

- Objetivo:
 - Fornecer ao aluno, conceitos básicos e características de modelos conexionistas, apresentando os principais modelos, bem como, as diversas áreas em que esses modelos podem ser utilizados

Universidade de São Paulo

Instituto de Ciências Matemáticas e de Computação

Departamento de Ciências de Computação

Rodrigo Fernandes de Mello

mello@icmc.usp.br

Sobre a Disciplina

Referências

- Ementa:
 - Definição de modelos conexionistas
 - Aprendizado em Modelos Conexionistas
 - Arquiteturas básicas:
 - Perceptron
 - Perceptron Multi-Camadas
 - Redes RBF
 - Memórias Associativas:
 - BAM
 - Redes de Hopfield
 - Rede de Kohonen
- Aplicações
- Freeman and Skapura, Neural Networks: Algorithms, Applications and Programming Techniques, Addison-Wesley, 1991
- HAYKIN, S. "Neural Networks. A Comprehensive Foundation", New Jersey, Prentice Hall, 2 ed., 1999
- BRAGA,A.; CARVALHO, A.; LUDERMIR, T. "Redes Neurais Artificiais: Teoria e Aplicações", Livro Técnico e Científico, Rio de Janeiro, 2000

Método de Avaliação

Atendimento

- Datas de provas:
 - P1 → 02/10
 - P2 → 04/12
- 2 Provas e Trabalhos:
- Com Professor:
 - Sextas-feiras
 - 15:30 hs às 18:30 hs
 - Sala 3-242
 - mello@icmc.usp.br

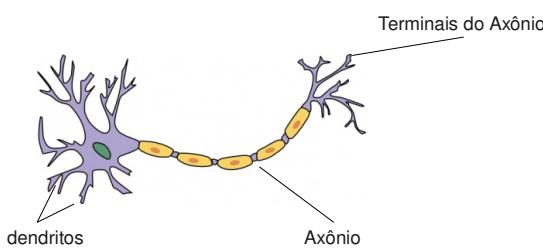
$$M = \frac{7}{\frac{2}{p_1} + \frac{4}{p_2} + \frac{1}{MT}}$$

- Média de Trabalhos (MT) é aritmética simples

- Presenças:
 - Mínimo de 70%
 - Sobre Atestados
- Evitar atrasos
- Evitar conversas em excesso
- Perguntem!

Redes Neurais Artificiais

- Baseada no conceito de neurônios biológicos
- Códigos são programados para mimetizar o comportamento de neurônios biológicos
- Conexões sinápticas encaminham sinais vindos dos dendritos em direção ao axônio

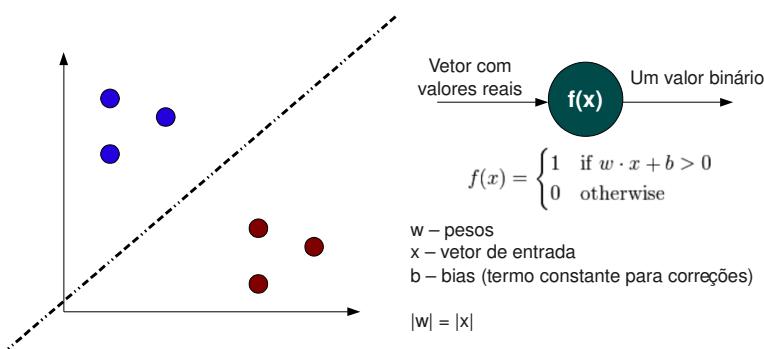


Redes Neurais Artificiais: Histórico

- McCullouch e Pitts (1943) propuseram um modelo computacional baseado em redes neurais biológicas
 - Modelo denominado Threshold logic
- Hebb (década de 1940), psicólogo, propôs a hipótese de aprendizado baseado no mecanismo de plasticidade neural
 - Plasticidade neural
 - Capacidade do cérebro em se remodelar em função de experiências do sujeito
 - Reformulação de conexões em função das necessidades e fatores do meio ambiente
 - Deu origem ao Aprendizado Hebbiano (emprego na Computação à partir de 1948)

Redes Neurais Artificiais: Histórico

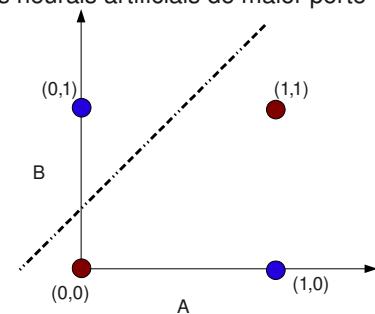
- Rosenblatt (1958) propôs o modelo Perceptron
 - Um classificador linear e binário



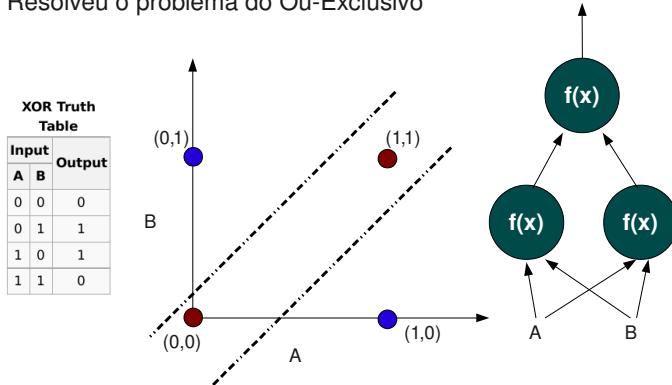
Redes Neurais Artificiais: Histórico

- Após a publicação de Minsky e Papert (1969) a área ficou stagnada, pois descobriram:
 - Que problemas como Ou-Exclusivo não poderiam ser resolvidos utilizando Perceptron
 - Computadores não tinham capacidade suficiente para processar redes neurais artificiais de maior porte

XOR Truth Table		
Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0



- Reinício das pesquisas na área após a proposta do algoritmo Backpropagation (Webos 1975)
 - Resolveu o problema do Ou-Exclusivo



- Em meados de 1980 a área de processamento paralelo e distribuído surge com o nome de conexionismo
 - Devido a seu uso para implementar Redes Neurais Artificiais
- “Redescoberta” do algoritmo Backpropagation por meio do artigo “Learning Internal Representations by Error Propagation” (1986)
 - Motivou adoção e popularizou uso

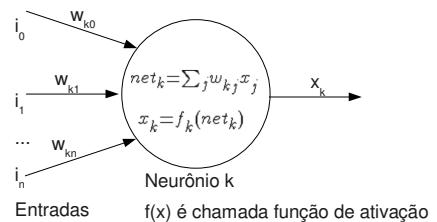
Redes Neurais Artificiais

Elemento de Processamento Geral

- Aplicações:
 - Reconhecimento de Fala
 - Classificação de Imagens
 - Identificação de portadores de doenças
 - AML, ALL, etc.
 - Agentes de Software
 - Video games
 - Robôs autônomos

Neurônios artificiais

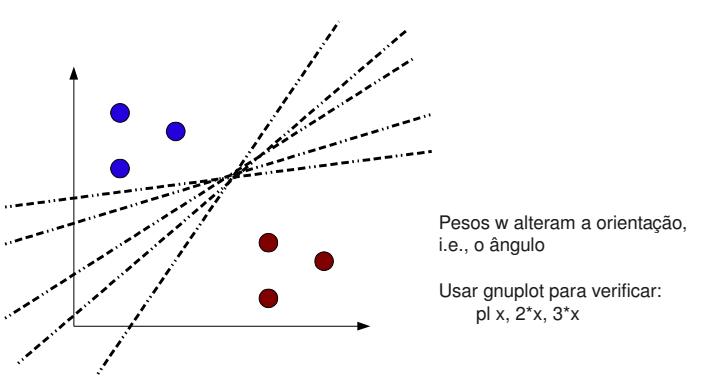
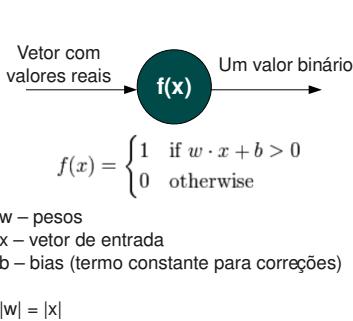
- Nós, unidades ou elementos de processamento
- Pode receber várias entradas, mas somente produz uma saída
- Cada conexão está associada a um peso w (força de conexão)
- O aprendizado ocorre com a adaptação dos pesos w



O Perceptron

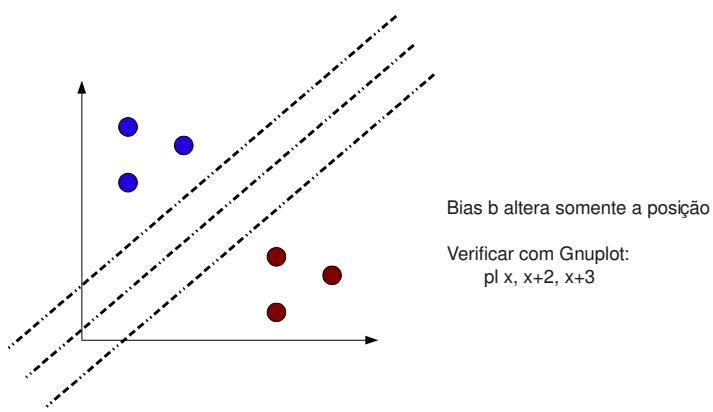
O Perceptron

- Psicólogo Rosenblatt (1958) propôs o Perceptron
 - Um classificador linear e binário



O Perceptron

O Perceptron



- Algoritmo de aprendizado para o Perceptron não termina se dados não são linearmente separáveis
- Parâmetros do Algoritmo:
 - $y = f(i)$ é a saída do perceptron para um vetor de entrada i
 - b é o termo de bias
 - $D = \{(x_1, d_1), \dots, (x_s, d_s)\}$ representa o conjunto de treinamento com s exemplos, em que:
 - X_i é o vetor de entrada com n dimensões
 - d_i é a saída esperada pelo perceptron
 - x_{ji} é o valor do neurônio i para um vetor de entrada j
 - w_i é o valor do peso i que será multiplicado pelo i -ésimo valor do vetor de entrada
 - α é a taxa de aprendizado (0,1]
 - Altas taxas de aprendizado fazem com que o perceptron oscile em torno da solução

O Perceptron

O Perceptron

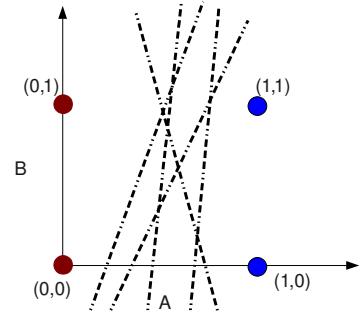
- Algoritmo
 - Iniciarizar os pesos w de forma aleatória
 - Para cada par j do conjunto de treinamento D
 - Calcular a saída
$$y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j] = f[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \dots + w_n(t)x_{j,n}]$$

 - Adaptar os pesos

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}, \text{ for all nodes } 0 \leq i \leq n.$$
- Continuar até o erro ser menor que um dado threshold (limiar) ou por um número prédefinido de vezes
$$d_j - y_j(t) < \gamma$$
- Adaptação:
 - Pode-se, também, observar todos exemplos de treinamento antes de verificar limiar

- Função de Transferência para o Perceptron
 - Função degrau
 - Verificar no Gnuplot:
 - $f(x) = (x > 0.5) ? 1 : 0$
 - $pl f(x)$
- Implementação
 - Exemplo de função NAND

