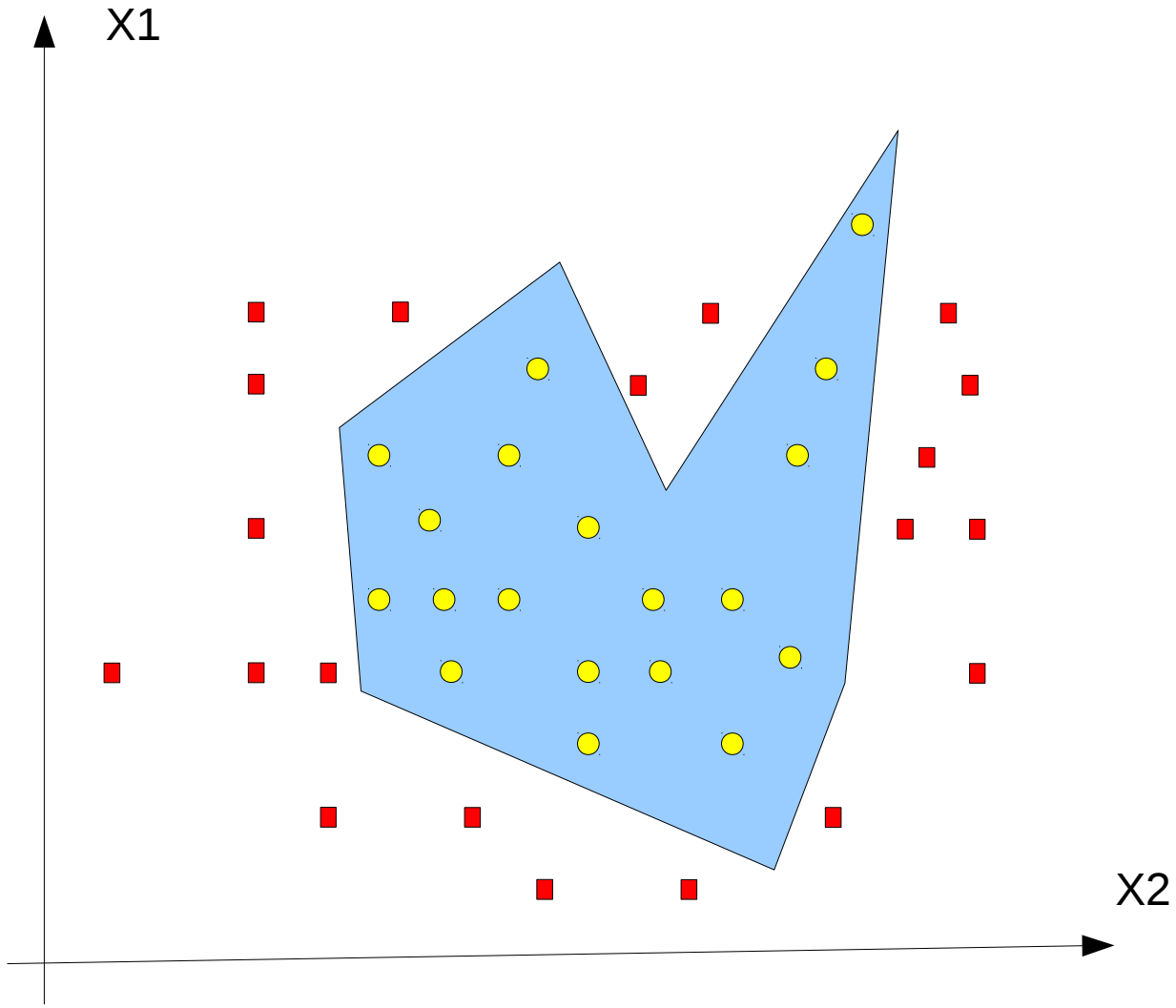
CAMADA OCULTA

CAMADA DE SAÍDA

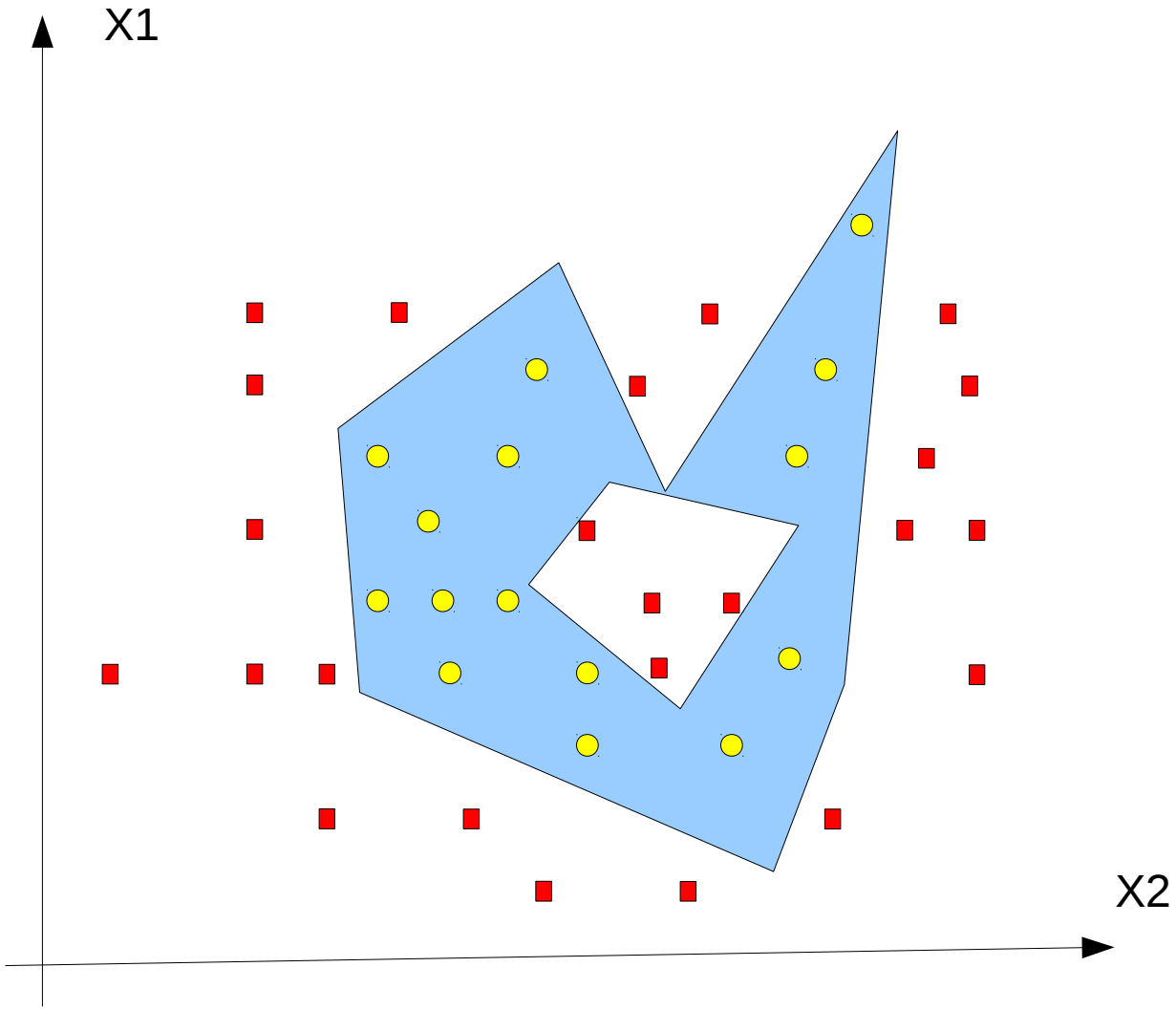
Funcionalidade

- Primeira Camada Oculta
  - Cada nó traça retas no espaço de padrões de treinamento
- Segunda Camada Oculta
  - Cada nó combina as retas traçadas pelos Neurônios da camada anterior, formando regiões convexas
- Camada de Saída
  - Cada nó forma regiões abstratas que são combinações das regiões convexas das camadas anteriores

Regiões
Segunda Camada Oculta



Regiões
Camada de Saída



Número de Nós nas Camadas Intermediárias

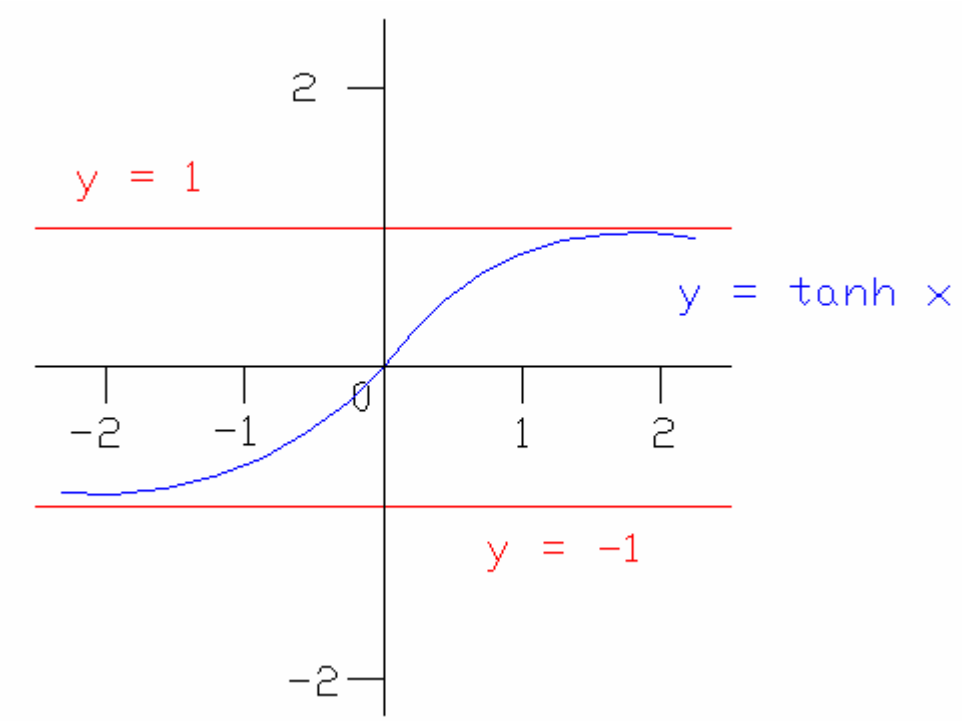
- O número de nós nas camadas intermediárias depende:
  - Número de Exemplos de Treinamento
  - Quantidade de ruído presente nos exemplos
  - Complexidade da Função a ser aprendida
  - Distribuição Estatística dos dados de treinamento

- Geralmente o número de nós é determinado empiricamente
  - Costuma-se utilizar o número de conexões dez vezes menor que o número de exemplos
    - Muitos nós → *overfitting*
    - Pucos nós → *underfitting*

Funções de Ativação

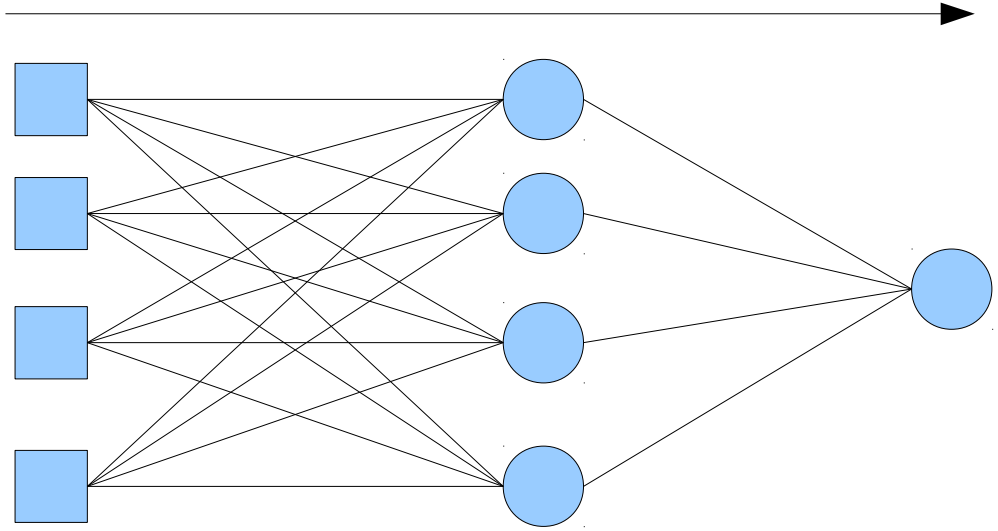
- Se cada nó de uma MLP utilizar uma função de ativação do tipo degrau (limiar), os nós não sabem quão distantes estão do valor desejado
  - As camadas intermediárias também não conseguem determinar seus erros
- Uma alternativa é utilizar uma função linear (rampa) como função de ativação
  - No entanto, é possível provar por meio de álgebra linear simples que uma rede com mais de uma camada com nós com funções de ativação lineares comporta-se como uma rede de apenas uma camada