INPUT	OUTPUT	
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0



O Perceptron

O Perceptron

- Implementação
 - NAND
 - Verificar os pesos da rede e plotar usando Gnuplot
 - Como são duas dimensões de entrada, deve-se plotar usando o comando "spl" considerando os eixos x e y
 - Plotar separação em função dos pesos finais

```
gnuplot> set border 4095 front linetype -1 linewidth 1.000
gnuplot> set view map
gnuplot> set isosamples 100, 100
gnuplot> unset surface
gnuplot> set style data pm3d
gnuplot> set style function pm3d
gnuplot> set ticslevel 0
gnuplot> set title "gray map"
gnuplot> set xlabel "x"
gnuplot> set xrange [-15.0000 : 15.0000 ] noreverse nowriteback
gnuplot> set ylabel "y"
gnuplot> set xrange [-15.0000 : 15.0000 ] noreverse nowriteback
gnuplot> set xrange [-0.250000 : 1.00000 ] noreverse nowriteback
gnuplot> set pm3d implicit at b
gnuplot> set palette positive nops_allcF maxcolors 0 gamma 1.5 gray
gnuplot> set xr [0:1]
gnuplot> set yr [0:1]
gnuplot> spl 1.0290568822825088+0.15481468877189009*x+0.46986458608516524*y
```

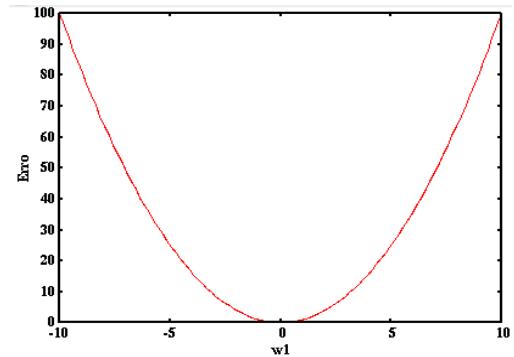
- Analisar a adaptação de pesos

$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}, \text{ for all nodes } 0 \leq i \leq n.$$

- Mede-se o erro ou desvio do esperado
- Obtém-se uma proporção para alteração do peso
- Alpha indica o quanto iremos aceitar dessa proporção na adaptação de pesos

- Mais sobre Gradiente descendente

- O que ocorre com a adaptação dos pesos?
 - Consideremos o Erro versus somente um peso w_1
- $$f(x) = x^2$$



- Para encontrar o mínimo devemos:
 - Encontrar a derivada da função na direção do peso
- $$\frac{\partial f(x)}{\partial x}$$
- Para chegarmos ao mínimo devemos, para um dado peso w_1 , adaptá-lo em pequenos passos
 - Se passos grandes, “dancamos” ao redor do mínimo
- $$x(t + 1) = x(t) - \mu \frac{\partial f(x(t))}{\partial x}$$
- Se mudarmos o sinal para positivo, seguimos em direção ao máximo da função

Implementação

```
x_old = 0
x_new = 6 # valor inicial a ser aplicado na função
eps = 0.01 # passo
Precision = 0.00001
```

```
double derivada(double x) { return 2 * x; }
```

```
while (fabs(x_new - x_old) > precision) {
    x_old = x_new
    x_new = x_old - eps * derivada(x_new)
    printf("Local minimum occurs at %f \n", x_new);
}
```

*Testar com diferentes valores para o passo

*Verificar a mudança de sinal para eps

- Formalizando a equação de adaptação

- Como chegamos a essa equação de adaptação?

$$w_i(t + 1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}, \text{ for all nodes } 0 \leq i \leq n.$$
- Considere um vetor de entrada como \mathbf{x}
- Considere um conjunto de treinamento $\{\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_L\}$
- Considere que cada \mathbf{x} deve gerar um valor de saída \mathbf{d}
 - Logo $\{d_0, d_1, \dots, d_L\}$
- Considere que cada \mathbf{x} gerou, de fato, uma saída \mathbf{y}
- O problema consiste em encontrar um único de vetor de pesos \mathbf{w}^* que satisfaça essa relação de entradas e saídas esperadas
 - Ou que gere o menor erro possível, i.e., se aproxime mais dessa relação

- Consideremos a diferença entre saída esperada e saída real para um padrão de entrada \mathbf{x} como:

$$\epsilon_k = d_k - y_k$$

- Logo o **erro médio quadrático** esperado para todos padrões de entrada usados no treinamento será:

$$\langle \epsilon^2 \rangle = \frac{1}{L} \sum_{k=1}^L \epsilon_k^2 \quad *Por\ que\ quadrático?$$

- Desconsiderando a função de transferência degrau, temos:

$$y = \mathbf{w}^t \mathbf{x}$$

- Logo podemos assumir o erro médio quadrático esperado para um exemplo k por:

$$\langle \epsilon_k^2 \rangle = \langle (d_k - \mathbf{w}^t \mathbf{x}_k)^2 \rangle$$

- Assim:

$$\begin{aligned} \langle \epsilon_k^2 \rangle &= \langle (d_k - \mathbf{w}^t \mathbf{x}_k)^2 \rangle \\ \langle \epsilon_k^2 \rangle &= \langle d_k^2 \rangle - 2 \langle d_k \mathbf{x}_k^t \rangle \mathbf{w} + \mathbf{w}^t \mathbf{w} \langle \mathbf{x}_k \mathbf{x}_k^t \rangle \end{aligned}$$

- Observar que transpostas são necessárias para o produto de vetores

- Podemos definir a matrix R, matrix de correlação de entradas, para simplificar a formulação

$$\mathbf{R} = \langle \mathbf{x}_k \mathbf{x}_k^t \rangle$$

O Perceptron

O Perceptron

- O seguinte termo pode ser representado pelo vetor \mathbf{p}^t

$$\mathbf{p}^t = \langle d_k \mathbf{x}_k^t \rangle$$

- Assumindo:

$$\varepsilon = \langle \epsilon_k^2 \rangle$$

- Logo:

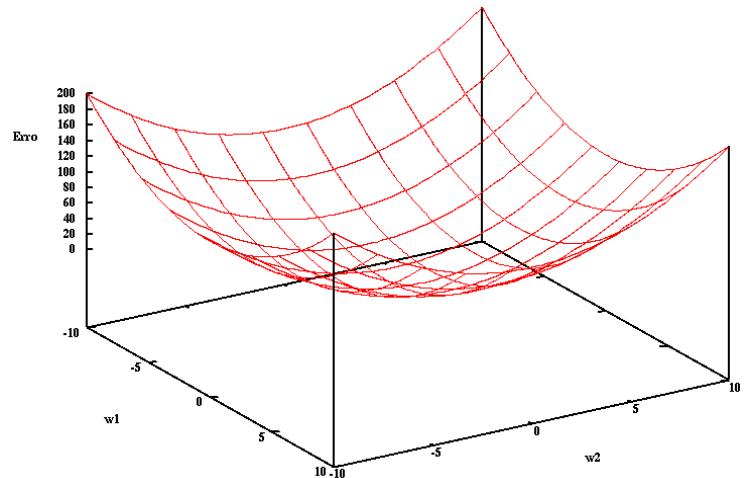
$$\varepsilon = \langle d_k^2 \rangle + \mathbf{w}^t \mathbf{R} \mathbf{w} - 2 \mathbf{p}^t \mathbf{w}$$

- Assim para reduzir o erro podemos encontrar a derivada em função dos pesos atuais

$$\varepsilon = \varepsilon(\mathbf{w}) = \frac{\partial \varepsilon(\mathbf{w})}{\partial \mathbf{w}} = 2 \mathbf{R} \mathbf{w} - 2 \mathbf{p}$$

- Ou seja, como o erro varia em função dos pesos?
- Lembrar que \mathbf{R} é dado pelo exemplo de entrada k

- Essa derivada considera:



O Perceptron

O Perceptron

- Igualando a derivada a zero, podemos encontrar o mínimo

- Há prova de concavidade para cima

$$\varepsilon = \varepsilon(\mathbf{w}) = \frac{\partial \varepsilon(\mathbf{w})}{\partial \mathbf{w}} = 2 \mathbf{R} \mathbf{w} - 2 \mathbf{p}$$

- Dessa maneira (sendo \mathbf{w}^* o vetor ideal de pesos):

$$2 \mathbf{R} \mathbf{w}^* - 2 \mathbf{p} = 0$$

$$\mathbf{R} \mathbf{w}^* = \mathbf{p}$$

$$\mathbf{w}^* = \mathbf{R}^{-1} \mathbf{p}$$

- Consideremos uma situação real com quatro exemplos

INPUT	OUTPUT
A	B
0	0
0	1
1	0
1	1
A NAND B	
1	1

$$\mathbf{R} = \langle \mathbf{x}_k \mathbf{x}_k^t \rangle = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$\mathbf{p} = \langle d_k \mathbf{x}_k^t \rangle = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

- Consideremos uma situação real

- Solução Iterativa

INPUT	OUTPUT	
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

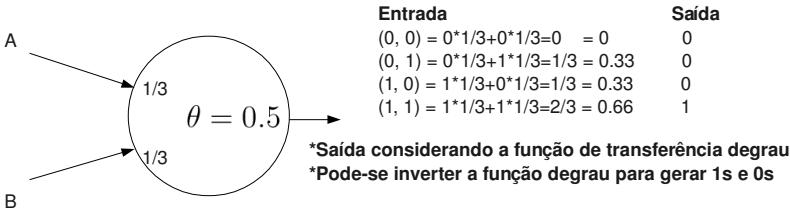
$$\mathbf{Rw}^* = \mathbf{p}$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$2w_1 + w_2 = 1$$

$$w_1 + 2w_2 = 1$$

$$w_2 = \frac{1}{3} \quad w_1 = \frac{1}{3}$$



- Essa solução exata depende:
 - De todos vetores de entrada
- No entanto, se quisermos uma versão iterativa do algoritmo de aprendizado, podemos aproximar ao invés de adotar a solução ideal apresentada, assim:

$$\langle \epsilon_k^2 \rangle = \langle (d_k - \mathbf{w}^t \mathbf{x}_k)^2 \rangle$$

- Pode ser aproximado instantaneamente por:

$$\epsilon_i^2(t) = (d_i - \mathbf{w}^t(t) \mathbf{x}_i)^2$$

- Em que i se refere ao erro para uma entrada \mathbf{x}_i e saída esperada d_i

- Para isso **derivamos o erro em função dos pesos** para que possamos adaptá-los:

$$\nabla \epsilon_i^2(t) \approx \nabla \langle \epsilon_i^2 \rangle$$

$$\nabla \epsilon_i^2(t) = -2\epsilon_i(t) \mathbf{x}_i$$

- Passos:

$$\epsilon_i^2(t) = (d_i - \mathbf{w}^t(t) \mathbf{x}_i)^2$$

Logo:

$$\frac{d}{d\mathbf{w}} \epsilon_i^2(t) = 2 \cdot (d_i - \mathbf{w}^t(t) \mathbf{x}_i) \cdot -\mathbf{x}_i$$

Pela regra da cadeia, temos:

$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x)$$

$$\frac{d}{d\mathbf{w}} \epsilon_i^2(t) = -2 \cdot \epsilon_i(t) \cdot \mathbf{x}_i$$

- Conforme visto anteriormente o gradiente descendente é dado por:

$$x(t+1) = x(t) - \mu \frac{\partial f(x(t))}{\partial x}$$

- Em nosso caso modelamos como:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \mu \nabla \epsilon(\mathbf{w}(t))$$