Função Tangente Hipérbólica

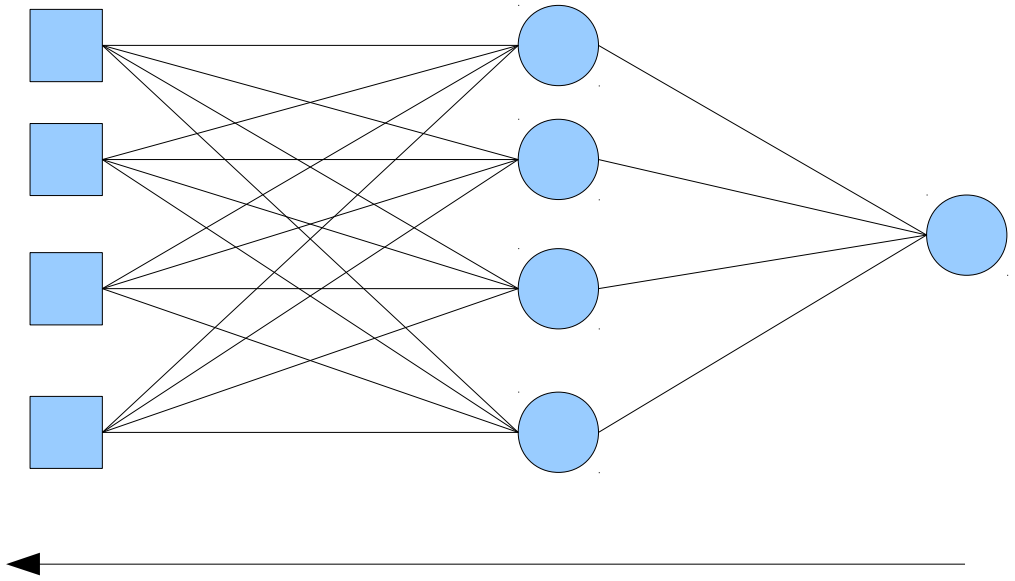


Algoritmo Backpropagation

Fase *Forward*

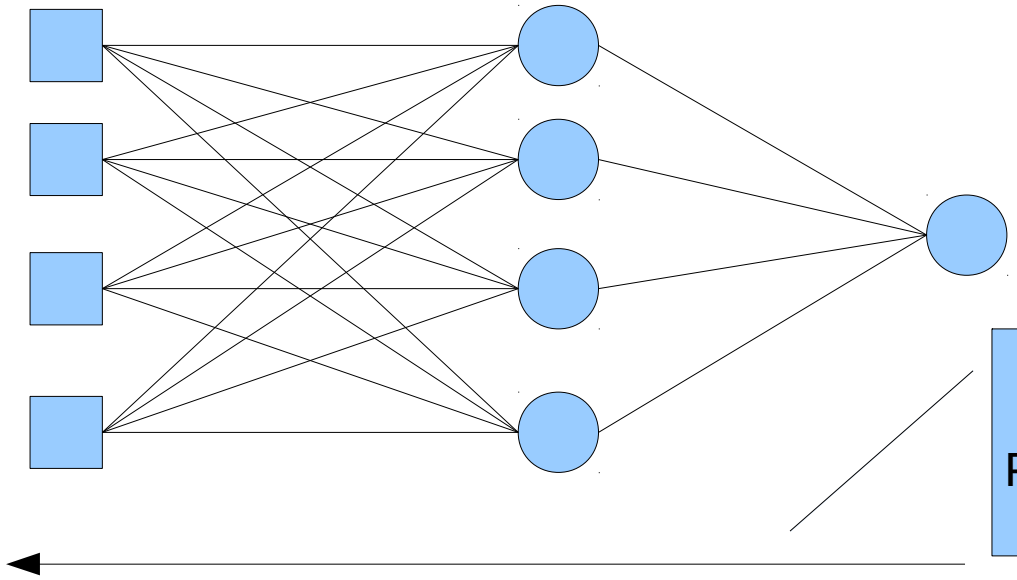


Algoritmo Backpropagation



Fase *Backward*

Algoritmo Backpropagation



O Erro é Propagado Para as camadas anteriores

Fase *Backward*

Atualização dos Pesos

- Camada de Saída
  - $\Delta w_{ji} = \eta \delta_j x_i$
  - $\delta_j = (d_j - y_j) f'(net_j)$

- Demais Camadas
  - $\Delta w_{ji} = \eta \delta_j x_i$
  - $\delta_j = f'(net_j) \sum_{l=0}^{M} \delta_l w_{lj}$

$$net_j = \sum_{i=0}^{n} x_i w_{ji}$$

Atualização dos Pesos

- Camada de Saída
  - $\Delta w_{ji} = \eta \delta_j x_i$
  - $\delta_j = (d_j - y_j) f'(net_j)$

Derivada da função de ativação

- Demais Camadas
  - $\Delta w_{ji} = \eta \delta_j x_i$
  - $\delta_j = f'(net_j) \sum_{l=0}^{M} \delta_l w_{lj}$

$$net_j = \sum_{i=0}^{n} x_i w_{ji}$$

Atualização dos Pesos

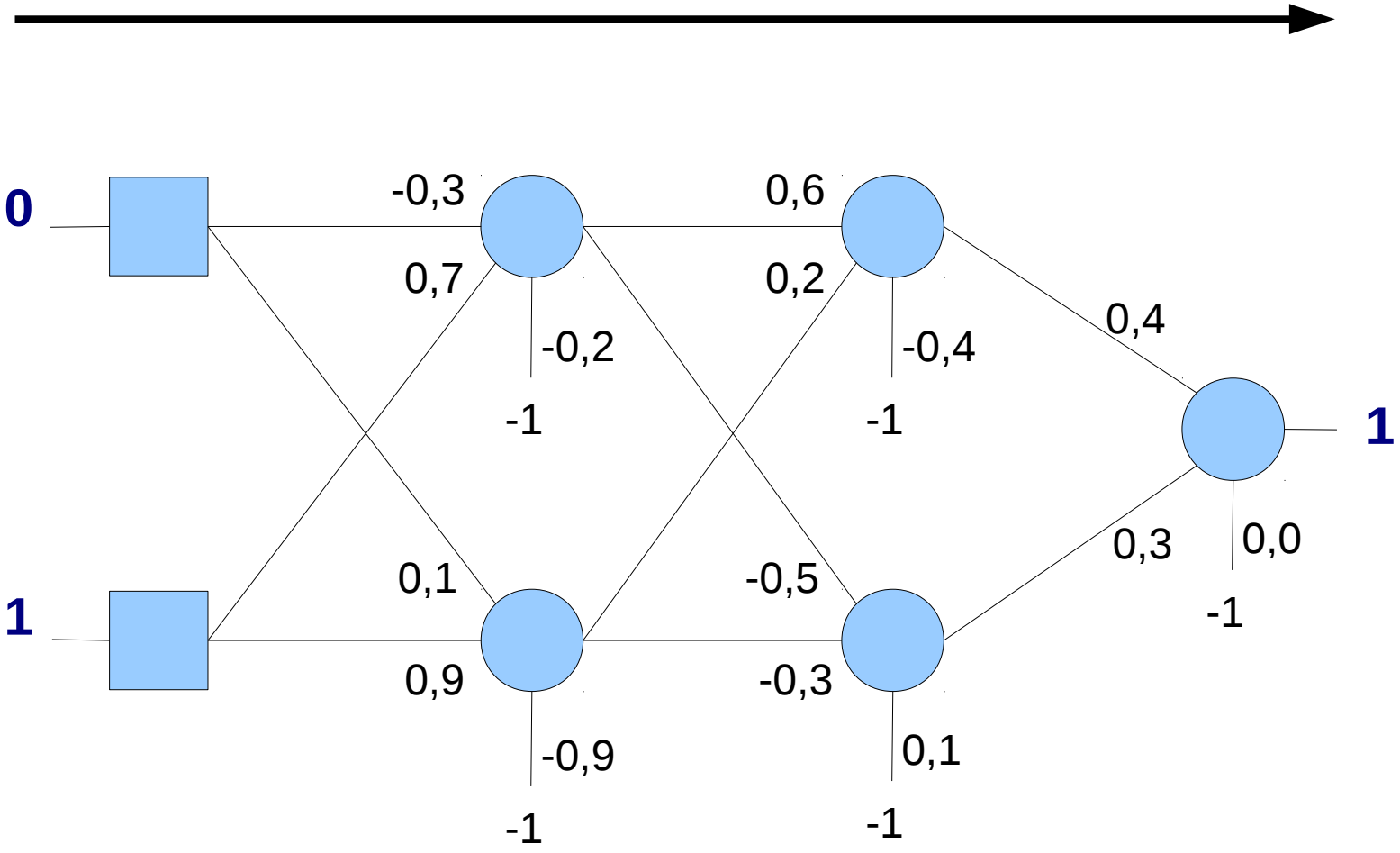
- Camada de Saída
  - $\Delta w_{ji} = \eta \delta_j x_i$
  - $\delta_j = (d_j - y_j) f'(net_j)$

- Demais Camadas
  - $\Delta w_{ji} = \eta \delta_j x_i$
  - $\delta_j = f'(net_j) \sum_{l=0}^{M} \delta_l w_{lj}$