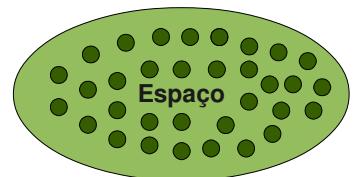
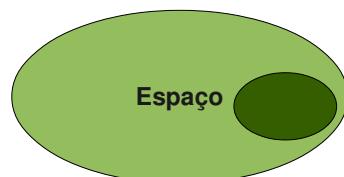
Sendo: $\nabla \epsilon(\mathbf{w}(t)) = \nabla \epsilon_i^2(t)$ logo:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + 2\mu \epsilon_i \mathbf{x}_i$$

- Em que μ é a taxa de aprendizado [0,1]

- Observações:

- Conjunto de treinamento deve ser significativo para adaptar os pesos
 - Conjunto com diversidade
 - Deve ter padrões (samples ou exemplos) que representem bem a população total de possibilidades
- Caso contrário testes não gerarão os resultados que esperamos



Mesma quantidade de samples, porém menos significativo no primeiro caso e melhor no segundo cenário

O Perceptron

- Implementação
 - XOR
 - Verificar a implementação do Perceptron para conjuntos de treinamento e teste para o problema de classificação do XOR

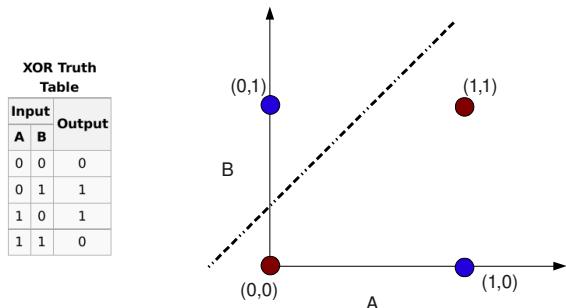
XOR Truth Table		
Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

Perceptron: Resolvendo o Problema do XOR

- Minsky and Papert (1969) escreveram o livro “Perceptrons: An Introduction to Computational Geometry”, MIT
 - Demonstraram que perceptron separa classes linearmente
 - No entanto, diversos problemas (p.e., XOR) não são linearmente separáveis
 - A forma como escreveram o livro parece dar descredito à área
 - Como, na época, o perceptron resumia grande parte da área, então, acreditou-se que redes neurais artificiais e, até mesmo IA, não seriam úteis para problemas reais

Perceptron: Resolvendo o Problema do XOR

- Como separar essas classes?
- Quais pesos adotar? Qual bias?



Perceptron: Resolvendo o Problema do XOR

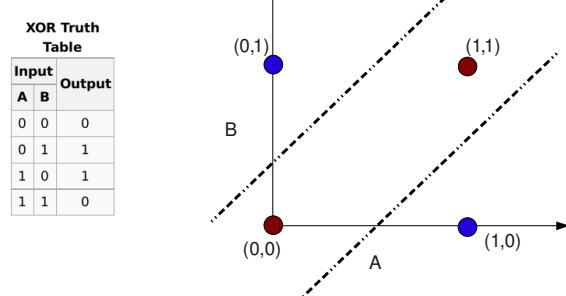
- Observe que a equação é linear

$$net_i = w_1x_1 + w_2x_2$$
- O resultado dessa equação é aplicada na função de ativação

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$
- Assim, pode-se, apenas, separar classes linearmente
- Em problemas linearmente separáveis
 - Essa equação representa um hiperplano
 - Hiperplanos são objetos de $n-1$ dimensões usados para separar hiperspaços de n dimensões em subregiões

Perceptron: Resolvendo o Problema do XOR

- Poderíamos usar dois hiperplanos
- Regiões disjuntas podem pertencer a uma mesma classe



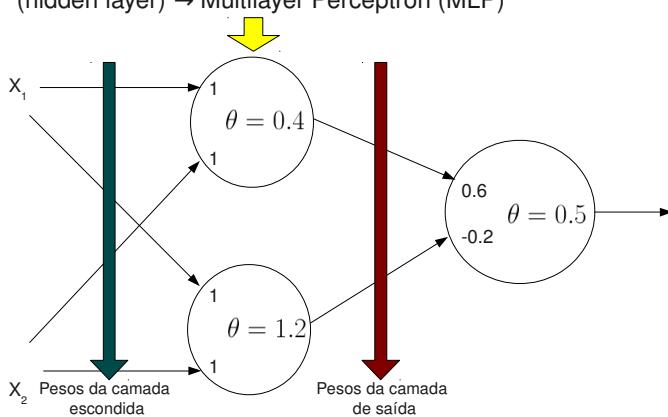
Perceptron: Resolvendo o Problema do XOR

- Esse fato não anula algumas das questões levantadas por Minsky and Papert
 - Eles ainda questionam a escalabilidade das redes neurais artificiais
 - Conforme se aborda um problema de larga escala, há efeitos indesejáveis:
 - Treinamento mais lento
 - Muitos neurônios tendem a tornar treinamento mais lento ou dificultam convergência
 - Alguns pesquisadores afirmam que se pode combinar redes menores
 - Porém mostra que a área está em constante evolução

Multilayer Perceptron: Resolvendo o Problema do XOR

- Implementação para o XOR

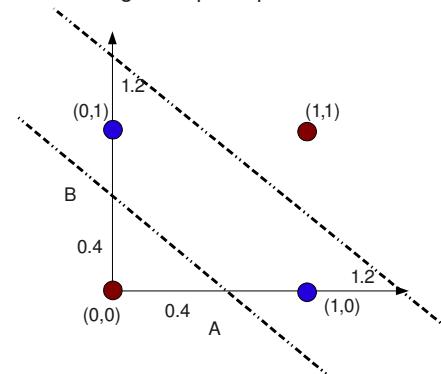
- Camada adicional também chamada de camada escondida (hidden layer) → Multilayer Perceptron (MLP)



Multilayer Perceptron: Resolvendo o Problema do XOR

- Implementação para o XOR

- Camada adicional também chamada de camada escondida (hidden layer)
- Este resultado foi gerado pelos parâmetros anteriores...



Multilayer Perceptron: Resolvendo o Problema do XOR

- Implementação para o XOR

- No entanto há um novo problema

- Como treinar essa rede?

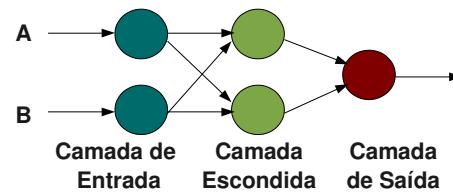
- Treinamento significa alterar pesos para representar os vetores de entrada
- Devemos encontrar uma forma de treinar os pesos da camada de saída e da camada anterior ou escondida
 - O termo “escondida” surge devido ao fato que a primeira camada pode ser vista como as entradas simplesmente

Multilayer Perceptron: Resolvendo o Problema do XOR

- Implementação para o XOR

- Topologia

- Tamanho da entrada: 2 bits
- Tamanho da saída: 1 bit
- Número de neurônios na camada de saída: 1
- Número de neurônios na camada de entrada = Tamanho da entrada, ou seja, 2
- Número de neurônios na camada escondida pode variar

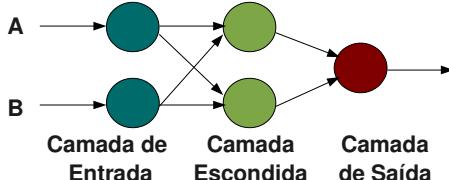


XOR Truth Table		
Input	Output	
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Multilayer Perceptron: Resolvendo o Problema do XOR

- Rede feedforward
- Entradas geram saídas
- Não há retroalimentação ou recorrência tal como em Redes Neurais Artificiais BAM e Hopfield

XOR Truth Table		
Input	Output	
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



- Utiliza-se Algoritmo Backpropagation para treinamento
 - Erro propaga da última camada em direção à primeira

- O treinamento (adaptação de pesos) ocorre em função do erro medido na camada de saída

- O aprendizado segue a Regra Delta Generalizada (RDG)

- É uma generalização da LMS (Least Mean Square) vista anteriormente
- LMS é utilizada para regressão linear (separa espaço com reta)
 - A regra delta generalizada permite regressão não linear

- Suponha:

- Os pares de vetores (entrada, saída esperada):

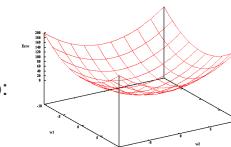
$$(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_p, \mathbf{y}_p)$$

- Dado:

$$\mathbf{y} = \phi(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^N, \mathbf{y} \in \mathbb{R}^M$$

- Objetivo do treinamento é obter uma aproximação:

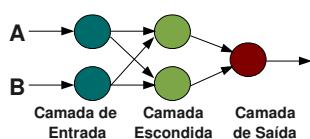
$$\bar{\mathbf{y}} = \bar{\phi}(\mathbf{x})$$



Multilayer Perceptron: Regra Delta Generalizada

- A camada de entrada é simples:
 - Neurônios apenas encaminham valores para a camada escondida
- Camada escondida computa:

$$\text{net}_{pj}^h = \sum_{i=1}^N w_{ji}^h x_{pi} + \theta_j^h$$



- Camada de saída computa:

$$\text{net}_{pk}^o = \sum_{j=1}^L w_{kj}^o x_{pj} + \theta_k^o$$

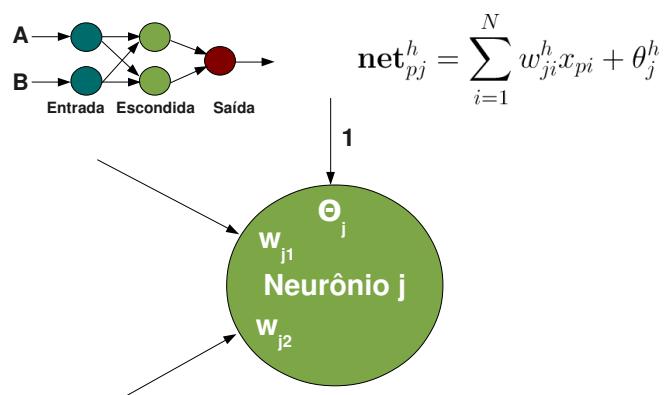
- Em que:

w_{ji}^h é o peso da conexão com o neurônio de entrada i
 w_{kj}^o é o peso da conexão com o neurônio j da camada escondida
 θ_j^h e θ_k^o são os bias

Multilayer Perceptron: Regra Delta Generalizada

- Por exemplo:

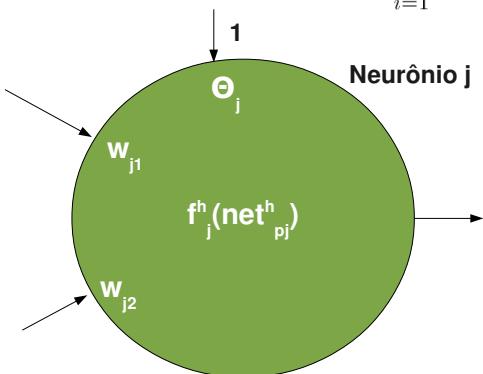
- Considere um neurônio da camada escondida



Multilayer Perceptron: Regra Delta Generalizada

- Mais detalhes...

$$\text{net}_{pj}^h = \sum_{i=1}^N w_{ji}^h x_{pi} + \theta_j^h$$



Multilayer Perceptron: Regra Delta Generalizada Atualização dos Pesos da Camada de Saída

- Atualização dos pesos da Camada de Saída

- Na camada de saída pode haver vários neurônios

- O erro para um dado neurônio nessa camada é dado por:

$$\delta_{pk} = (y_{pk} - o_{pk})$$

- Em que:

y_{pk} saída esperada do neurônio k para vetor de entrada p
 o_{pk} saída produzida pelo neurônio k para vetor de entrada p
 p identifica o vetor de entrada usado no treinamento
 k indica o neurônio da camada de saída

Multilayer Perceptron: Regra Delta Generalizada Atualização dos Pesos da Camada de Saída

- O objetivo é minimizar a soma de quadrados de erro para todas as unidades de saída, considerando uma entrada p

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2$$

- O fator $1/2$ foi adicionado apenas para auxiliar no posterior cálculo da derivada

- Como haverá uma constante na adaptação de peso, essa constante $1/2$ não invalida a definição da regra

- M indica o número de neurônios na camada de saída

- Importante:

- Erro relativo à cada entrada elevado ao quadrado

- Nosso objetivo é dar um “passo” no sentido de redução de erro o qual varia em função dos pesos w

$$E_p = \frac{1}{2} \sum_{k=1}^M \delta_{pk}^2$$

$$\delta_{pk} = (y_{pk} - o_{pk})$$

$$E_p = \frac{1}{2} \sum_{k=1}^M (y_{pk} - o_{pk})^2$$

Multilayer Perceptron: Regra Delta Generalizada Atualização dos Pesos da Camada de Saída

- Derivando o erro em função do que pode ser alterado (pesos w), para isso temos (segundo a regra da cadeia):

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x) \text{ ou } \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

- Simplificando:

$$E_{pk} = \frac{1}{2}(y_{pk} - o_{pk})^2 \text{ em que:}$$

$$f(g(x)) = \frac{1}{2}(y_{pk} - o_{pk})^2$$

$$g(x) = y_{pk} - o_{pk}$$

- Assim:

$$f'(g(x)) = 2 \cdot \frac{1}{2}(y_{pk} - o_{pk})$$

$$g'(x) = 0 - o'_{pk}$$

em que y_{pk} é uma constante (saída esperada)

- Sendo:

$$o_{pk} = f_k^o(\text{net}_{pk}^o)$$

- Logo a derivada será (também pela regra da cadeia):

$$o'_{pk} = \frac{\partial f_k^o}{\partial \text{net}_{pk}^o} \cdot \frac{\partial \text{net}_{pk}^o}{\partial w_{kj}^o}$$

Multilayer Perceptron: Regra Delta Generalizada Atualização dos Pesos da Camada de Saída

- Unificando:

$$f'(g(x))g'(x) = \left(2 \cdot \frac{1}{2}(y_{pk} - o_{pk})\right) \cdot \left(0 - \frac{\partial f_k^o}{\partial \text{net}_{pk}^o} \frac{\partial \text{net}_{pk}^o}{\partial w_{kj}^o}\right)$$

- Logo:

$$\frac{\partial E_{pk}}{\partial w_{kj}^o} = -(y_{pk} - o_{pk}) \frac{\partial f_k^o}{\partial \text{net}_{pk}^o} \frac{\partial \text{net}_{pk}^o}{\partial w_{kj}^o}$$

- Resolvendo o último termo:

$$\begin{aligned} \text{net} &= \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o \\ \frac{\partial \text{net}_{pk}^o}{\partial w_{kj}^o} &= \frac{\partial}{\partial w_{kj}^o} \left(\sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o \right) = i_{pj} \end{aligned}$$

- Substituindo:

$$\frac{\partial E_{pk}}{\partial w_{pj}^o} = -(y_{pk} - o_{pk}) \frac{\delta f_k^o}{\partial \text{net}_{pk}^o} i_{pj}$$

- Resta derivar a função de ativação

- A função deve ser diferenciável

- Isso anula a função degrau usada anteriormente com LMS

- Exemplos de funções de ativação que podem ser utilizadas:

1) se $f_k^o(\text{net}_k^o) = \text{net}_k^o$ então $f'_k(\text{net}_k^o) = 1$ Função Linear
assim como $f(x) = x$ temos $f'(x) = 1$

2) se $f_k^o(\text{net}_k^o) = (1 + e^{-\text{net}_k^o})^{-1}$ então
 $f'_k(\text{net}_k^o) = f_k^o(1 - f_k^o)$ Função Sigmóide

Multilayer Perceptron: Regra Delta Generalizada Atualização dos Pesos da Camada de Saída

- Considerando as duas possibilidades para funções de ativação, temos as adaptações de peso conforme:

- Para o primeiro caso:

$$f_k^o(\text{net}_{jk}^o) = \text{net}_{jk}^o$$

- Temos:

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})i_{pj}$$

- Para a função sigmóide:

$$f_k^o(\text{net}_{jk}^o) = \frac{1}{1 + e^{-\text{net}_{jk}^o}}$$

- Temos:

$$f'_k(\text{net}_k^o) = f_k^o(1 - f_k^o) \text{ logo, neste cenário } f'_k(\text{net}_k^o) = o_{pk}(1 - o_{pk})$$

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta(y_{pk} - o_{pk})o_{pk}(1 - o_{pk})i_{pj}$$

Multilayer Perceptron: Regra Delta Generalizada Atualização dos Pesos da Camada de Saída

- Podemos definir o termo de adaptação de maneira genérica, ou seja, para qualquer função de ativação:

$$\delta_{pk}^o = (y_{pk} - o_{pk})f'_k(\text{net}_{pk}^o)$$

- E generalizar (**Regra Delta Generalizada**) a adaptação de pesos para qualquer função de ativação, temos:

$$w_{kj}^o(t+1) = w_{kj}^o(t) + \eta\delta_{pk}^o i_{pj}$$

- Atualização dos pesos da Camada Escondida

- Como conhecer a saída esperada pela camada escondida?
 - Na camada de saída conhecemos a saída
- De alguma forma o erro E_p medido na camada de saída deve influenciar os pesos na camada escondida

- Assim o erro medido na camada de saída é dado por:

$$\begin{aligned} E_p &= \frac{1}{2} \sum_k (y_{pk} - o_{pk})^2 \\ &= \frac{1}{2} \sum_k (y_{pk} - f_k^o(\text{net}_{pk}^o))^2 \\ &= \frac{1}{2} \sum_k \left(y_{pk} - f_k^o \left(\sum_j w_{kj}^o i_{pj} + \theta_k^o \right) \right)^2 \end{aligned}$$

- O termo i_{pj} se trata dos valores vindos da camada anterior, ou seja, da camada escondida
- Podemos explorar esse fato para montar o equacionamento e adaptação da camada escondida

- Dessa maneira definimos a variação do erro em função da camada escondida:

$$\begin{aligned} \frac{\partial E_p}{\partial w_{ji}^h} &= \frac{1}{2} \sum_k \frac{\partial}{\partial w_{ji}^h} (y_{pk} - o_{pk})^2 \\ &= - \sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \text{net}_{pk}^o} \frac{\partial \text{net}_{pk}^o}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial \text{net}_{pj}^h} \frac{\partial \text{net}_{pj}^h}{\partial w_{ji}^h} \end{aligned}$$

- A partir do equacionamento anterior obtemos:

$$\frac{\partial E_p}{\partial w_{ji}^h} = - \sum_k (y_{pk} - o_{pk}) f_k^{ot}(\text{net}_{pk}^o) w_{kj}^o f_j^{ht}(\text{net}_{pj}^h) x_{pi}$$

- Dessa maneira definimos a variação do erro em função da camada escondida:

$$\begin{aligned} \frac{\partial E_p}{\partial w_{ji}^h} &= \frac{1}{2} \sum_k \frac{\partial}{\partial w_{ji}^h} (y_{pk} - o_{pk})^2 \\ &= - \sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \text{net}_{pk}^o} \frac{\partial \text{net}_{pk}^o}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial \text{net}_{pj}^h} \frac{\partial \text{net}_{pj}^h}{\partial w_{ji}^h} \\ &\quad \text{Derivada da saída da função de ativação da camada de saída pelo net} \\ &\quad \text{Derivada da saída da função de ativação da camada escondida pelo net} \end{aligned}$$

- A partir do equacionamento anterior obtemos:

$$\frac{\partial E_p}{\partial w_{ji}^h} = - \sum_k (y_{pk} - o_{pk}) f_k^{ot}(\text{net}_{pk}^o) w_{kj}^o f_j^{ht}(\text{net}_{pj}^h) x_{pi}$$

- Sendo:

$$\begin{aligned} \text{net}_{pk}^o &= \sum_{j=1}^L w_{kj}^o i_{pj} + \theta_k^o \\ \text{net}_{pi}^h &= \sum_{j=1}^L w_{kj}^h x_{pi} + \theta_k^h \end{aligned}$$

$$\begin{aligned} \frac{\partial E_p}{\partial w_{ji}^h} &= \frac{1}{2} \sum_k \frac{\partial}{\partial w_{ji}^h} (y_{pk} - o_{pk})^2 \\ &= - \sum_k (y_{pk} - o_{pk}) \frac{\partial o_{pk}}{\partial \text{net}_{pk}^o} \frac{\partial \text{net}_{pk}^o}{\partial i_{pj}} \frac{\partial i_{pj}}{\partial \text{net}_{pj}^h} \frac{\partial \text{net}_{pj}^h}{\partial w_{ji}^h} \\ &\quad \text{Derivada da saída da função de ativação da camada de saída pelo net} \\ &\quad \text{Derivada da saída da função de ativação da camada escondida pelo net} \end{aligned}$$

$$\frac{\partial E_p}{\partial w_{ji}^h} = - \sum_k (y_{pk} - o_{pk}) f_k^{ot}(\text{net}_{pk}^o) w_{kj}^o f_j^{ht}(\text{net}_{pj}^h) x_{pi}$$

- Podemos, então, calcular a atualização dos pesos de neurônios da camada escondida como:

$$\Delta_p w_{ji}^h = \eta f_j^{ht}(\text{net}_{pj}^h) x_{pi} \sum_k (y_{pk} - o_{pk}) f_k^{ot}(\text{net}_{pk}^o) w_{kj}^o$$

$$\Delta_p w_{ji}^h = \eta f_j^{ht}(\text{net}_{pj}^h) x_{pi} \sum_k \delta_{pk}^o w_{kj}^o$$

- Assim a adaptação dos pesos da camada escondida depende do erro da camada de saída
- Dessa noção de dependência do erro da camada de saída que surgiu o termo **Backpropagation**

- Pode-se, assim como para a camada de saída, computar o termo delta para a camada escondida:

$$\delta_{pj}^h = f_j^{h\prime}(\text{net}_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o$$

- Assim a atualização de pesos da camada escondida é dada por:

$$w_{ji}^h(t+1) = w_{ji}^h(t) + \eta \delta_{pj}^h x_i$$

- Finalmente podemos implementar o aprendizado da MLP

- Algoritmo Backpropagation de Aprendizado**

- Funções essenciais para a adaptação de pesos:

- Considerando Funções Sigmóides de ativação:

$$f(x) = \frac{1}{1 + e^{-x}} \quad f'(x) = f(x) \cdot (1 - f(x))$$

- Camada de Saída:

$$\begin{aligned} \delta_{pk}^o &= (y_{pk} - o_{pk}) f_k^{o\prime}(\text{net}_{pk}^o) \\ w_{kj}^o(t+1) &= w_{kj}^o(t) + \eta \delta_{pk}^o i_{pj} \end{aligned}$$

Muito Importante!!!

Primeiramente
calcular todos os
deltas para depois
atualizar pesos!!!