Somatório dos δ dos nós da camada seguinte ponderados pelo peso da conexão

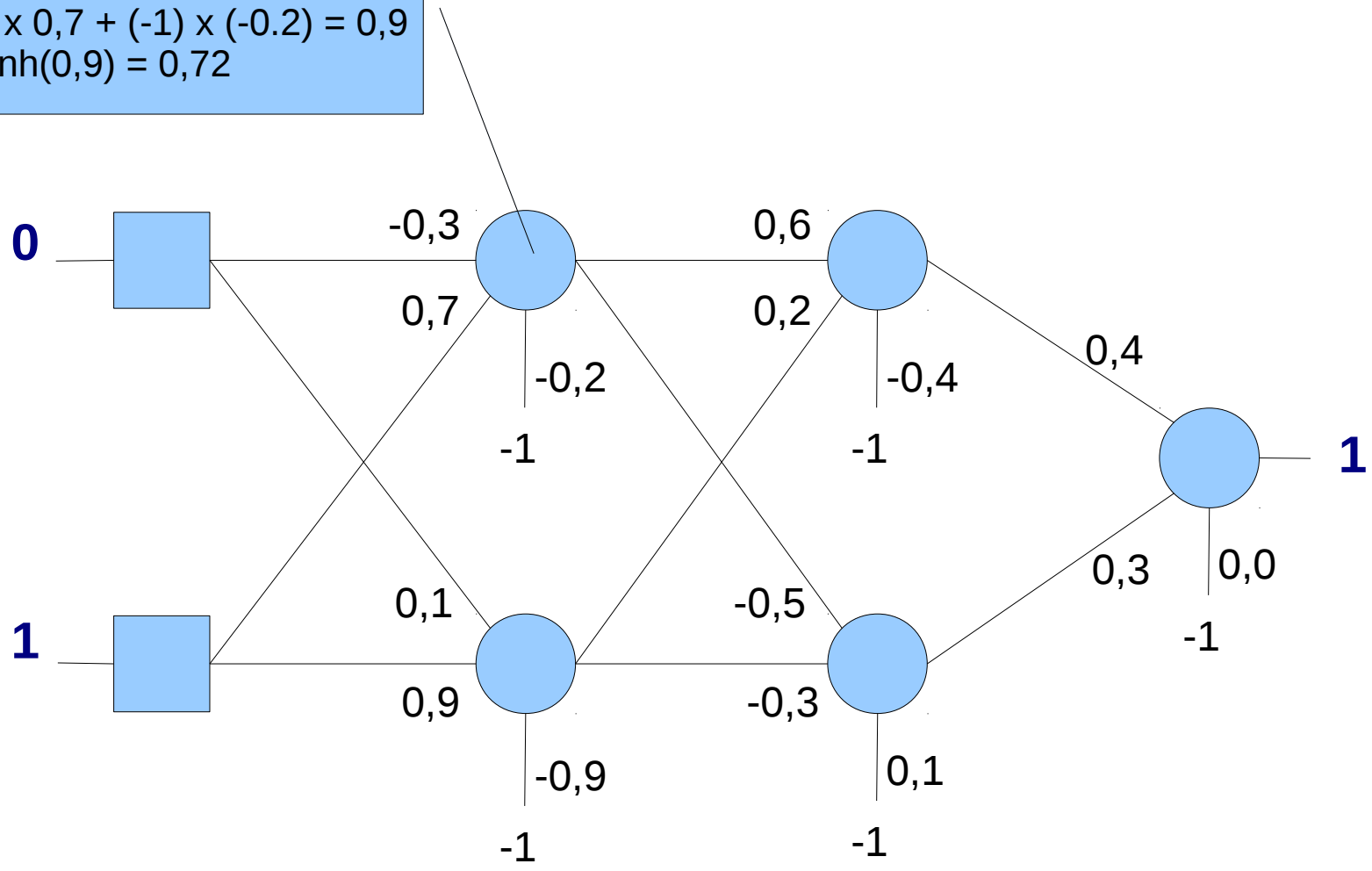
$$net_j = \sum_{i=0}^{n} x_i w_{ji}$$

Backpropagation



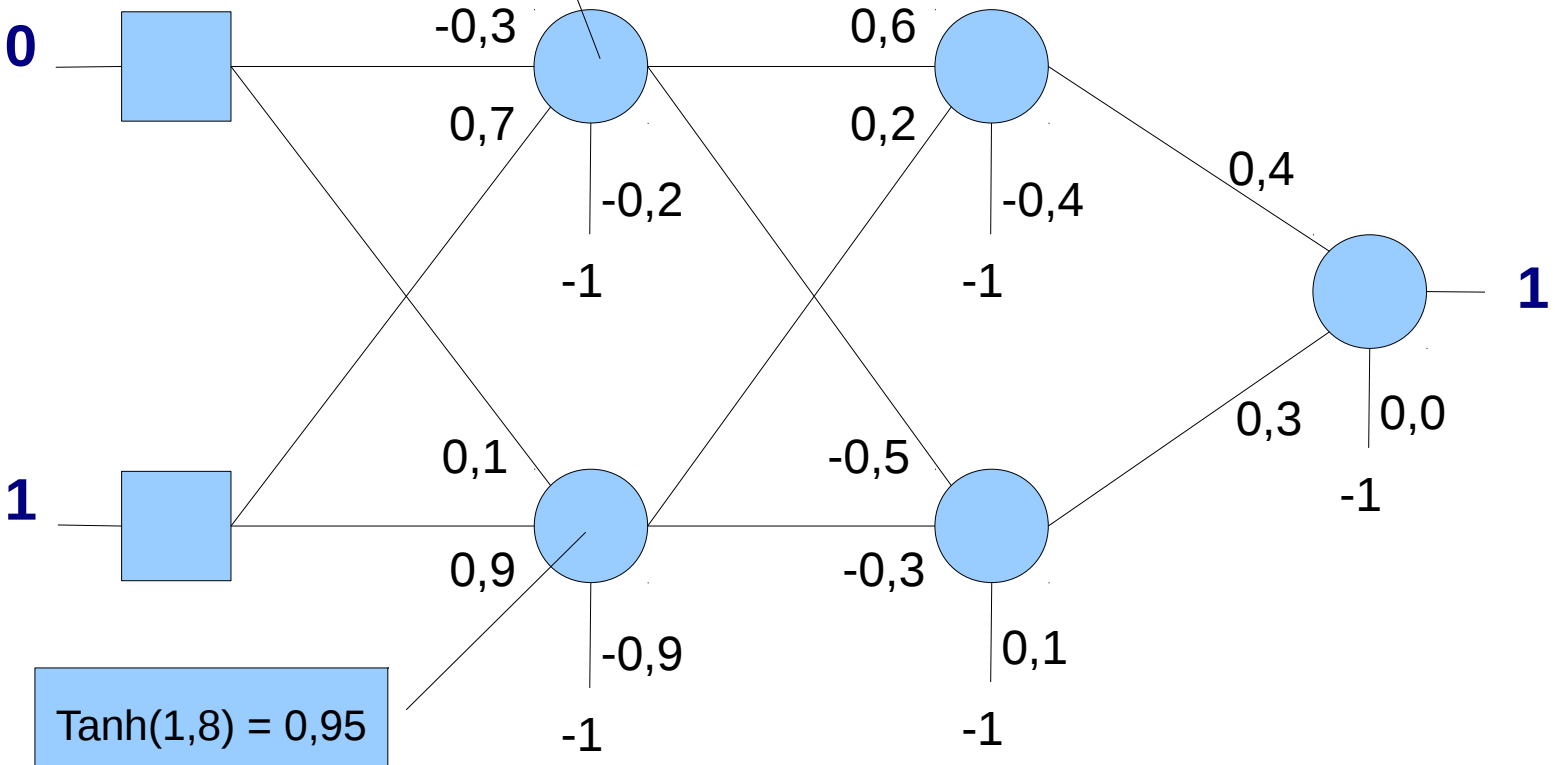
Backpropagation

0 x (-0,3) + 1 x 0,7 + (-1) x (-0.2) = 0,9
Tanh(0,9) = 0,72



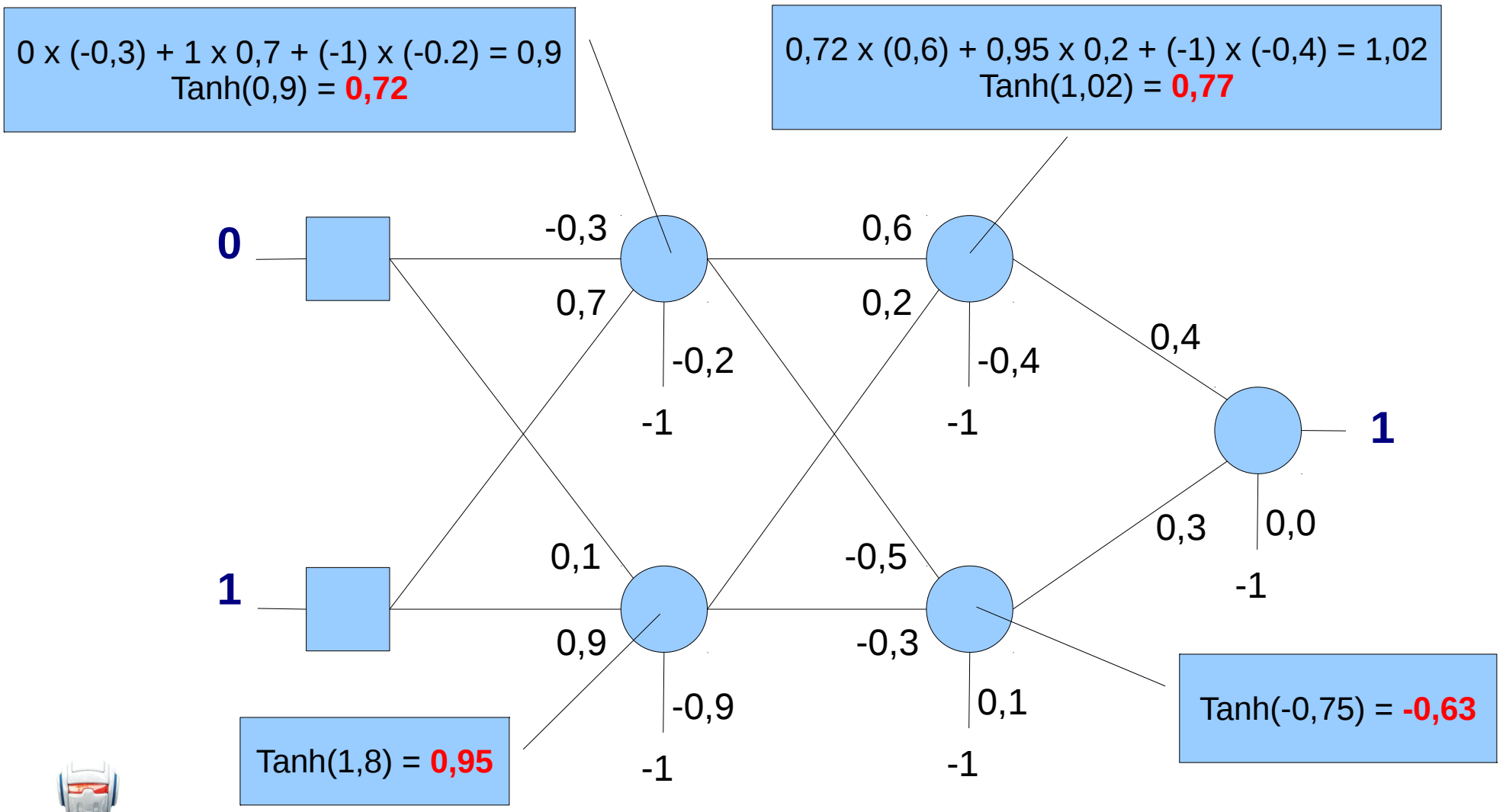
Backpropagation

0 x (-0,3) + 1 x 0,7 + (-1) x (-0.2) = 0,9
Tanh(0,9) = 0,72

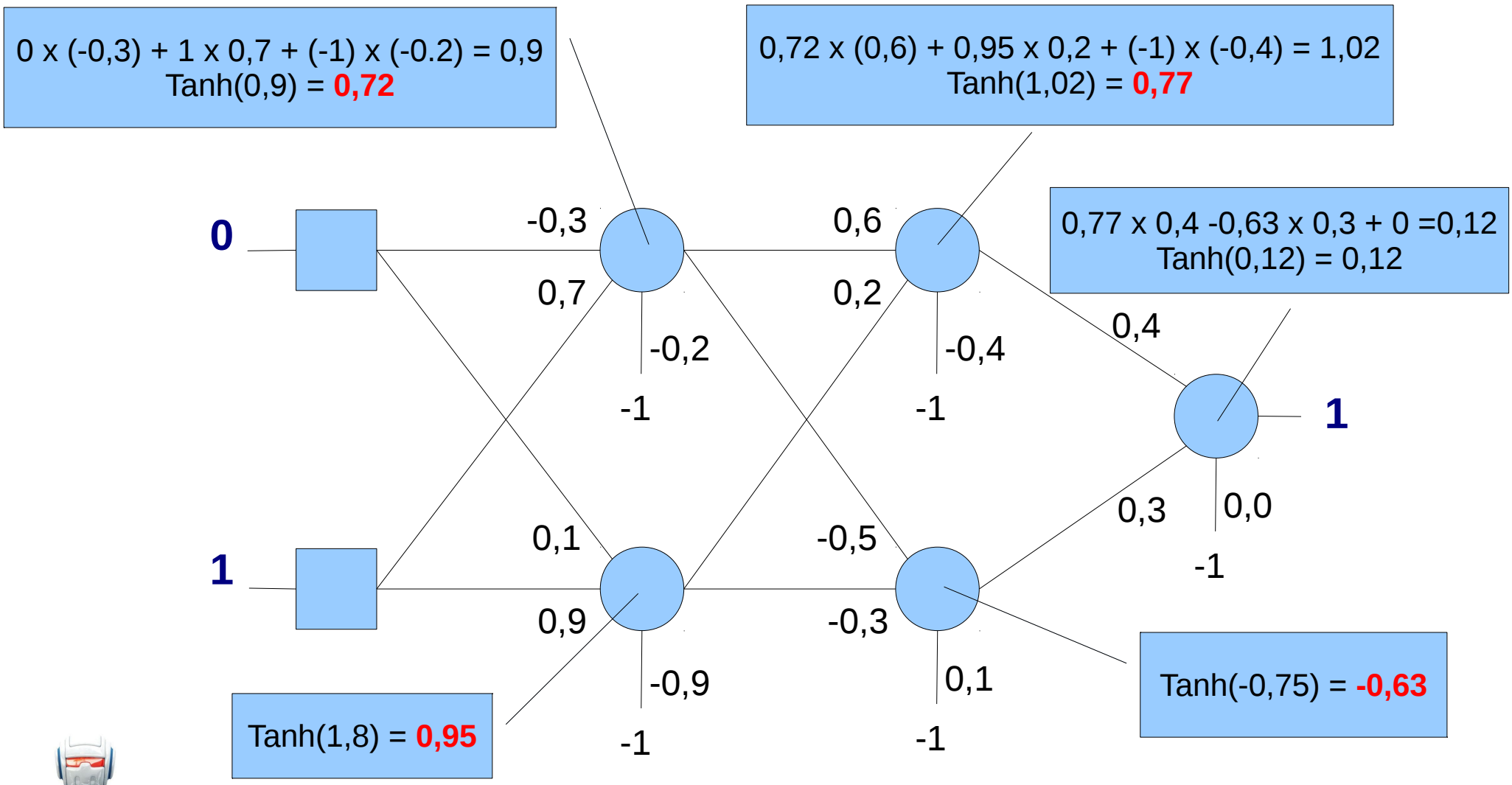


Tanh(1,8) = 0,95

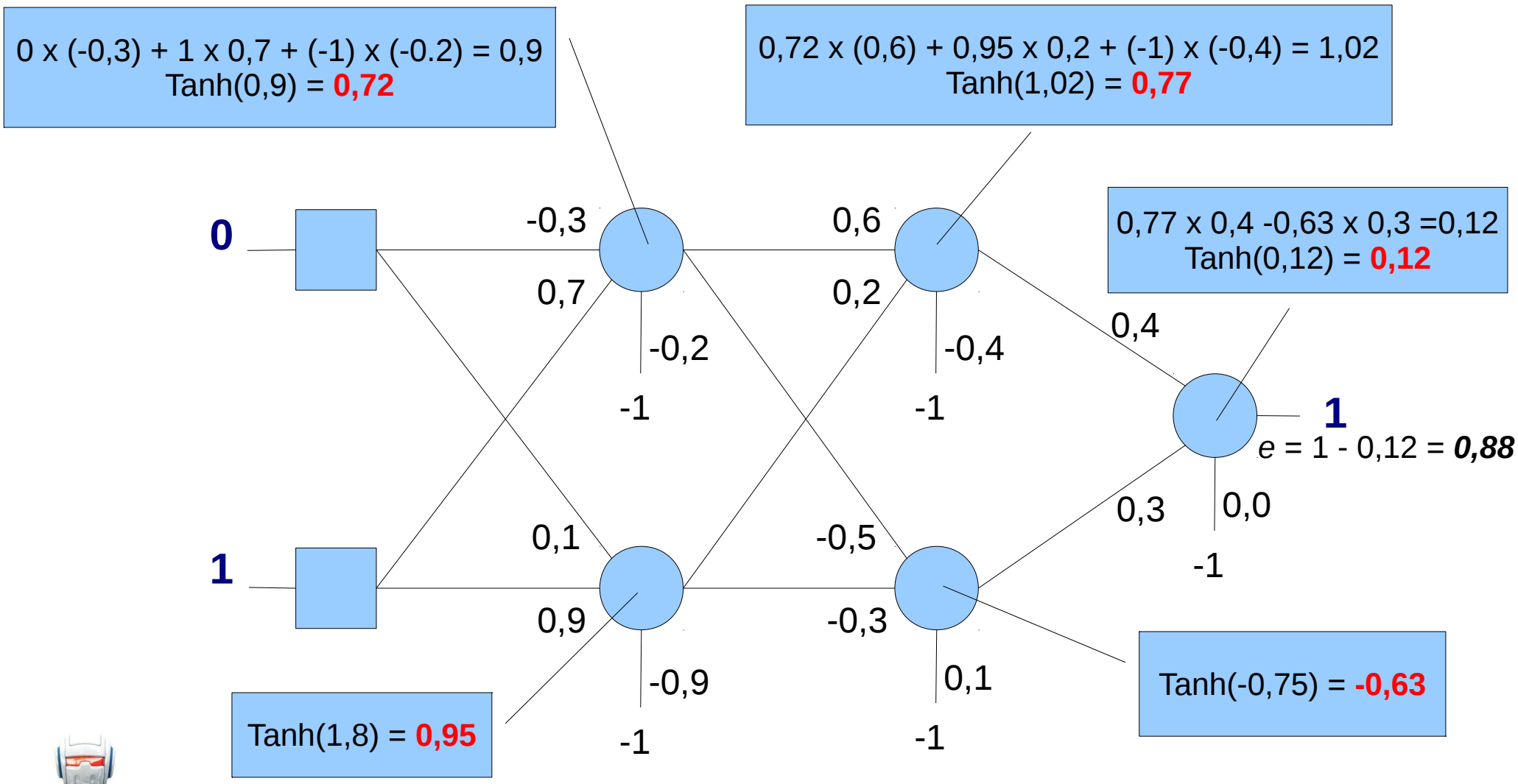
Backpropagation



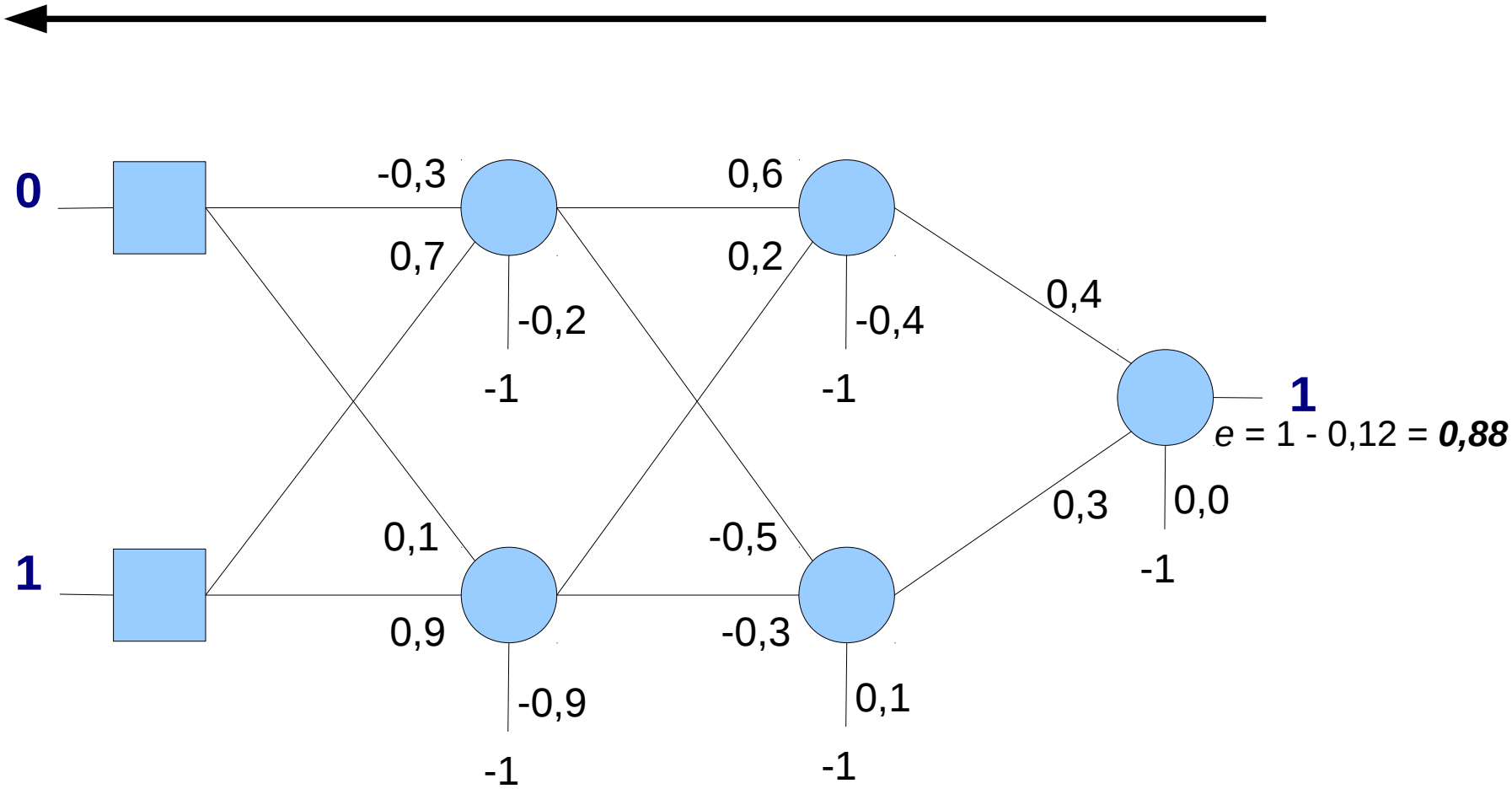
Backpropagation



Backpropagation

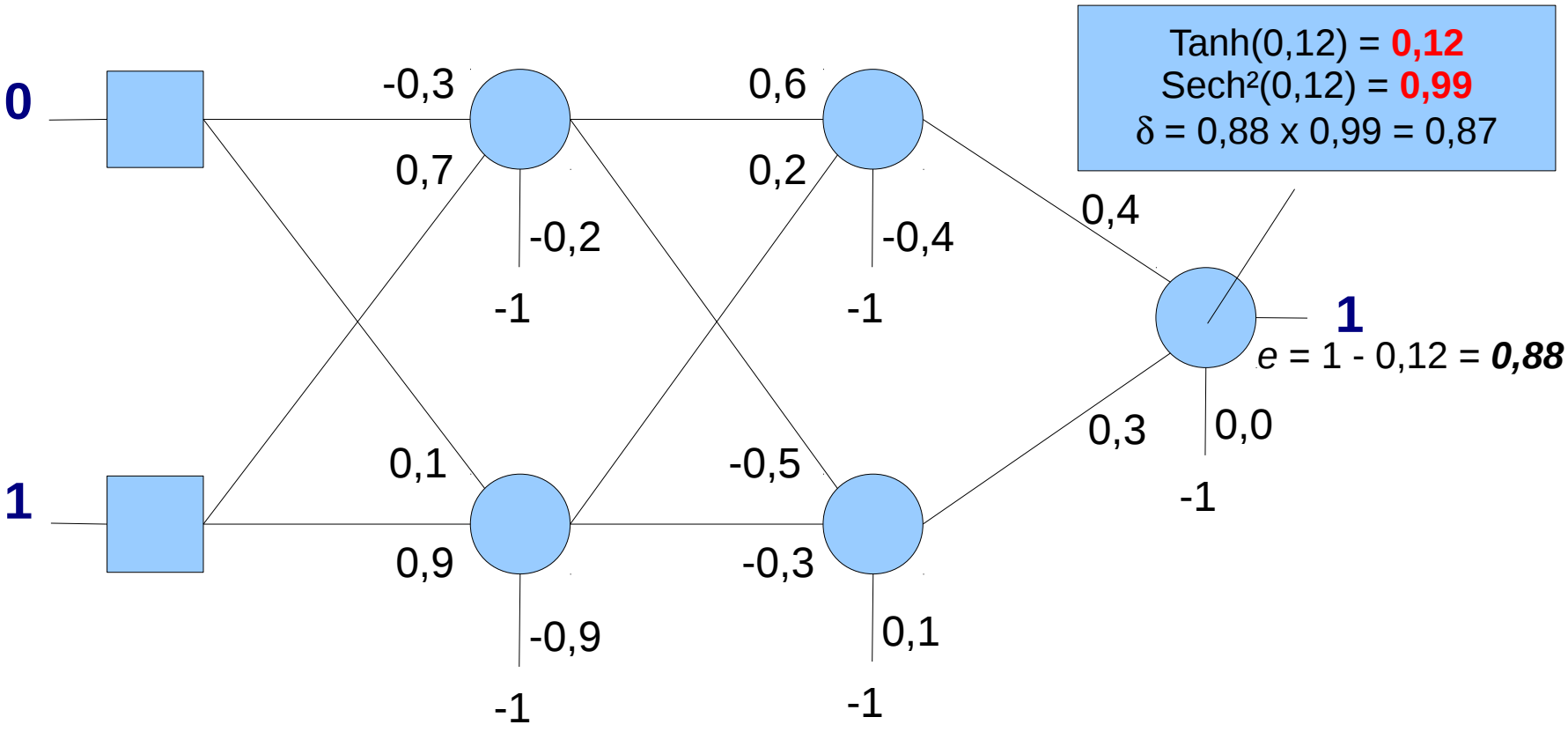


Backpropagation



$$\Delta w_{ji} = \eta \delta_j x_i$$
$$\delta_j = (d_j - y_j) f'(net_j)$$

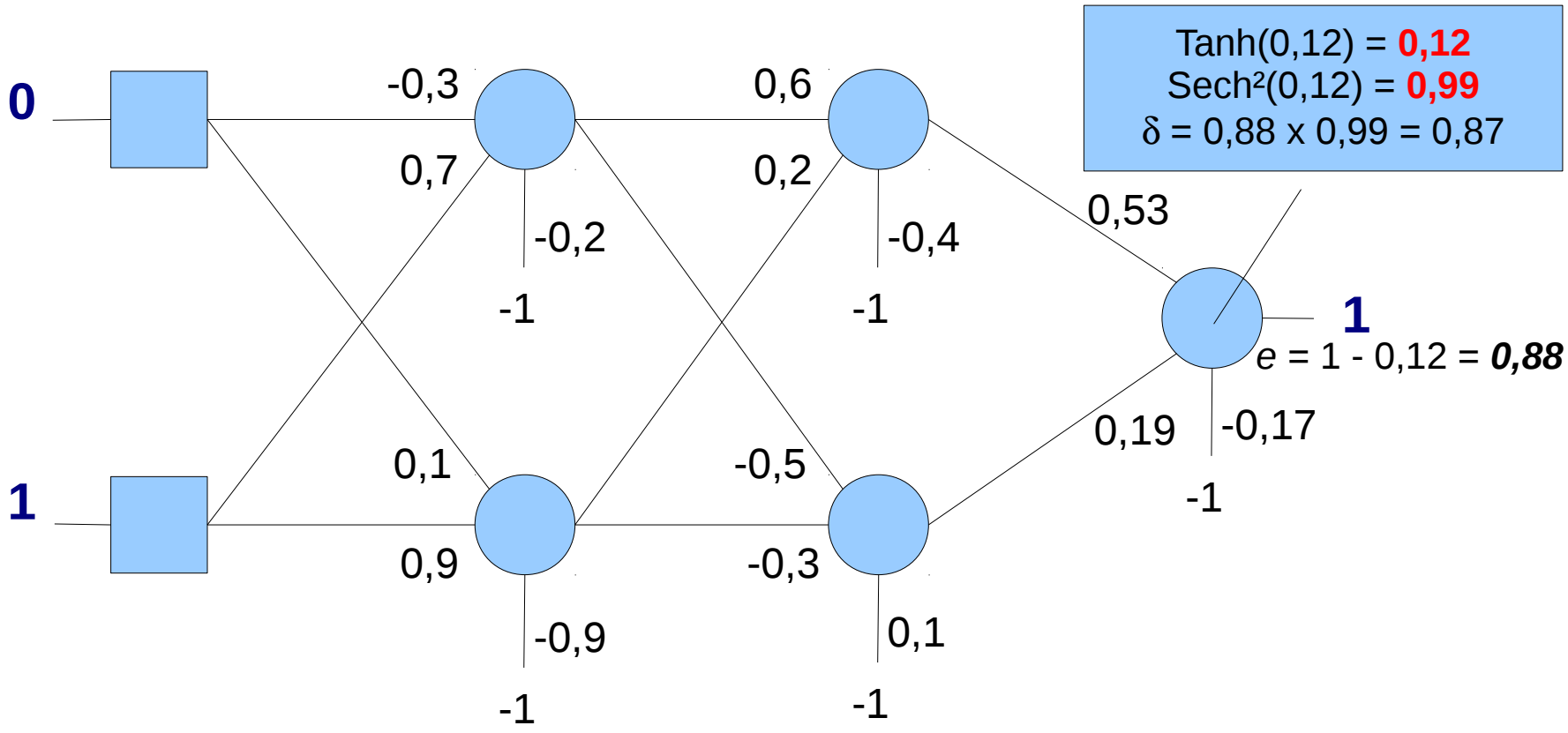
Backpropagation



$$\Delta w_{ji} = \eta \delta_j x_i$$
$$\delta_j = (d_j - y_j) f'(net_j)$$

Backpropagation

$\eta = 0,2$



$$\Delta w_{ji} = \eta \delta_j x_i$$
$$\delta_j = (d_j - y_j) f'(net_j)$$

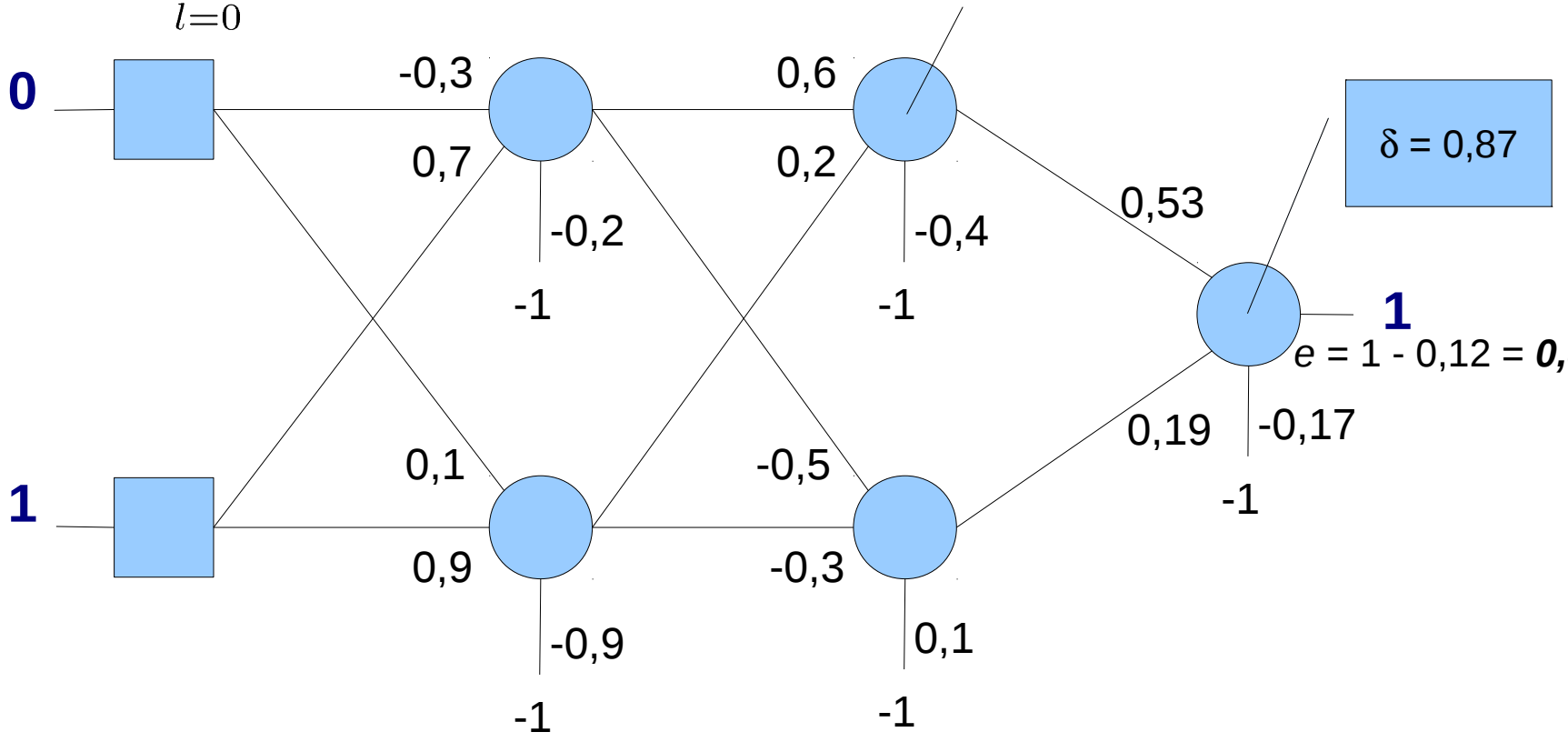
Backpropagation

$$\Delta w_{ji} = \eta \delta_j x_i$$

$$\delta_j = f'(net_j) \sum_{l=0}^{M} \delta_l w_{lj}$$

$\eta = 0,2$

Tanh(1,02) = **1,02**
Sech²(1,02) = **0,41**
δ = 0,41 x (0,87 x 0,53) = 0,19



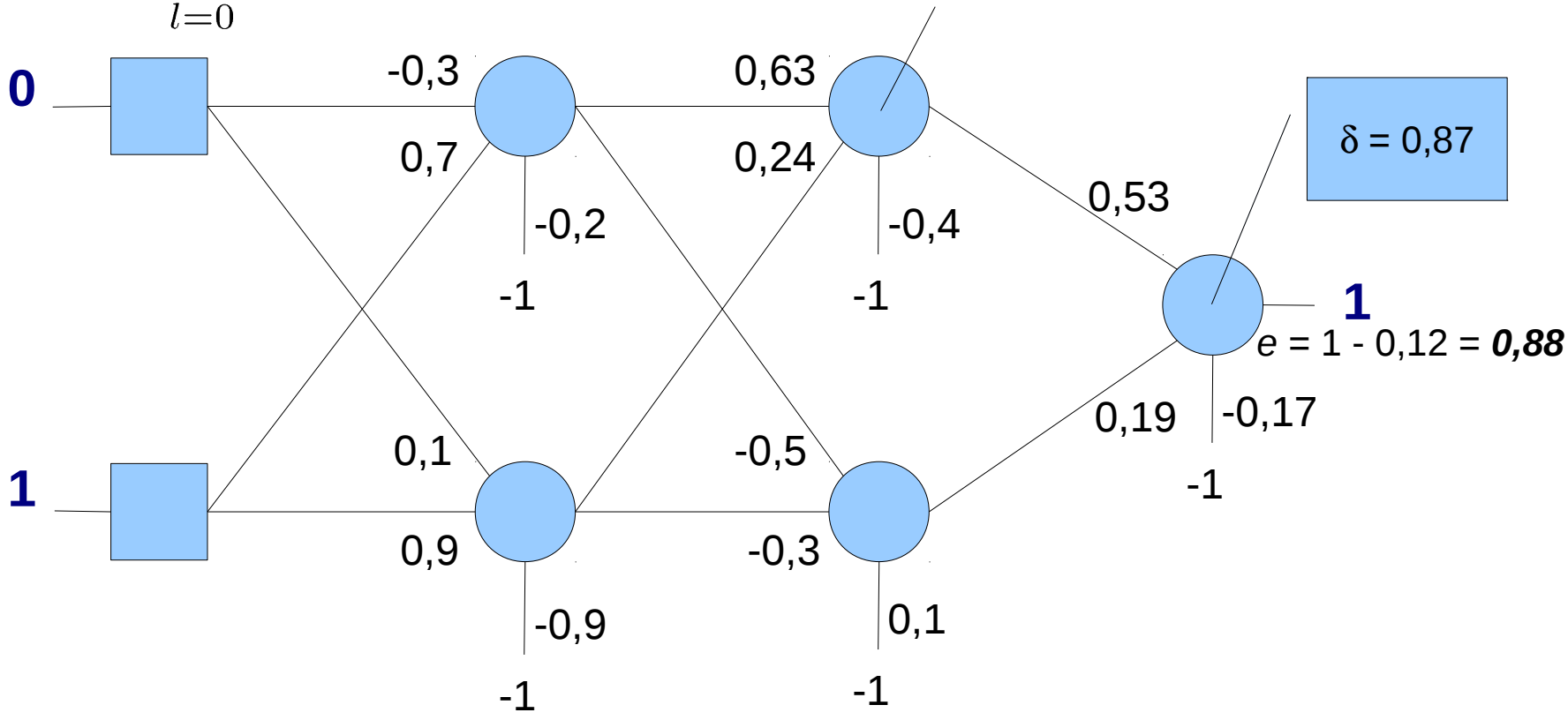
Backpropagation

$$\Delta w_{ji} = \eta \delta_j x_i$$

$$\delta_j = f'(net_j) \sum_{l=0}^{M} \delta_l w_{lj}$$

$\eta = 0,2$

Tanh(1,02) = **1,02**
Sech²(1,02) = **0,41**
δ = 0,41 x (0,87 x 0,53) = 0,19



Treinamento

- Amostras de Entrada
  - Entradas normalizadas ajudam o treinamento
    - Conjunto de Treinamento
      - Escolher aleatoriamente
    - Conjunto de Validação
      - Somatório dos Erros Quadráticos