- Camada Escondida:

$$\begin{aligned} \delta_{pj}^h &= f_j^{h\prime}(\text{net}_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \\ w_{ji}^h(t+1) &= w_{ji}^h(t) + \eta \delta_{pj}^h x_i \end{aligned}$$

Multilayer Perceptron

- Algoritmo Backpropagation de Aprendizado**

- Funções essenciais para a adaptação de pesos:

- Considerando Funções Sigmóides de ativação:

$$f(x) = \frac{1}{1 + e^{-x}} \quad f'(x) = f(x) \cdot (1 - f(x))$$

- Camada de Saída:

$$\begin{aligned} \delta_{pk}^o &\leftarrow (y_{pk} - o_{pk}) f_k^{o\prime}(\text{net}_{pk}^o) \\ w_{kj}^o(t+1) &= w_{kj}^o(t) + \eta \delta_{pk}^o i_{pj} \end{aligned}$$

- Camada Escondida:

$$\begin{aligned} \delta_{pj}^h &= f_j^{h\prime}(\text{net}_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \\ w_{ji}^h(t+1) &= w_{ji}^h(t) + \eta \delta_{pj}^h x_i \end{aligned}$$

- Algoritmo Backpropagation de Aprendizado**

- Funções essenciais para a adaptação de pesos:

- Considerando Funções Sigmóides de ativação:

$$f(x) = \frac{1}{1 + e^{-x}} \quad f'(x) = f(x) \cdot (1 - f(x))$$

- Camada de Saída:

$$\begin{aligned} \delta_{pk}^o &\leftarrow (y_{pk} - o_{pk}) f_k^{o\prime}(\text{net}_{pk}^o) \\ w_{kj}^o(t+1) &= w_{kj}^o(t) + \eta \delta_{pk}^o i_{pj} \end{aligned}$$

Multilayer Perceptron

- Camada Escondida:

$$\begin{aligned} \delta_{pj}^h &= f_j^{h\prime}(\text{net}_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \\ w_{ji}^h(t+1) &= w_{ji}^h(t) + \eta \delta_{pj}^h x_i \end{aligned}$$

Multilayer Perceptron

- Algoritmo Backpropagation de Aprendizado**

- Funções essenciais para a adaptação de pesos:

- Considerando Funções Sigmóides de ativação:

$$f(x) = \frac{1}{1 + e^{-x}} \quad f'(x) = f(x) \cdot (1 - f(x))$$

- Camada de Saída:

$$\begin{aligned} \delta_{pk}^o &\leftarrow (y_{pk} - o_{pk}) f_k^{o\prime}(\text{net}_{pk}^o) \\ w_{kj}^o(t+1) &= w_{kj}^o(t) + \eta \delta_{pk}^o i_{pj} \end{aligned}$$

- Camada Escondida:

$$\begin{aligned} \delta_{pj}^h &= f_j^{h\prime}(\text{net}_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \\ w_{ji}^h(t+1) &= w_{ji}^h(t) + \eta \delta_{pj}^h x_i \end{aligned}$$

- Algoritmo Backpropagation de Aprendizado**

- Funções essenciais para a adaptação de pesos:

- Considerando Funções Sigmóides de ativação:

$$f(x) = \frac{1}{1 + e^{-x}} \quad f'(x) = f(x) \cdot (1 - f(x))$$

- Camada de Saída:

$$\begin{aligned} \delta_{pk}^o &= (y_{pk} - o_{pk}) f_k^{o\prime}(\text{net}_{pk}^o) \\ w_{kj}^o(t+1) &= w_{kj}^o(t) + \eta \delta_{pk}^o i_{pj} \end{aligned}$$

Multilayer Perceptron

- Camada Escondida:

$$\begin{aligned} \delta_{pj}^h &= f_j^{h\prime}(\text{net}_{pj}^h) \sum_k \delta_{pk}^o w_{kj}^o \\ w_{ji}^h(t+1) &= w_{ji}^h(t) + \eta \delta_{pj}^h x_i \end{aligned}$$

Valores de entrada vindos da
camada escondida

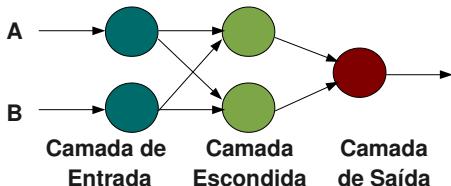
Valores de entrada vindos da
camada de entrada

Multilayer Perceptron

- Implementação

- XOR

XOR Truth Table		
Input	Output	
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0



- Implementação

- OCR (Optical Character Recognition)

- Soluções:

- Montar uma Tabela Hash (Bits → Código ASCII)
 - Problema quando um dos bits ou mais (aleatórios) apresentam ruídos)
 - Como tratar esses ruídos?
- MLP
 - Capaz de aprender e generalizar conhecimento
 - Mesmo na presença de ruídos é capaz de gerar bons resultados

Multilayer Perceptron

- Implementação

- OCR (Optical Character Recognition)

- Entrada dada por uma matrix 7x5

- Exemplo (separar letra A da letra B)



- Estender para separar qualquer caracter

- Estender para qualquer problema de classificação

- Implementação

- Reconhecimento de Faces

- Classificação de Músicas

- Considerar Datasets:

- UCI (<http://archive.ics.uci.edu/ml/datasets.html>):

- Iris
- Reuters-21578 Text Categorization Collection
- CMU Face Images

- Million Song Dataset

- <http://labrosa.ee.columbia.edu/millionsong/pages/getting-dataset>

Associative Memory

- Utilizada para associar um vetor de entrada a uma saída
- Em alguns casos (aplicações específicas) o Perceptron ou a MLP são tipos de memórias associativas
- Memória associativa é útil:
 - Por exemplo:
 - Associar nome a endereço
 - Converter padrões, por exemplo, EBCDIC para ASCII
 - Conversão de bases (hexadecimal, binária, etc.)
 - Associar palavra falada a uma operação (URA em telefonia)

- Redes neurais artificiais baseadas em memórias associativas costumam ser usadas para **reconstrução de padrões**:

- Imagine um padrão:

01010010 que representa um caractere ASCII

- Considere que esse padrão foi aprendido pela rede neural
- Agora considere que precisamos imprimir o caractere contido em uma posição de memória e seu valor é:

11010010

- Pode-se, nesse caso, utilizar uma rede neural de memória associativa para reconstruir o padrão correto

- O mesmo ocorre, por exemplo, com o áudio que falamos em URA (há ruído ou a voz varia)

Associative Memory

- Para essa reconstrução as redes neurais de memória associativa geralmente utilizam a distância de **Hamming**
 - É uma distância que mede a diferença entre dois valores em função dos bits distintos
 - Por exemplo: 1010_2
 1110_2
 - Distância igual a 1, pois há um bit distinto
 - Exemplo com palavras:
teste
tosta
 - Distância igual a 2

- Formalmente o espaço de **Hamming** é definido por:

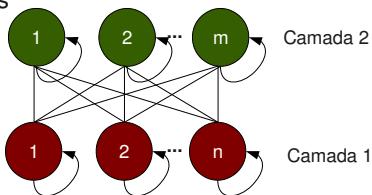
$$\mathbf{H}^n = \{\mathbf{x} = (x_1, x_2, \dots, x_n)^t \in \mathbb{R}^n : x_i \in \{\pm 1\}\}$$

- As redes neurais artificiais de memória associativa:

- Utilizam vetores com elementos {-1, +1} para representar instâncias ou padrões a serem aprendidos
- Também representam saídas esperadas por meio de vetores com elementos {-1, +1}

BAM: Bidirectional Associative Memory

- Nossa primeira **arquitetura** de rede neural de memória associativa a ser estudada
 - Consiste de duas camadas totalmente interconectadas
 - Neurônios podem ou não ter conexões para si próprios
 - Assim como outras arquiteturas de redes neurais:
 - Há pesos associados às conexões
 - No entanto, esses pesos podem ser previamente definidos



BAM: Bidirectional Associative Memory

- A definição dos pesos é oriunda do modelo de associador linear que afirma que se pode mapear uma dada entrada \mathbf{x}

$$\Phi(\mathbf{x}) = (\mathbf{y}_1 \mathbf{x}_1^t + \mathbf{y}_2 \mathbf{x}_2^t + \dots + \mathbf{y}_L \mathbf{x}_L^t) \mathbf{x}$$

- Considerando que essa função foi criada a partir de um conjunto de treinamento

$$\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \dots, (\mathbf{x}_L, \mathbf{y}_L)\}$$

- Assim, constrói-se uma matrix de pesos para representar associações entre uma entrada \mathbf{x} e uma saída \mathbf{y} na forma:

$$\mathbf{w} = \mathbf{y}_1 \mathbf{x}_1^t + \mathbf{y}_2 \mathbf{x}_2^t + \dots + \mathbf{y}_L \mathbf{x}_L^t$$

BAM: Bidirectional Associative Memory

- Por exemplo, considere as seguintes entradas e respectivas saídas esperadas:

$$\begin{aligned} \mathbf{x}_1 &= (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \text{ e } \mathbf{y}_1 = (1, -1, -1, -1, 1, 1)^t \\ \mathbf{x}_2 &= (1, 1, 1, -1, -1, 1, 1, -1, -1)^t \text{ e } \mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t \end{aligned}$$

- Para obter a matrix de pesos computa-se o produto de cada elemento da entrada pela saída como segue:

$$\mathbf{y}_1 = \begin{bmatrix} 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 \end{bmatrix}$$

- Após isso, soma-se as matrizes resultantes dessas computações e obtém-se a matrix de pesos \mathbf{w}

- Assim, para as entradas e saídas:

$$\begin{aligned} \mathbf{x}_1 &= (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \text{ e } \mathbf{y}_1 = (1, -1, -1, -1, 1, 1)^t \\ \mathbf{x}_2 &= (1, 1, 1, -1, -1, 1, 1, -1, -1)^t \text{ e } \mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t \end{aligned}$$

- Teremos:

$$\mathbf{w} = \mathbf{y}_1 \mathbf{x}_1^t + \mathbf{y}_2 \mathbf{x}_2^t$$

$$\mathbf{w} = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{bmatrix}$$

BAM: Bidirectional Associative Memory

BAM: Bidirectional Associative Memory

- Essa matriz de pesos é então usada para associar entradas \mathbf{x} a saídas \mathbf{y} :

- Cada neurônio da BAM processa conforme um perceptron, ou seja:

$$\text{net}^y = \mathbf{w}\mathbf{x}$$

- Em que o processamento net^y ocorre na camada \mathbf{y}
- Ou seja, os valores que chegam para um neurônio em \mathbf{y} é dado por:

$$\text{net}^y = \sum_{j=1}^n w_{ij}x_j$$

- No entanto, como os neurônios na camada 1 e 2 dessa arquitetura de rede neural apresentam pesos de feedback então:

- Após apresentar uma entrada \mathbf{x} e ela fluir até a segunda camada, ela flui de volta para a camada 1
- Ou seja, um neurônio da camada 1 também é ativado por padrões que retornam (ou refletem) na camada 2:

$$\text{net}^x = \sum_{j=1}^m w_{ji}y_j$$

BAM: Bidirectional Associative Memory

BAM: Bidirectional Associative Memory

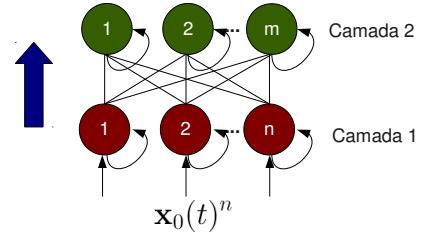
- Após computar net todo neurônio da BAM aplica uma função de ativação como segue:

$$y_i(t+1) = \begin{cases} +1 & \text{net}_i^y > 0 \\ y_i(t) & \text{net}_i^y = 0 \\ -1 & \text{net}_i^y < 0 \end{cases}$$

$$x_i(t+1) = \begin{cases} +1 & \text{net}_i^x > 0 \\ x_i(t) & \text{net}_i^x = 0 \\ -1 & \text{net}_i^x < 0 \end{cases}$$

- Implementação mais comum:

- Encaminha-se a entrada \mathbf{x}_0 , definida pelo usuário em t



- Processamento:

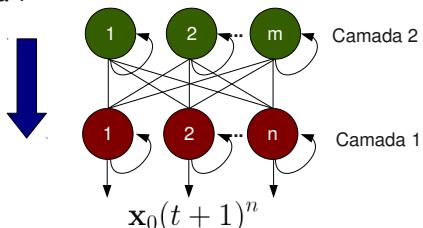
- Encaminha-se \mathbf{x}_0 para a Camada 2
- Computa-se o valor de net_i^y para cada neurônio da Camada 2
- Inicialmente os pesos de feedback podem ter um valor aleatório bipolar

BAM: Bidirectional Associative Memory

BAM: Bidirectional Associative Memory

- Processamento:

- Função de ativação da Camada 2 é computada e cada neurônio gera:
 $y_0(t+1)$
- O valor de $y_0(t+1)$ reside nas conexões de feedback que antes armazenavam $y_0(t) = \{-1, +1\}$
- O valor $y_0(t+1)$ também é encaminhado de volta para a Camada 1



- Processamento:

- Camada 1 produz saída $\mathbf{x}_0(t+1)^n$ a qual também é retroalimentada na BAM e o processamento continua tal como um loop
- Após k passos no tempo teremos um vetor de saída:
 $\mathbf{x}_0(t+k)^n$
- De tal forma que um próximo vetor no tempo, ou seja:
 $\mathbf{x}_0(t+k+1)^n = \mathbf{x}_0(t+k)^n$
- Isso significa que a rede neural de memória associativa convergiu, assim:
 - A Camada 2 modelou as saídas esperadas \mathbf{y}
 - A Camada 1 modelou as entradas \mathbf{x}
- Essas iterações auxiliam a reconstruir informações

BAM: Bidirectional Associative Memory

- Voltando ao exemplo inicial

$$\begin{aligned} \mathbf{x}_1 &= (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \text{ e } \mathbf{y}_1 = (1, -1, -1, -1, -1, 1)^t \\ \mathbf{x}_2 &= (1, 1, 1, -1, -1, -1, 1, 1, -1, -1)^t \text{ e } \mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t \end{aligned}$$

$$\mathbf{w} = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{bmatrix}$$

- Consideremos o caso:

$$x(0) = (-1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$$

- Seja \mathbf{y} inicializado como:

$$y(0) = (-1, -1, -1, -1, -1, -1)$$

- Após computar na Camada 2, assumindo que não há um \mathbf{x} anterior, temos:

$$x(0)\mathbf{w} = (4, -12, -12, -12, -4, 12)^t$$

- A Camada 2 aplica a função de ativação e gera:

$$y(1) = (1, -1, -1, -1, -1, 1)^t$$

- $y(1)$ é propagado de volta para a Camada 1

$$y(1)\mathbf{w} = (4, -8, -8, 8, -4, 8, 4, -8, -4, 8)^t$$

- Logo, a saída da Camada 1 é:

$$x(1) = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$$

BAM: Bidirectional Associative Memory

- Observem a entrada:

$$x(0) = (-1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$$

- Observem a saída em $x(1)$

$$x(1) = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$$

- Observem os padrões modelados em \mathbf{w}

$$\begin{aligned} \mathbf{x}_1 &= (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \text{ e } \mathbf{y}_1 = (1, -1, -1, -1, -1, 1)^t \\ \mathbf{x}_2 &= (1, 1, 1, -1, -1, -1, 1, 1, -1, -1)^t \text{ e } \mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t \end{aligned}$$

- Reparem que a entrada $\mathbf{x}(0)$ tinha “ruído” no primeiro bit e a rede neural foi capaz de reconstruir seu padrão original

- Se retroalimentarmos essa rede com $\mathbf{x}(1)$

- Continuaremos obtendo o mesmo valor
- Ou seja, não há variação, pois a rede estabilizou!

- Observação:

- A memória associativa “aprende” os padrões que utilizamos para gerar \mathbf{w} e também seus complementos:

$$\begin{aligned} \mathbf{x}_1 &= (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \text{ e } \mathbf{y}_1 = (1, -1, -1, -1, -1, 1)^t \\ \mathbf{x}_2 &= (1, 1, 1, -1, -1, -1, 1, 1, -1, -1)^t \text{ e } \mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t \end{aligned}$$

BAM: Bidirectional Associative Memory

- Se observarmos melhor a BAM ela tem comportamento dinâmico:

- O que isso significa?

- Um padrão de entrada \mathbf{x} gera \mathbf{y} na Camada 2
- O padrão \mathbf{y} é retroalimentado e retorna \mathbf{x}
- Avaliando \mathbf{x} podemos perceber se mudou ou não
 - Poderia criar um loop e parar a execução da rede neural somente quando \mathbf{x} converge, ou seja, não altera mais

- Esse comportamento de alteração de \mathbf{x} e \mathbf{y} pode ser visto como um comportamento dinâmico:

- Valores são alterados no tempo e dependem de valores anteriores

- Valores são alterados no tempo e dependem de valores anteriores:

$$x(t+1) = f(x(t))$$

- Por exemplo, consideremos o mapa Logístico dado pela equação:

$$x_{n+1} = rx_n(1 - x_n)$$

- O termo r é uma constante

- Essa equação é utilizada para modelar o crescimento de populações, tal como bactérias, animais, etc.

- Observe que seu próximo comportamento depende do anterior

- O mesmo ocorre com as entradas para a rede neural BAM

- Portanto, podemos vê-la como uma função $f(x(t))$

- Assim, podemos fazer a seguinte analogia:

$$x(0) \rightarrow \text{BAM} \rightarrow x(1)$$

- O que significa aplicar na BAM?

$$y(1) = x(0)\mathbf{w}$$

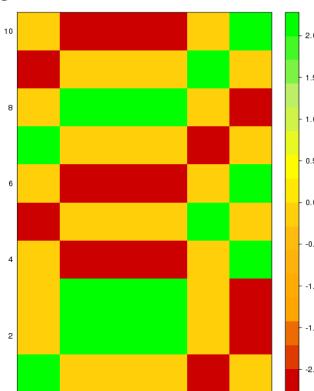
$$x(1) = y(1)\mathbf{w}$$

- O que \mathbf{w} faz?

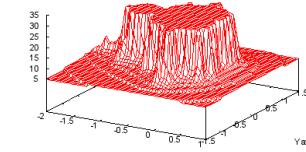
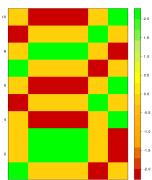
- Matrix \mathbf{w} foi criada a partir de entradas e saídas esperadas
- Ela resume similaridades e dissimilaridades de entradas e saídas esperadas

$$\mathbf{w} = \mathbf{y}_1\mathbf{x}_1^t + \mathbf{y}_2\mathbf{x}_2^t + \dots + \mathbf{y}_L\mathbf{x}_L^t$$

- Podemos plotar nossa matrix \mathbf{w} usada no exemplo anterior como uma superfície e verificar similaridades e dissimilaridades:



- Perceba que há extremos (mínimos e máximos)
- Quando entregamos uma entrada $\mathbf{x}(0)$ para a BAM ela processa até o $\mathbf{x}(0+k)$ convergir
 - Essa convergência funciona como a busca por mínimos
 - Portanto, se entregamos um mínimo, ela já está em um estado convergente
 - Caso contrário, irá procurar pelo mínimo, “descendo na superfície”



Obtida do site <http://gnuplot.sourceforge.net/demo/>

- Há duas versões:
 - Discreta
 - Contínua

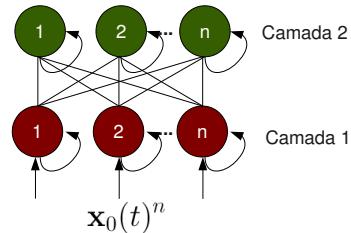
- Podemos entendê-la como uma derivação da BAM
 - Apesar de não ter surgido, historicamente, dessa maneira
- Para comprehendê-la, consideremos uma BAM autoassociativa
 - Em que entradas são:

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$$

- As saídas esperadas são as próprias entradas dadas (autoassociativa), ou seja:

$$\mathbf{y} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L\}$$

- Nesse caso, o número de neurônios na Camada 1 e na Camada 2 é igual

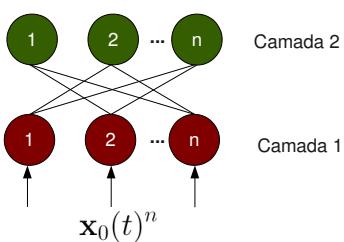


- A matrix de pesos \mathbf{w} será, então, dada por:

$$\mathbf{w} = \sum_{i=1}^L \mathbf{x}\mathbf{x}^t$$

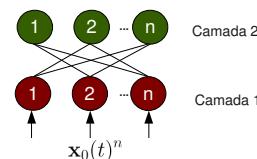
Rede Neural de Hopfield Discreta

- A Rede Neural de Hopfield Discreta ainda assume que não há pesos de feedback, ou seja:

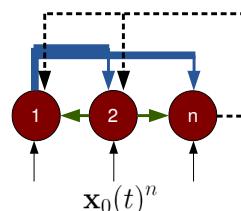


- Nem pesos de um neurônio k da Camada 1 para a Camada 2

- Como o número de neurônios em ambas camadas é igual e cada neurônio da Camada 1 e da 2 estão conectados a todos os demais, então podemos simplificar a arquitetura de:

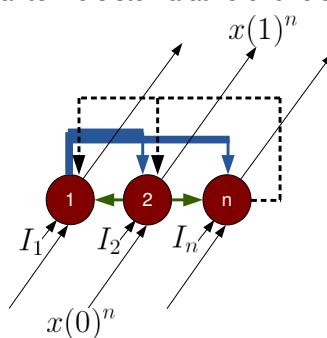


- Para:



Rede Neural de Hopfield Discreta

- No entanto, a rede neural de Hopfield Discreta ainda considera que cada neurônio recebe um conjunto de entradas ou sinais I_i
- Há um paralelo desses sinais com o refresh de memórias
 - Eles mantêm o sistema ativo e funcionando



Rede Neural de Hopfield Discreta

- Exemplo:
- Suponha os padrões aprendidos pela rede:

$$\mathbf{x}_1 = (-1, -1, 1, 1, -1, 1, 1, 1, 1, 1)^t$$

$$\mathbf{x}_2 = (1, 1, 1, 1, 1, 1, -1, -1, 1, 1)^t$$
- Obtemos a matriz de pesos da forma:

$$\mathbf{w} = \mathbf{x}_1 \mathbf{x}_1^t + \mathbf{x}_2 \mathbf{x}_2^t$$
- Haverá valores diferentes de zero na diagonal de \mathbf{w} , mas como Hopfield assume que os pesos de feedback são iguais a zero, então zeramos a diagonal

Rede Neural de Hopfield Discreta

- Exemplo:
- Aplicando:

$$\mathbf{x}(0) = (-1, -1, 1, 1, -1, 1, 1, 1, 1, 1)^t$$
- Obtemos:

$$\mathbf{x}(0)\mathbf{w} = (-8, -8, 8, 8, -8, 8, 8, 8, 8, 8)^t$$
- Aplicando a função de ativação:

$$x_i(t+1) = \begin{cases} +1 & \text{net}_i^x > 0 \\ x_i(t) & \text{net}_i^x = 0 \\ -1 & \text{net}_i^x < 0 \end{cases}$$

- Resultando em:

$$\mathbf{x}(1) = (-1, -1, 1, 1, -1, 1, 1, 1, 1, 1)^t$$
- Portanto, reconstruímos o padrão de entrada \mathbf{x}_1