$$E = \frac{1}{2}\sum_{j=1}^{k}(d_j - y_j)^2$$

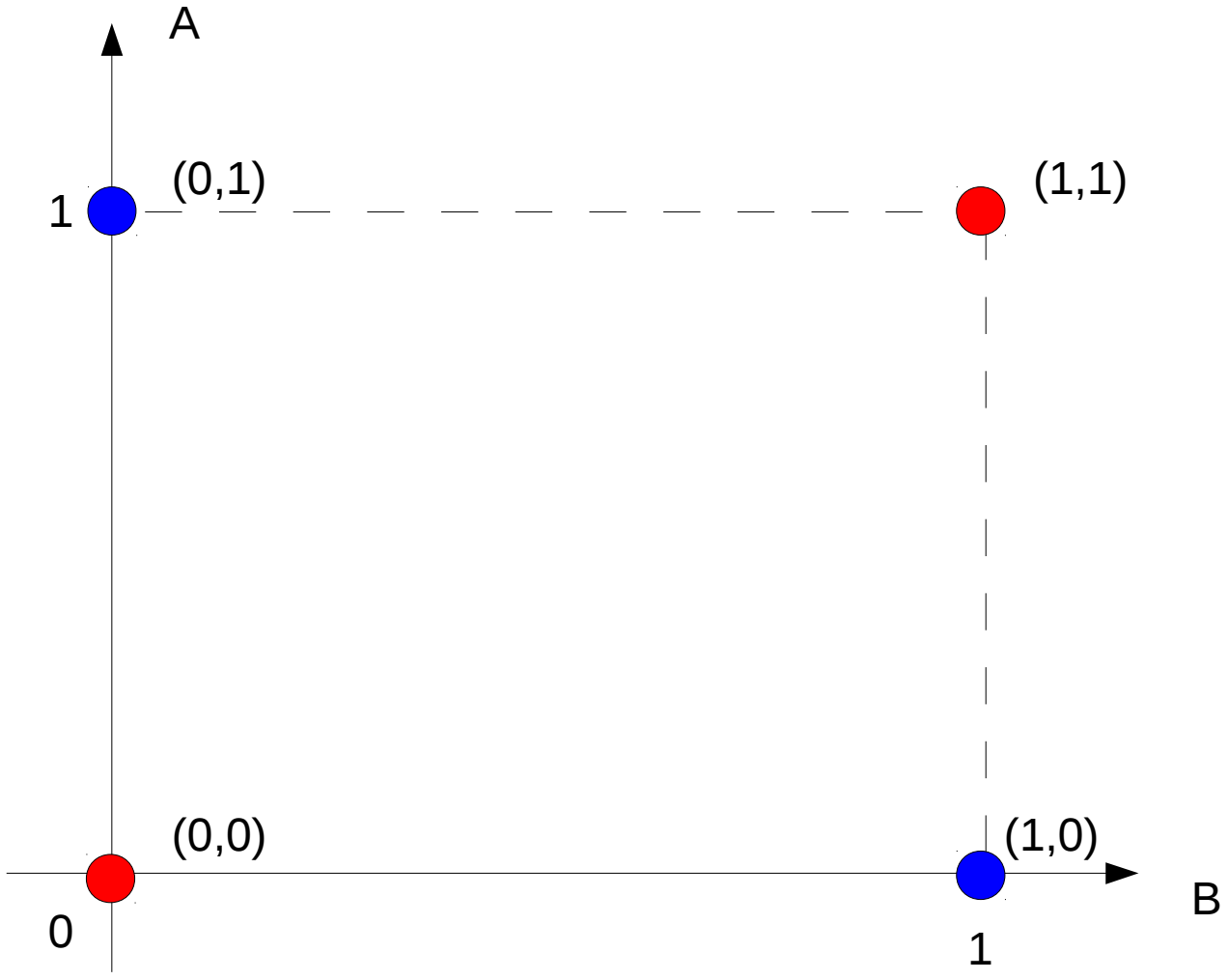
    - Conjunto de Teste

- Pesos Iniciais
  - Pesos Aleatórios
    - Para evitar mínimos locais quando for necessário reiniciar o treinamento

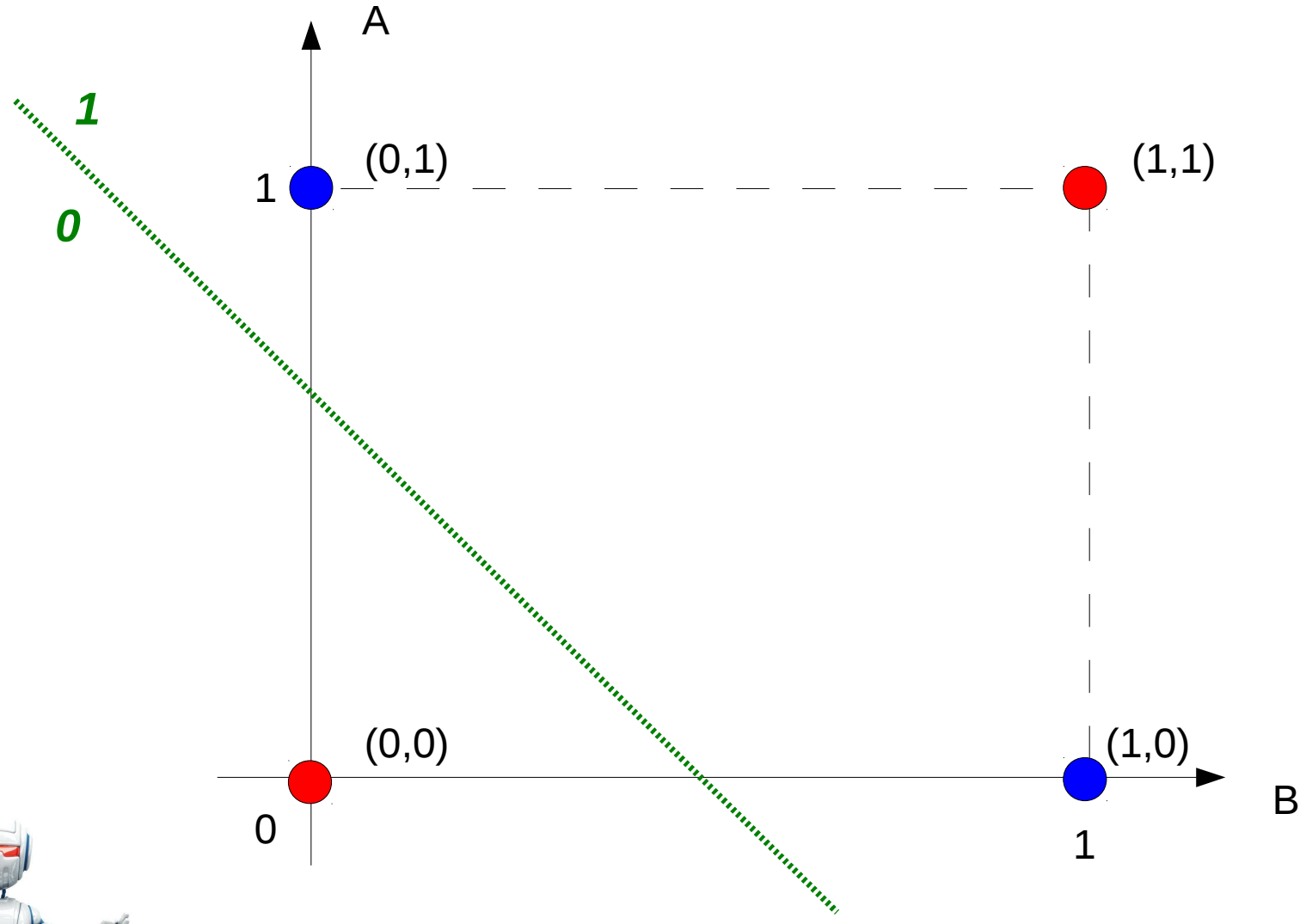
Aprendizado Ensemble (Combinado)

- Tanto o Perceptron quanto as redes MLP trabalham com uma única hipótese na classificação dos dados
  - No perceptron, as classes são separadas por um hiperplano
  - Nas MLPs, as classes são divididas em regiões abstratas
- É possível melhorar o desempenho das duas abordagens sem complicar muito o código
  - E o mesmo modelo pode ser empregado para outros métodos de classificação

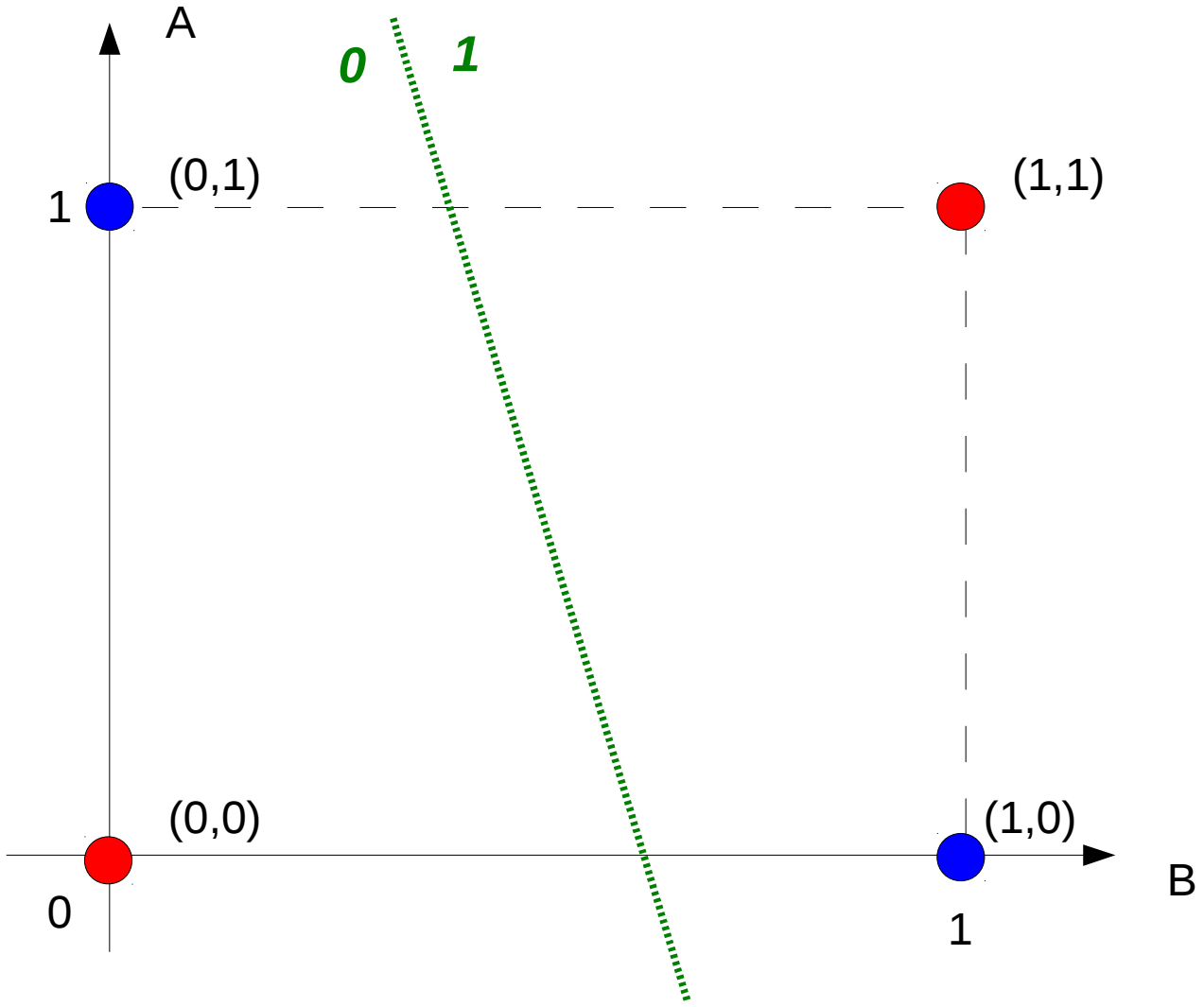
Perceptron e XOR



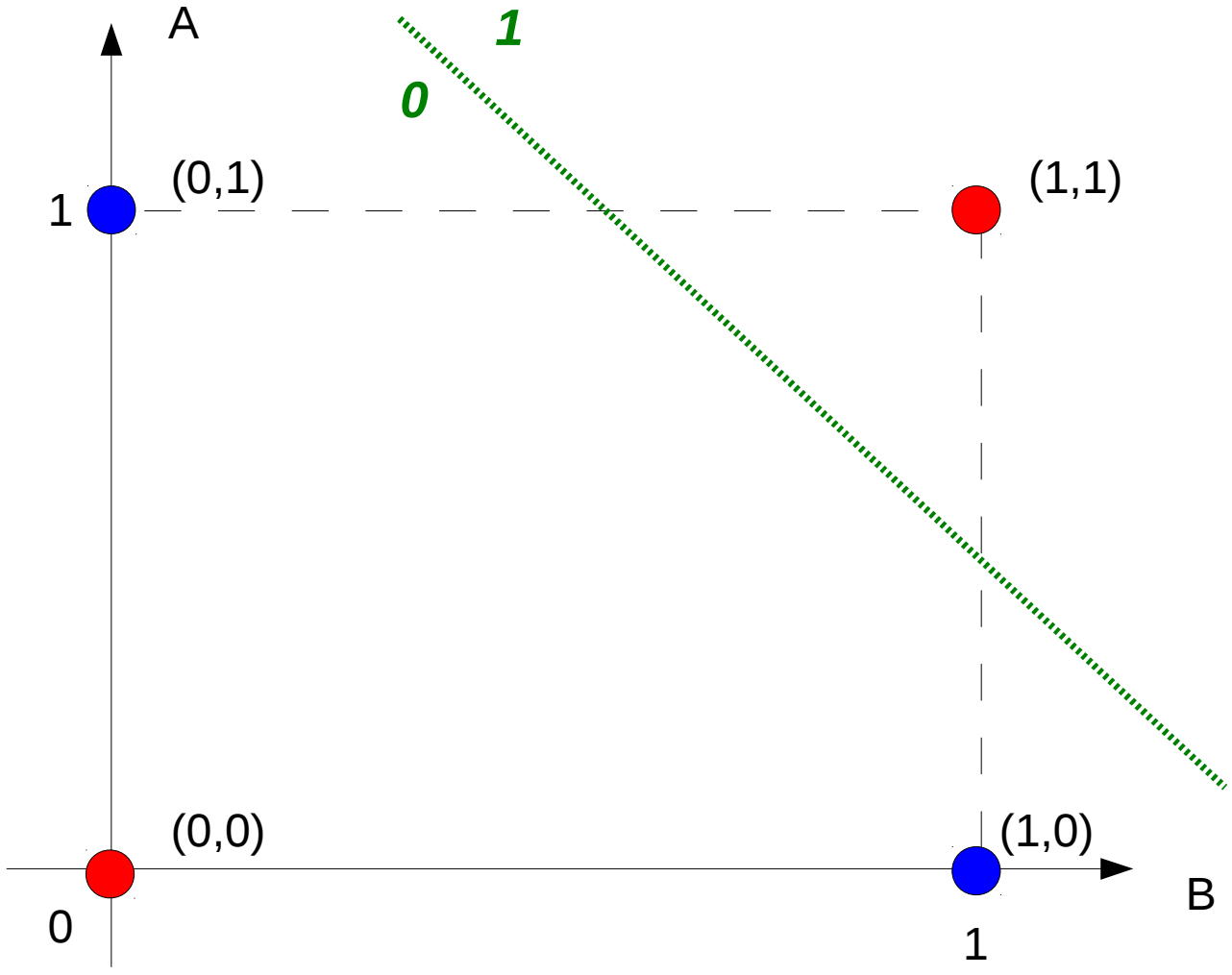
Perceptron e XOR



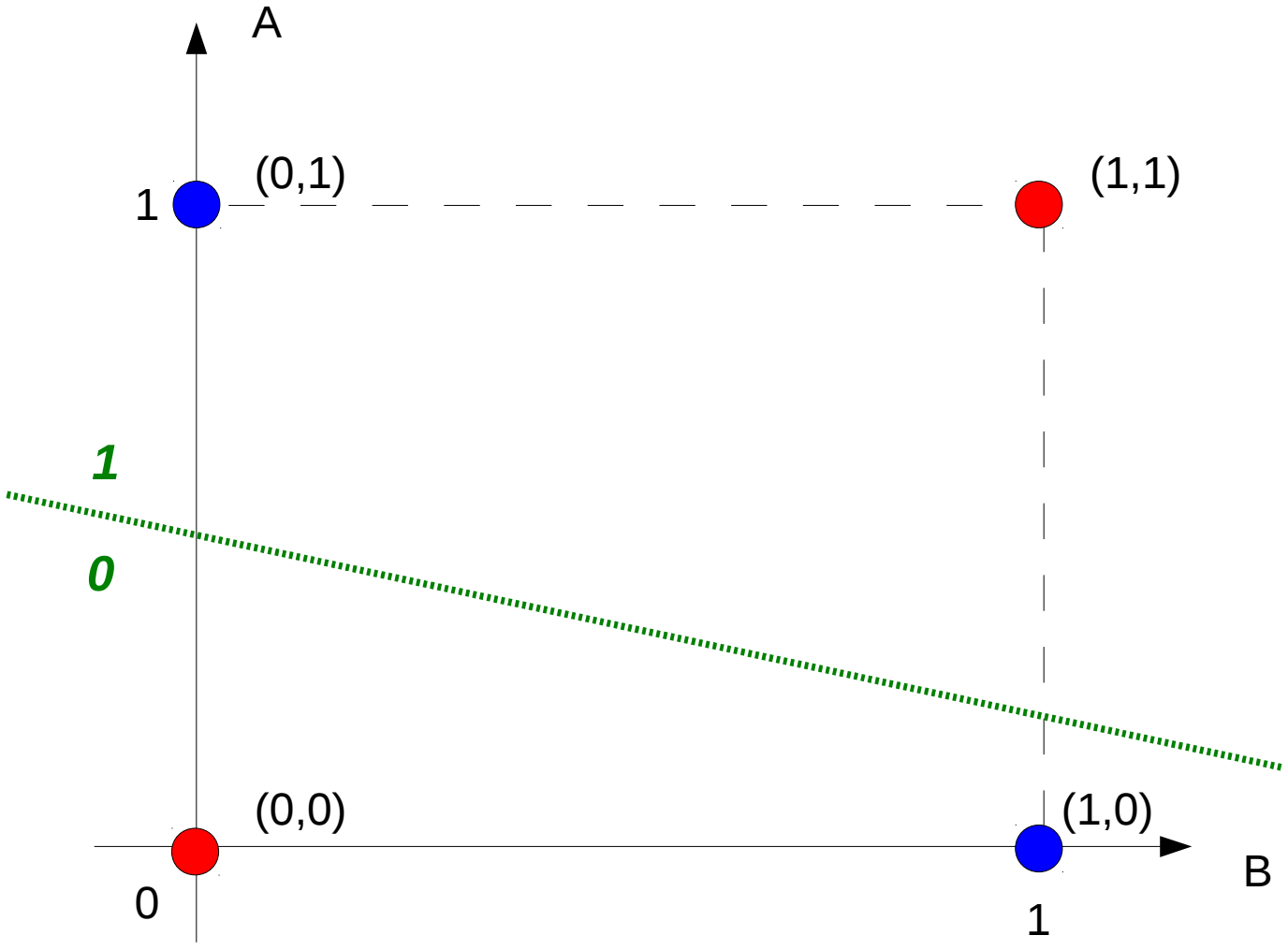
Perceptron e XOR



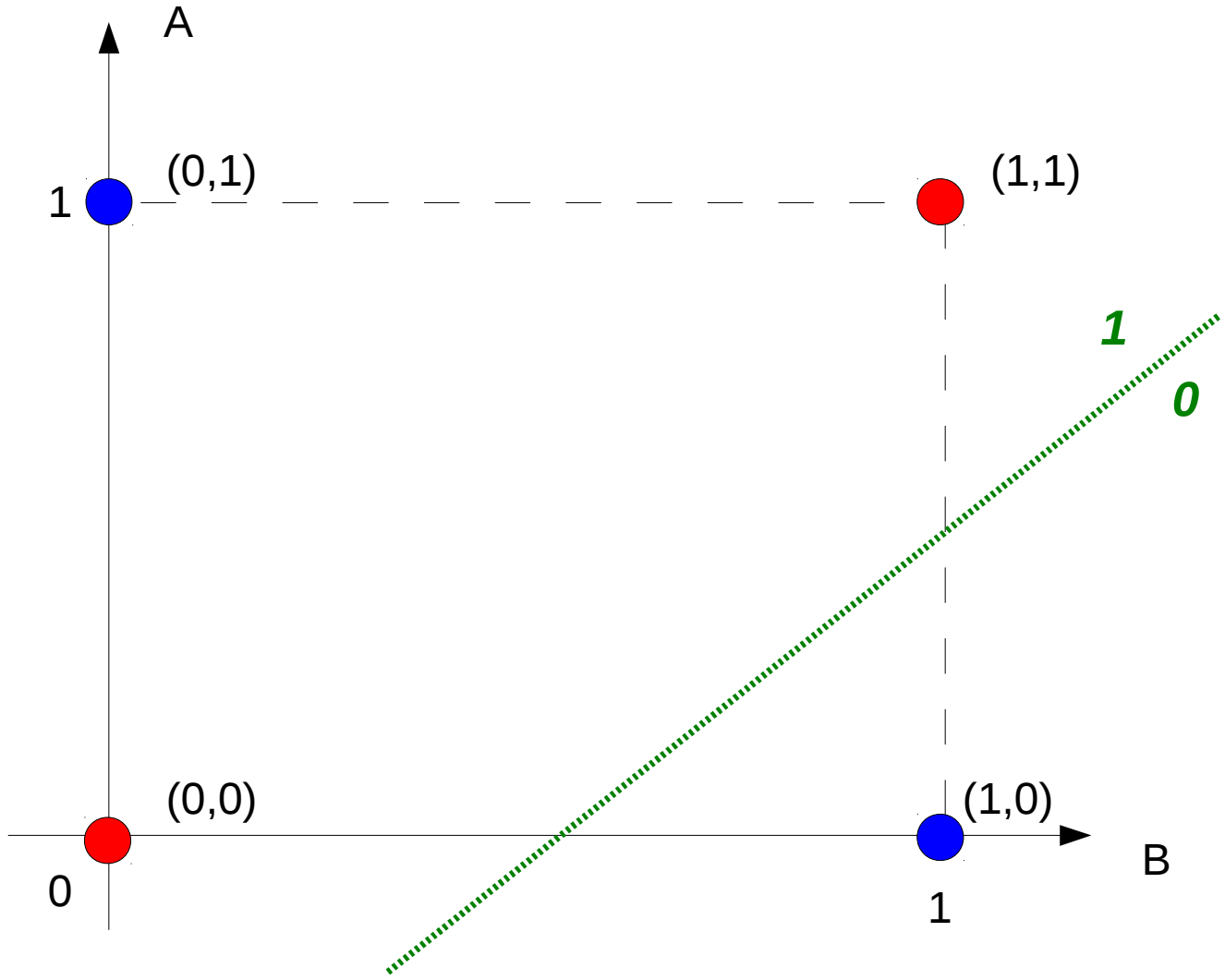
Perceptron e XOR



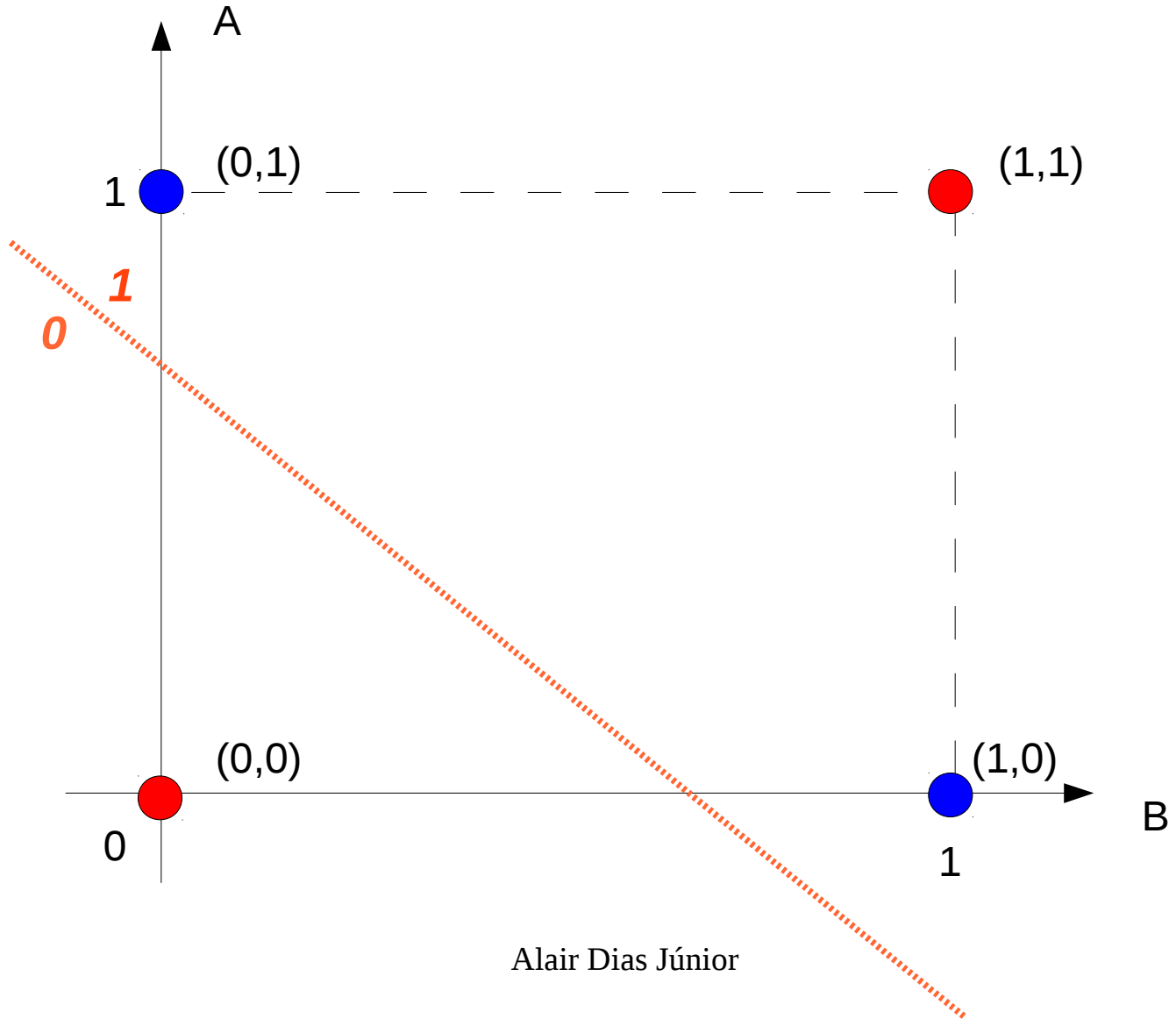
Perceptron e XOR



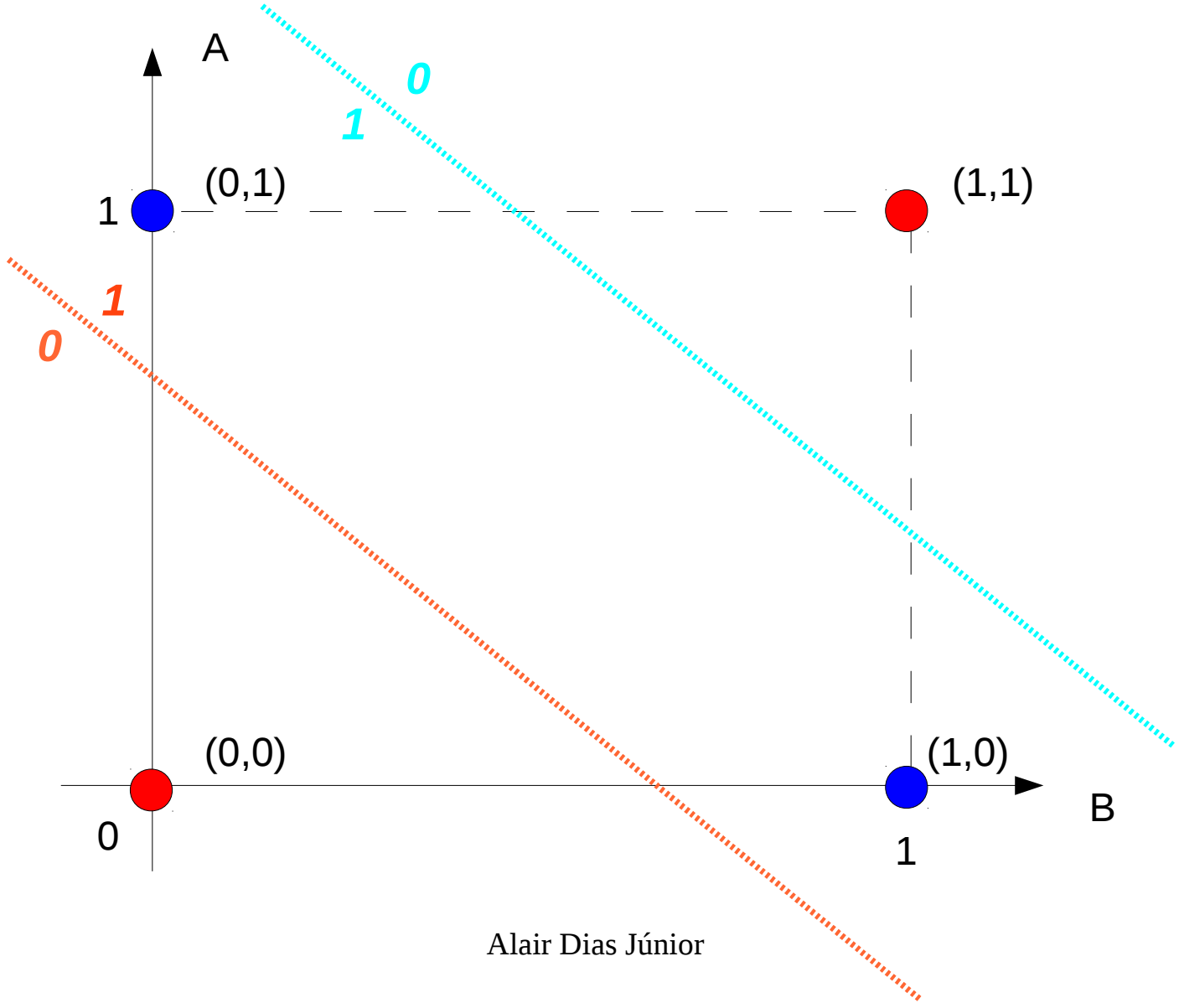
Perceptron e XOR



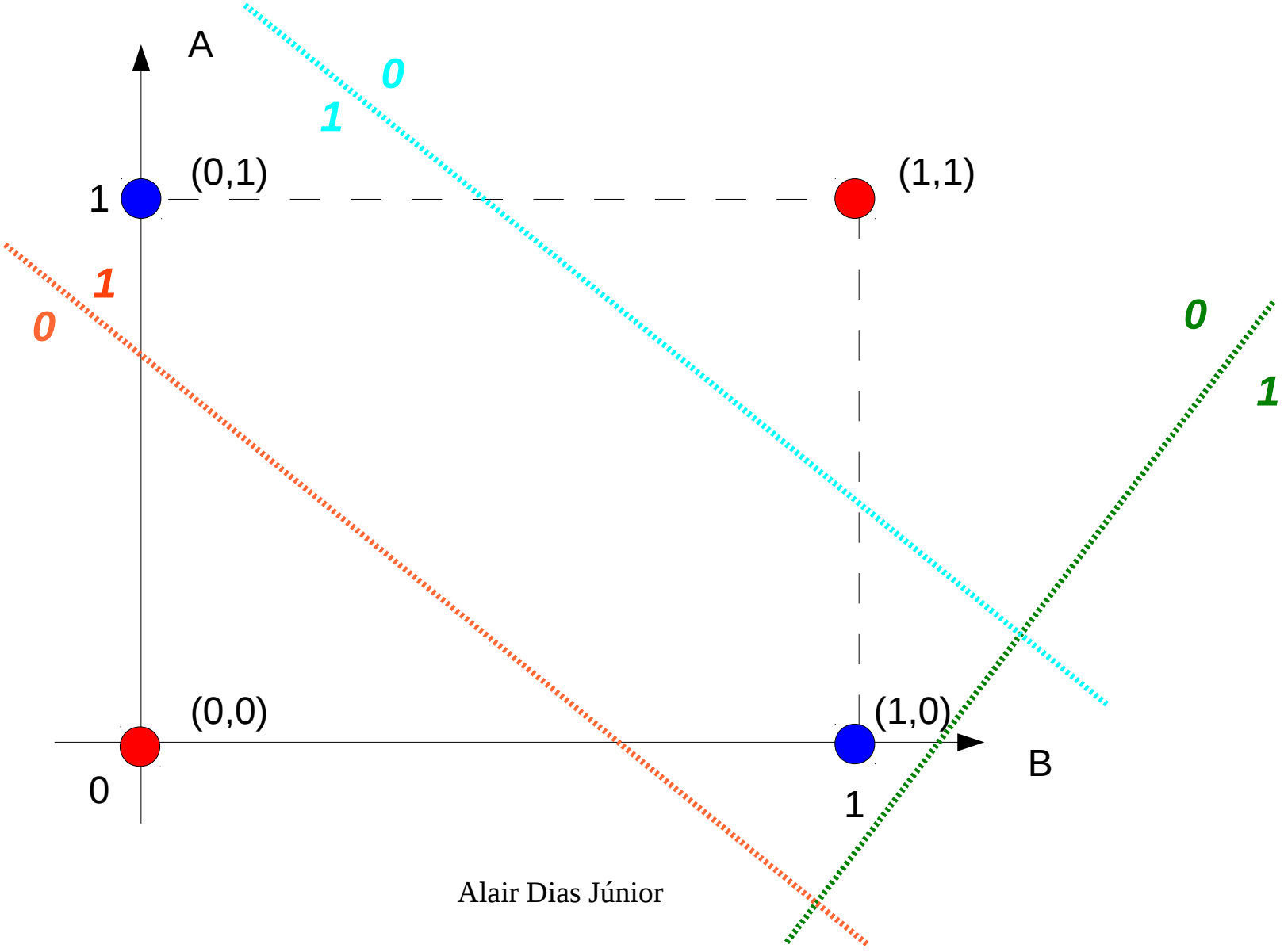
Combinando Perceptrons



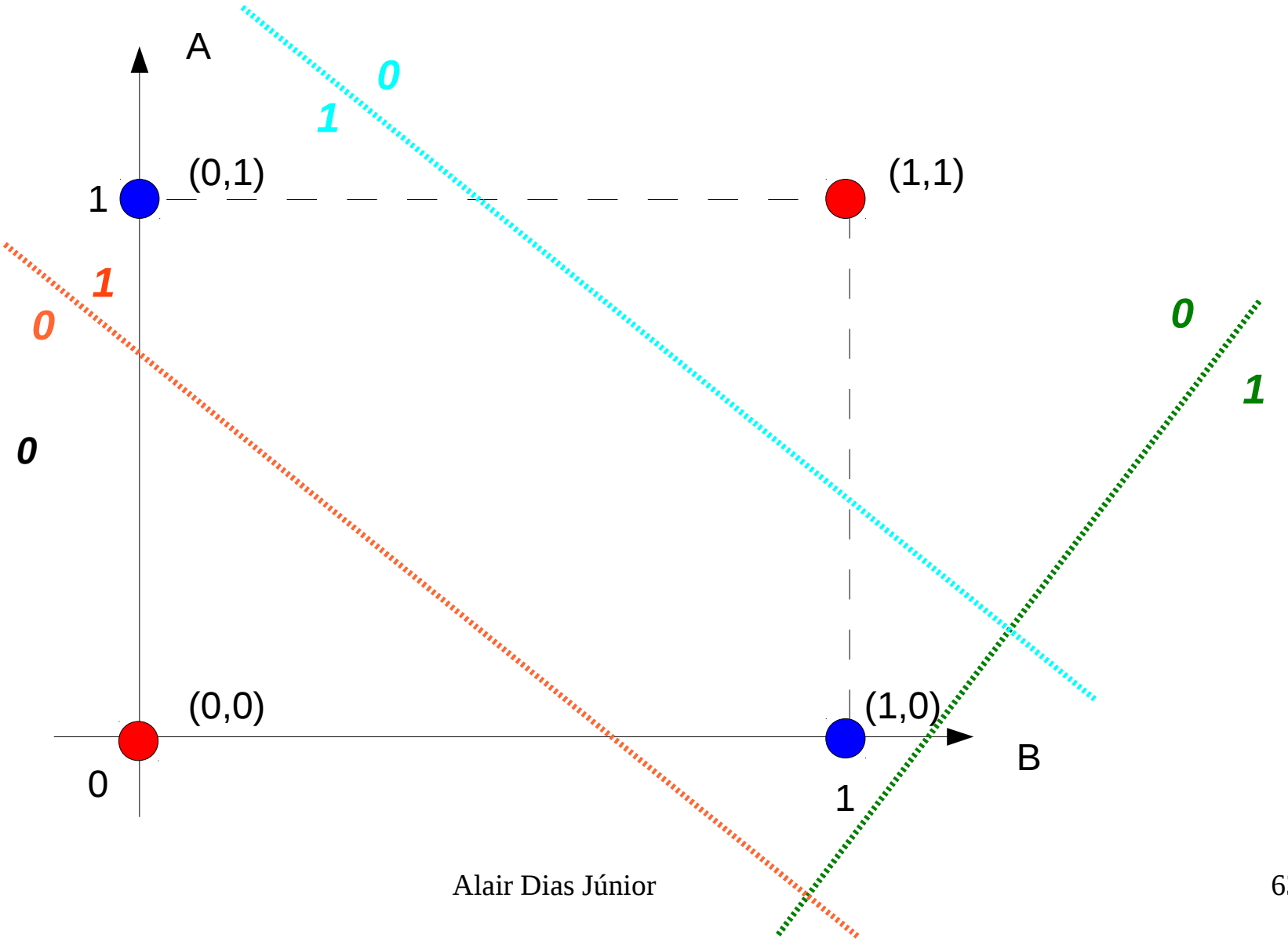
Combinando Perceptrons



Combinando Perceptrons

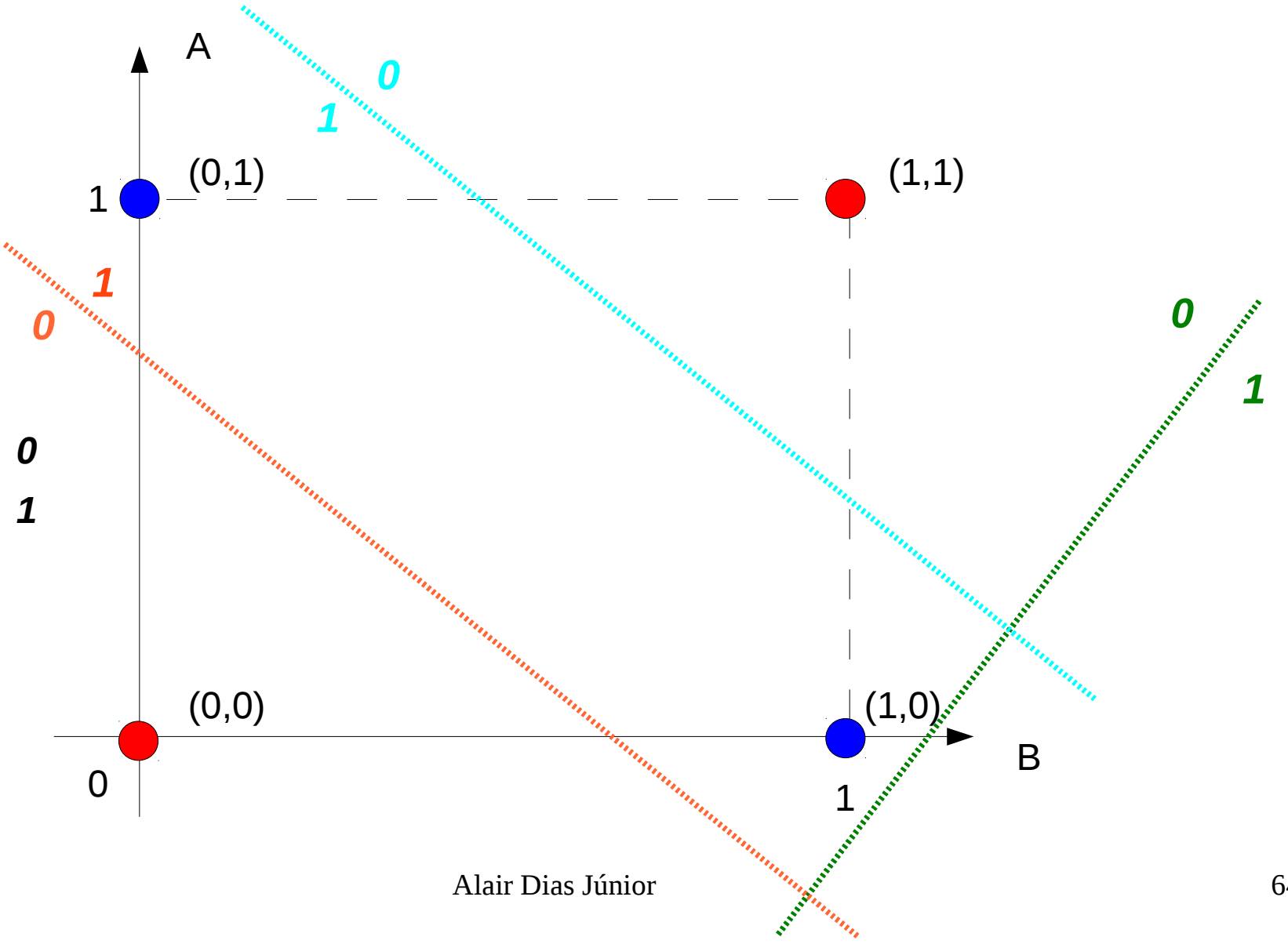


Combinando Perceptrons



(0,0) → 0,1,0 → *0*

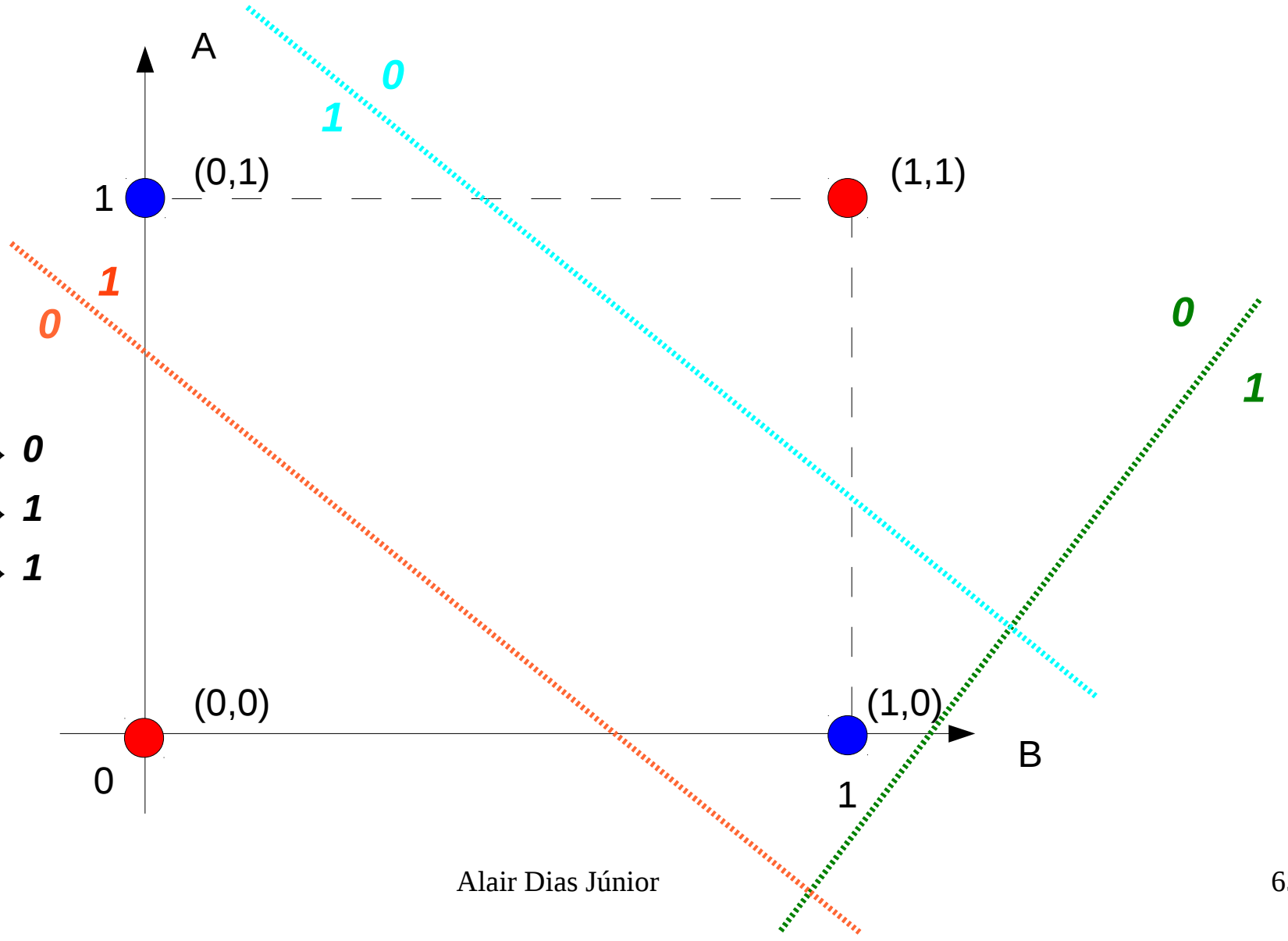
Combinando Perceptrons



(0,0) → 0,1,0 → *0*

(0,1) → 1,1,0 → *1*

Combinando Perceptrons

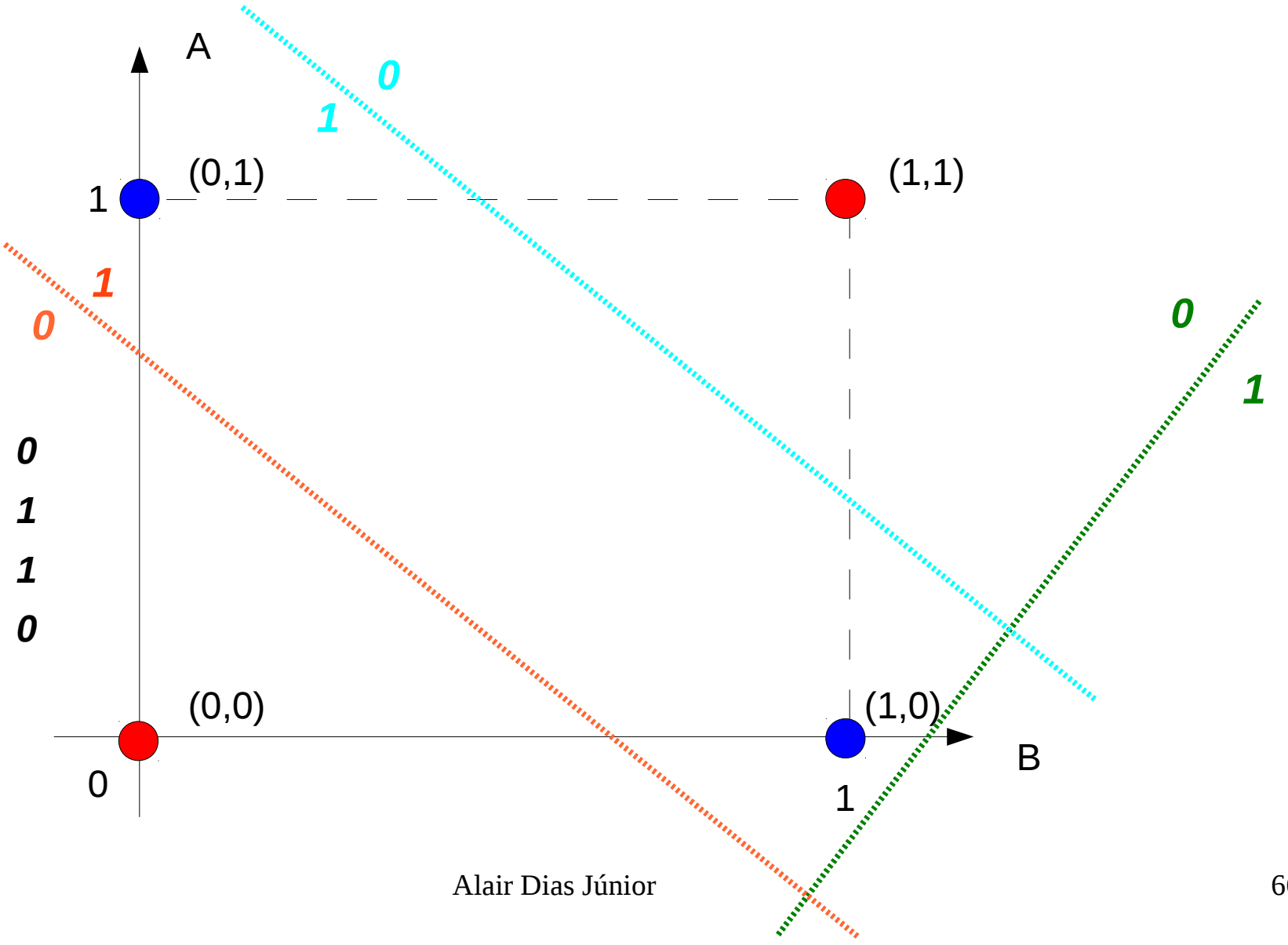


(0,0) → 0,1,0 → *0*

(0,1) → 1,1,0 → *1*

(1,0) → 1,1,0 → *1*

Combinando Perceptrons



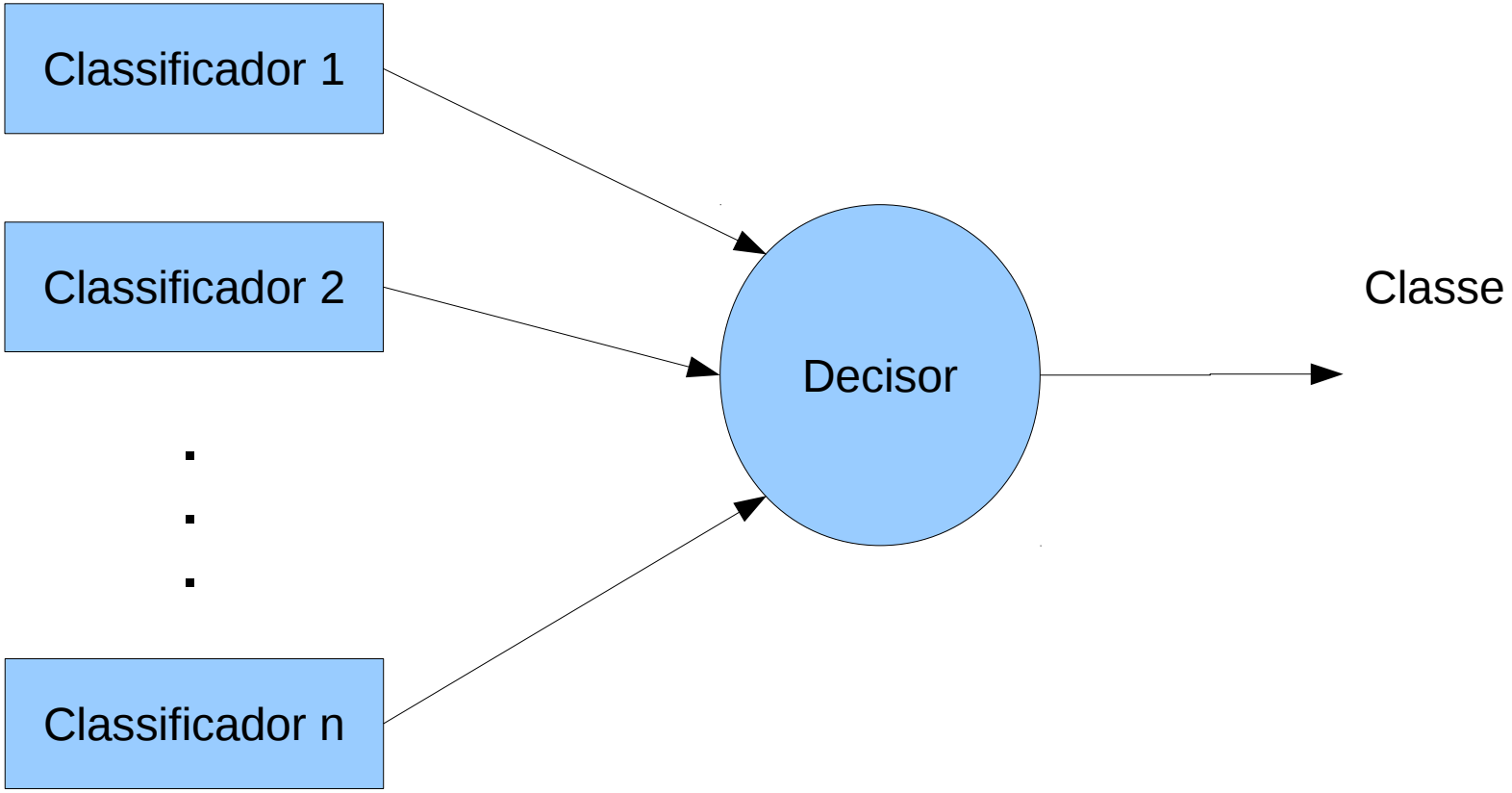
(0,0) → 0,1,0 → *0*

(0,1) → 1,1,0 → *1*

(1,0) → 1,1,0 → *1*