Rede Neural de Hopfield Discreta

- Exemplo:
 - Plotando \mathbf{w} :
-
- BAM e Hopfield representam os padrões aprendidos em \mathbf{w}
 - Ambas redes representam os padrões e seus complementos:
 - Repare que há quatro mínimos (em vermelho) sendo que foram aprendidos 2 padrões
 - Matrix deve ter tamanho suficiente para tal representação, ou seja, padrões representados não podem ser muito curtos (número de termos)

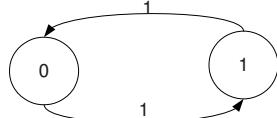
- A Rede Neural de Hopfield assume, ainda, que pode haver um vetor de sinais constantemente aplicado a seus neurônios I_i , os quais funcionam como Bias
 - Neste exemplo consideramos esse vetor preenchido com zeros
 - Hopfield Discreta também assume que a função de ativação pode ter um **threshold** U_i diferente de zero
 - No entanto, em nosso exemplo anterior, consideramos U_i igual a zero, para todo i

$$x_i(t+1) = \begin{cases} +1 & \text{net}_i > U_i \\ x_i(t) & \text{net}_i = U_i \\ -1 & \text{net}_i < U_i \end{cases}$$

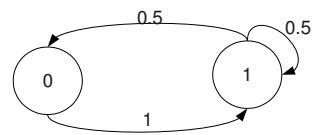
- Antes de Estudarmos a Energia produzida por essas redes:
 - Entropia é uma propriedade da Termodinâmica usada para determinar a quantidade de energia útil de um sistema qualquer
 - Gibbs afirmou que a melhor interpretação para entropia na mecânica estatística é como uma medida de incerteza
- Histórico:
 - Entropia inicia com o trabalho de Lazare Carnot (1803)
 - Rudolf Clausius (1850s-1860s) traz novas interpretações físicas
 - Claude Shannon (1948) desenvolve o conceito de Entropia em Teoria da Informação

Análise de Energia da BAM e Hopfield Discreta

- Para compreendermos a Entropia considere o seguinte sistema:



- Agora considere que o sistema sofreu uma alteração:



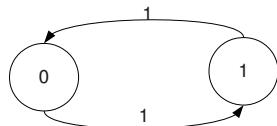
- Consideremos, agora a equação da Entropia:

$$E = - \sum_i \sum_j p_{ij} \log_2 p_{ij}$$

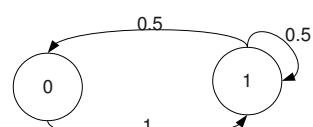
- Essa equação mede a energia total de um sistema:
 - Considerando que o sistema está no estado i e ocorre uma transição para o estado j
 - O \log_2 é usada para quantificar a Entropia em termos de bits

Análise de Energia da BAM e Hopfield Discreta

- Assim temos:



$$E = -(1 \log_2(1) + 1 \log_2(1)) = 0$$



Após modificar seu comportamento, o sistema agregou maior nível de incerteza ou energia

$$E = -(1 \log_2(1) + 0.5 \log_2(0.5) + 0.5 \log_2(0.5)) = -1.0$$

Análise de Energia da BAM e Hopfield Discreta

- Há uma conclusão interessante aqui:
 - Se um sistema altera seu comportamento, reduzindo sua energia (ou entropia), chegamos a um sistema com maior nível de certeza
 - O que ocorre com a BAM e a Hopfield?
 - Dado um vetor de entrada, queremos “reconstruí-lo” o mais próximo dos vetores que treinamos previamente
 - Para isso, essas redes:
 - Recebem vetores de entrada
 - Processam
 - Geram retorno
 - Retroalimentam esse retorno buscando convergí-lo para um dos padrões utilizados para gerar w

- Portanto:
 - Essas redes visam reduzir suas energias
 - Em busca de maior nível de “certeza” sobre um de seus padrões aprendidos em \mathbf{w}
 - Para isso **BAM** computa sua energia na forma:

$$E = - \sum_{i=1}^m \sum_{j=i}^n y_i w_{ij} x_j$$

- Ou seja:
 - Considera que o sistema está em um estado \mathbf{x} e transiciona para \mathbf{y} conforme o mapeamento (ou possíveis transições) definidas em \mathbf{w}
 - Simular o caminhamento na superfície em termos de energia

- Assim voltemos ao exemplo da **BAM**

- Entradas e saídas esperadas:

$$\mathbf{x}_1 = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \text{ e } \mathbf{y}_1 = (1, -1, -1, -1, -1, 1)^t$$

$$\mathbf{x}_2 = (1, 1, 1, -1, -1, 1, 1, -1, -1)^t \text{ e } \mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t$$

$$\mathbf{w} = \mathbf{y}_1 \mathbf{x}_1^t + \mathbf{y}_2 \mathbf{x}_2^t$$

$$\mathbf{w} = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{bmatrix}$$

- Assim voltemos ao exemplo da **BAM**:
 - Considere o vetor de entrada:
 $x(0) = (-1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$
 - E o valor inicial para os pesos da Camada 2
 $y(0) = (-1, -1, -1, -1, -1, -1)$
 - O processamento na Camada 2 gera:
 $y(1) = (1, -1, -1, -1, -1, 1)^t$
 - E o retorno:
 $x(1) = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$

- Assim voltemos ao exemplo da **BAM**:

- Computando as energias:

- A energia inicial para:

$$x(0) = (-1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$$

$$y(0) = (-1, -1, -1, -1, -1, -1)$$

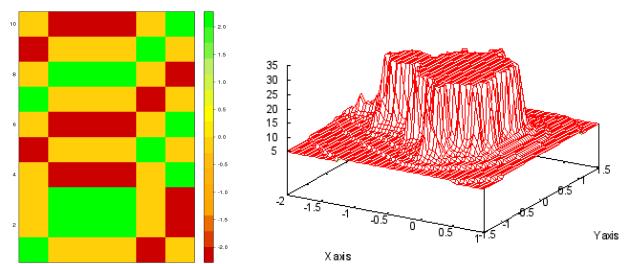
- É dada por:

$$E = - \sum_{i=1}^6 \sum_{j=1}^{10} y(0)_i \mathbf{w}_{ij} x(0)_j = -24$$

- Assim voltemos ao exemplo da **BAM**:
 - Após a retroalimentação da rede temos:
 - A energia inicial para:
 $y(1) = (1, -1, -1, -1, -1, 1)^t$
 $x(1) = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t$
 - É dada por:
 $E = - \sum_{i=1}^6 \sum_{j=1}^{10} y(1)_i \mathbf{w}_{ij} x(1)_j = -64$
 - Ou seja, reduzimos o nível de incerteza sobre nosso sistema
 - É isso que a **BAM** faz **recorrentemente** quando entregamos um vetor de entrada, até convergir

- Assim voltemos ao exemplo da **BAM**:

- Há um paralelo entre o nível de energia e os valores de \mathbf{w}
- A superfície \mathbf{w} define como caminharemos no sentido de mínimos de energia
 - Os quais são os mínimos da superfície



- Como queremos o mínimo de energia podemos implementar a BAM avaliando a derivada de Entropia:
 - Assim a BAM executa enquanto há variação de energia
- Sendo sua energia dada por:

$$E = - \sum_{i=1}^m \sum_{j=1}^n y_i w_{ij} x_j$$

- Sendo:

$$E = - \sum_{i=1}^m \sum_{j=1}^n y_i w_{ij} x_j$$

- Logo:

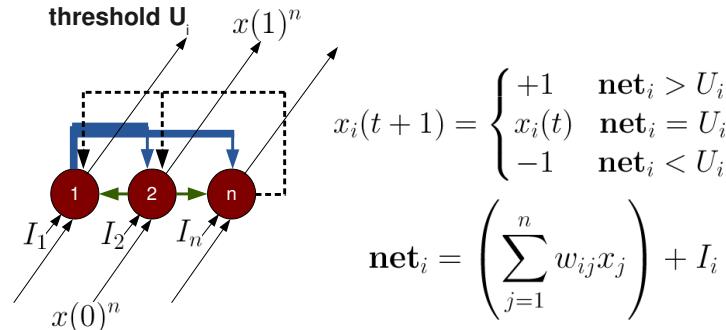
$$\frac{\partial E}{\partial t} = - \sum_{i=1}^m \sum_{j=1}^n \frac{\partial y_i}{\partial t} w_{ij} x_j = - \sum_{i=1}^m \frac{\partial y_i}{\partial t} \sum_{j=1}^n w_{ij} x_j$$

- Considerando uma entrada \mathbf{x}
- Assim temos a condição de parada, garantindo a convergência

- Exercício:
 - Implementar a BAM verificando a derivada de Entropia como critério de parada

- No caso da **Hopfield Discreta**:

- Há somente uma Camada
- Há um sinal I sendo aplicado à entrada
- A função de ativação de neurônio respeita um parâmetro de **threshold U_i**



- No caso da **Hopfield Discreta**:
 - Há somente uma Camada
 - Portanto podemos considerar que metade da energia é gerada
 - Há um sinal I sendo aplicado à entrada
 - Esse sinal precisa ser descontado da energia da rede, pois não foi produzido, mas sim definido como entrada (pode ser visto como Bias)
 - A função de ativação dos neurônios respeita um parâmetro de **threshold U_i**
 - Esse parâmetro influencia na energia produzida
 - Se o parâmetro é maior que zero, então mais energia deve ser produzida pela rede para ocorrer uma ativação
 - Se o parâmetro é menor que zero, menos energia deve ser produzida pela rede para ocorrer uma ativação

- No caso da **Hopfield Discreta**:

- Sendo:

$$\text{net}_i = \left(\sum_{j=1}^n w_{ij} x_j \right) + I_i$$

- A energia total é dada por:

$$E = -\frac{1}{2} \sum_i^n \sum_{j=1}^n y_i w_{ij} x_j - \sum_i^n I_i y_i$$

- Quando $U_i = 0$

- No caso da **Hopfield Discreta**:

- Assim para qualquer \mathbf{U}_i a Energia é dada por:

$$E = -\frac{1}{2} \sum_i^n \sum_{j=1}^n y_i w_{ij} x_j - \sum_i^n I_i y_i + \sum_i^n U_i y_i$$

- Da mesma maneira que a BAM, quando submetemos um vetor de entrada para a Hopfield Discreta:

- Processa buscando por um padrão já aprendido em \mathbf{w}
- Essa busca por um padrão similar tende a reduzir a energia produzida pela rede
- Esse processo é recursivo até não haver mais variações no padrão de entrada
- Portanto a rede tende a um estado de menor incerteza

- Como ocorre a variação de Energia na **Hopfield Discreta**?

- Se conhecemos a variação, podemos definir o critério de parada para a Hopfield Discreta

- Sendo sua Energia dada por:

$$E = -\frac{1}{2} \sum_i^n \sum_{j=1}^n y_i w_{ij} x_j - \sum_i^n I_i y_i + \sum_i^n U_i y_i$$

- Como ocorre a variação de Energia na **Hopfield Discreta**?

- Se conhecemos a variação, podemos definir o critério de parada para a Hopfield Discreta

- Sendo sua Energia dada por:

$$E = -\frac{1}{2} \sum_i^n \sum_{j=1}^n y_i w_{ij} x_j - \sum_i^n I_i y_i + \sum_i^n U_i y_i$$

- Assumindo $\mathbf{x}_j(t) = \mathbf{y}_j(t-1)$:

- Sua derivada no tempo é dada por:

$$\begin{aligned} \frac{\partial E}{\partial t} = & -\frac{1}{2} \sum_i^n \sum_j^n \frac{\partial y_i(t)}{\partial t} w_{ij} y_j - \frac{1}{2} \sum_i^n \sum_j^n y_i(t) w_{ij} \frac{\partial y_j(t-1)}{\partial t} \\ & - \sum_i^n I_i \frac{\partial y_i(t)}{\partial t} + \sum_i^n U_i \frac{\partial y_i(t)}{\partial t} \end{aligned}$$

- Como ocorre variação de Energia na **Hopfield Discreta**?