(1,1) → 1,0,0 → *0*

Modelo Ensemble



Modelo Ensemble

- Combinando vários classificadores, mesmo que simples, o resultado é melhorado
  - Quanto mais classificadores, menor a probabilidade de erro
- Pode-se utilizar classificadores de tipos diferentes
  - Combinar Perceptron, com MLP, com distância euclidiana, etc...

Como treinar um Modelo Ensemble

- Treinar um modelo ensemble com classificadores do mesmo tipo, requer cuidados
  - Se o mesmo conjunto de dados for apresentado, da mesma forma, para vários perceptrons, por exemplo, todos eles apresentarão a mesma classificação

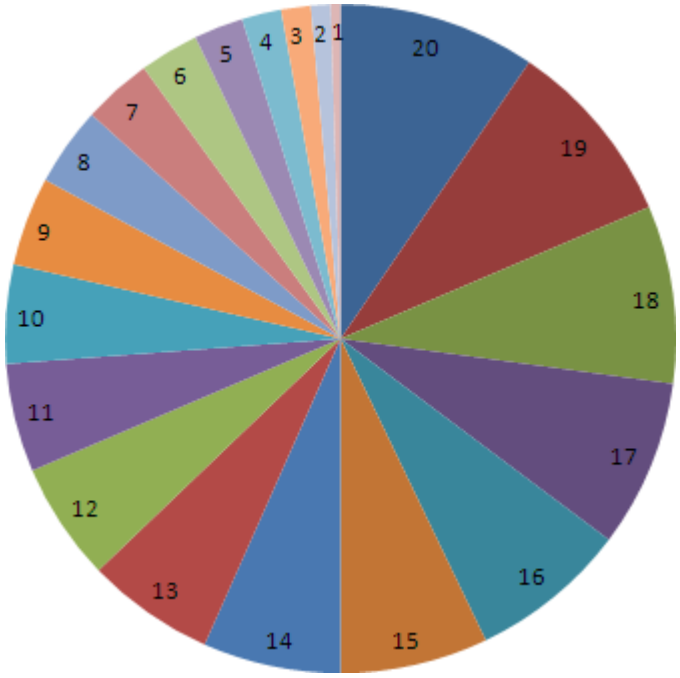
- O treinamento de cada um dos classificadores do modelo deve ser diferente
  - Geralmente emprega-se uma técnica chamada de ***boosting***

Boosting

- Os exemplos do conjunto de treinamento são apresentados para os classificadors aleatoriamente, de acordo com uma pro-babilidade
  - Para o primeiro classificador, esta probabilidade é igual para todos os exemplos do conjunto de treinamento

Boosting

- Depois que o primeiro classificador estiver treinado, os exemplos para os quais ele errou a classificação têm sua probabilidade aumentada (*boosting*)



Boosting

- O próximo classificador utiliza as novas probabilidades para definir quais exemplos serão treinados
  - Ao fim do treinamento, os elementos acertados por este classificador têm a probabilidade reduzida
    - Os que foram classificados erroneamente, têm a probabilidade aumentada novamente
- Este processo se repete até que todos os classificadores foram treinados

Aproximação de Funções

- Para o caso de aproximação de funções, pode ser interessante que a rede seja capaz de atingir valores mais altos na saída (menores que -1 e maiores que 1).
  - Com a função tanh isto não é possível
  - Solução: Um neurônio com função de ativação linear na saída
    - Utilizar este tipo de função somente na camada de saída não prejudica a capacidade de aproximação da rede