- Simplificando:

$$\begin{aligned} \frac{\partial E}{\partial t} &= - \sum_i^n \sum_j^n \frac{\partial y_i(t)}{\partial t} w_{ij} y_j - \sum_i^n I_i \frac{\partial y_i(t)}{\partial t} + \sum_i^n U_i \frac{\partial y_i(t)}{\partial t} \\ \frac{\partial E}{\partial t} &= - \sum_i^n \frac{\partial y_i(t)}{\partial t} \left[\left(\sum_j^n w_{ij} y_j \right) + I_i - U_i \right] \end{aligned}$$

- Exercício:

- Implementar a Hopfield Discreta verificando a derivada de Entropia como critério de parada

- John Hopfield buscou estender seu modelo de memória de associação discreta

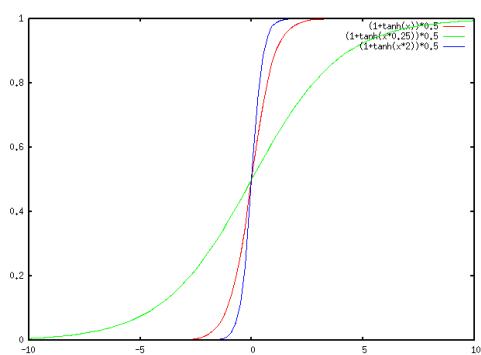
- Incorporou conceitos de neurobiologia
- Sabe-se que neurônios biológicos apresentam saídas contínuas ao invés de dois estados {+1,-1} ou {0,1}
- Na arquitetura contínua os vetores de entrada são definidos por:

$$\mathbf{u} = (u_1, u_2, \dots, u_n)^t$$

Ao invés de uma saída discreta, cada neurônio utiliza a seguinte função contínua:

$$v_i = g_i(\lambda u_i) = \frac{1}{2}(1 + \tanh(\lambda u_i))$$

- Função de ativação contínua (sigmóide):
 - Lambda é chamado parâmetro de ganho



- Aprendizado Contínuo

- Na Hopfield Discreta:

$$x_i(t+1) = \begin{cases} +1 & \text{net}_i > U_i \\ x_i(t) & \text{net}_i = U_i \\ -1 & \text{net}_i < U_i \end{cases}$$

- Na contínua

$$y_i = v_i = g_i(\lambda u_i) = \frac{1}{2}(1 + \tanh(\lambda u_i)) \quad u_i = \text{net}_i$$

- O que isso muda?

- Muda o termo \mathbf{U}_i da função de Energia

- Sendo assim:

- Derivada da Hopfield Discreta:

$$\frac{\partial E}{\partial t} = - \sum_i^n \frac{\partial y_i(t)}{\partial t} \left[\left(\sum_j^n w_{ij} y_j \right) + I_i - U_i \right]$$

- Como no caso contínuo:

$$u_i = \text{net}_i$$

- Podemos usar a função inversa da ativação para recuperá-lo

$$u_i = g_i^{-1}(y_i)$$

- Assim a forma discreta:

$$\frac{\partial E}{\partial t} = - \sum_i^n \sum_j^n \frac{\partial y_i(t)}{\partial t} w_{ij} y_j - \sum_i^n I_i \frac{\partial y_i(t)}{\partial t} + \sum_i^n U_i \frac{\partial y_i(t)}{\partial t}$$

- É reescrita como:

$$\frac{\partial E}{\partial t} = - \sum_i^n \sum_j^n \frac{\partial y_i(t)}{\partial t} w_{ij} y_j - \sum_i^n I_i \frac{\partial y_i(t)}{\partial t} + \sum_i^n g_i^{-1}(y_i) \frac{\partial y_i(t)}{\partial t}$$

- Simplificando:

$$\frac{\partial E}{\partial t} = - \sum_i^n \frac{\partial y_i(t)}{\partial t} \left[\left(\sum_j^n w_{ij} y_j \right) + I_i - u_i \right]$$

$$u_i = \text{net}_i$$

- Assim:

$$u_i = \text{net}_i = \left(\sum_j^n w_{ij} y_j \right) + I_i$$

$$\frac{\partial E}{\partial t} = - \sum_i^n \frac{\partial y_i(t)}{\partial t} \left[\underbrace{\left(\sum_j^n w_{ij} y_j \right)}_1 + I_i - u_i \right]$$

É o mesmo que $u_i(t) - u_i(t-1)$

- Dessa maneira: $\frac{\partial E}{\partial t} = - \sum_i^n \frac{\partial y_i(t)}{\partial t} \frac{\partial u_i(t)}{\partial t}$

- Derivando o primeiro termo de:

$$\frac{\partial E}{\partial t} = - \sum_i^n \frac{\partial y_i(t)}{\partial t} \frac{\partial u_i(t)}{\partial t}$$

- Temos:

$$y_i = g_i(u_i)$$

$$\frac{\partial y_i}{\partial t} = g'_i(u_i) u'_i$$

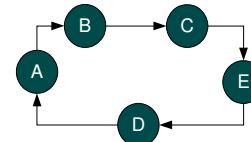
$$\frac{\partial E}{\partial t} = - \sum_i^n g'_i(u_i) \left(\frac{\partial u_i(t)}{\partial t} \right)^2$$

- Logo, a estabilidade é encontrada quando: $\frac{\partial E}{\partial t} = 0$

- O qual é o critério de parada para a Hopfield Contínua

- Sobre Hopfield Contínua:
 - Quando ela é usada?
 - Pode-se usar para mapear entradas e saídas
 - Memória associativa, tal como a BAM e a Hopfield Discreta
 - Mas é mais utilizada em problemas de Otimização
- Como implementar para resolver um dado problema?
 - Precisamos definir a função de Energia para o problema
 - Assim definimos a superfície w

- Para problemas de otimização conhecemos apenas:
 - As condições mínimas para aceitar ou rechaçar soluções
 - Por exemplo:
 - Caixeiro Viajante (ou Caminho ou Circuito Hamiltoniano)
 - Exemplo de solução válida:



- Como devemos buscar por soluções válidas e que reduzam o percurso

- A solução ideal do Caixeiro Viajante deveria avaliar todas as possibilidades
 - $O(n!)$ possibilidades
 - Em que n é o número de cidades

- Começamos pela codificação das soluções:

Ordem de visita

1	2	3	4	5
---	---	---	---	---

$$v = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{array}{l} A \\ B \\ C \\ D \\ E \end{array}$$

Cidade visitada

- Condições para uma solução ser válida:
 - Somente uma cidade pode ser visitada em dado instante
 - Cada cidade pode ser visitada apenas uma vez
 - Devemos voltar para a cidade da qual partimos
- Condição complementar:
 - É desejável reduzir, ao máximo, a distância percorrida

- Condições para uma solução ser válida:
 - **Como definimos isso em termos de Energia?**
 - Lembrar que objetivo é buscar menor energia (maior nível de certeza sobre algo)
 - Somente uma cidade pode ser visitada em dado instante

$$\sum_{X=1}^n \sum_{Y=1, Y \neq X}^n v_{Xi} v_{Yi}$$

i representa uma coluna da matrix, ou seja, ordem de visita

- Condições para uma solução ser válida:
 - Cada cidade pode ser visitada apenas uma vez

$$\sum_{i=1}^n \sum_{j=1, i \neq j}^n v_{Xi} v_{Xj}$$

X representa uma linha da matrix, ou seja, cidade

- Terá valor maior que 0 somente se a mesma cidade foi visitada mais de uma vez, caso contrário será zero

- Terá valor maior que 0 somente se na mesma posição mais de uma cidade foi visitada, caso contrário será zero

- Condições para uma solução ser válida:
 - Devemos voltar para a cidade da qual partimos
 - Isso assumimos na codificação da solução e no momento de computar a distância percorrida

Ordem de visita
1 2 3 4 5

$$v = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad \begin{array}{l} A \\ B \\ C \\ D \\ E \end{array}$$

Cidade visitada

- Condição complementar:
 - É desejável reduzir, ao máximo, a distância percorrida
- $$\sum_{Y=1, Y \neq X}^n \sum_{i=1}^n d_{XY} v_{Xi} (v_{Y,i+1} + v_{Y,i-1})$$

- Considerando a saída de uma cidade X
 - d_{XY} é a distância da cidade X para Y

- Assim a função de energia total é dada por:
 - John Hopfield considera uma condição a mais
 - Aumenta energia se houver mais de n visitas na matrix de codificação das cidades

$$E = +\frac{P_A}{2} \sum_{X=1}^n \sum_{i=1}^n \sum_{j=1, j \neq i}^n v_{Xi} v_{Xj} + \frac{P_B}{2} \sum_{i=1}^n \sum_{X=1}^n \sum_{Y=1, Y \neq X}^n v_{Xi} v_{Yi} \\ + \frac{P_C}{2} \left[\left(\sum_{X=1}^n \sum_{i=1}^n v_{Xi} \right) - n \right]^2 + \frac{P_D}{2} \sum_{X=1}^n \sum_{Y=1, Y \neq X}^n \sum_{i=1}^n d_{XY} v_{Xi} (v_{Y,i+1} + v_{Y,i-1})$$

- Outro problema é definir os valores para os parâmetros
 - $P_A = P_B = P_D = 500$
 - $P_C = 200$

- A condição de parada é a variação de Entropia
 - Logo precisamos derivar
- A atualização de u_i é dada por (lembrando que $u_i = \text{net}_i$):

$$u_i(t+1) = u_i(t) + \Delta u_i$$

- Sabendo que a equação de Entropia é definida em termos de u_i logo podemos concluir:

$$\frac{\Delta u_i}{\Delta t} = -P_A \sum_{j=1, j \neq i}^n v_{Xj} - P_B \sum_{Y=1, Y \neq X}^n v_{Yi} \\ - P_C \left(\sum_{X=1}^n \sum_{j=1}^n v_{Xj} - n' \right) \\ - P_D \sum_{Y=1, Y \neq X}^n d_{XY} (v_{Y,i+1} + v_{Y,i-1}) - u_{Xi}$$

- Assim:

$$\Delta u_i = [-P_A \sum_{j=1, j \neq i}^n v_{Xj} - P_B \sum_{Y=1, Y \neq X}^n v_{Yi} \\ - P_C \left(\sum_{X=1}^n \sum_{j=1}^n v_{Xj} - n' \right) \\ - P_D \sum_{Y=1, Y \neq X}^n d_{XY} (v_{Y,i+1} + v_{Y,i-1}) - u_{Xi}] \Delta t$$

- E temos a regra de atualização para u_i :

$$u_i(t+1) = u_i(t) + \Delta u_i$$

- Em Resumo:

- A função de Energia permite compreendermos a superfície que faz o mapeamento de entradas e saídas
 - De maneira simples, ela representa a superfície w
- Ao derivarmos a função de energia podemos "caminhar" sobre a superfície até não haver mais alterações em:

$$\Delta u_i$$

- Assim sabemos que convergimos para um mínimo de energia, o qual representa uma boa solução
- No entanto, ao mudar os pesos que damos aos termos da função de energia, esses mínimos podem ficar melhores ou piores
- Precisamos definir bem quais termos serão considerados na função de energia, pois eles alteram o comportamento da superfície w

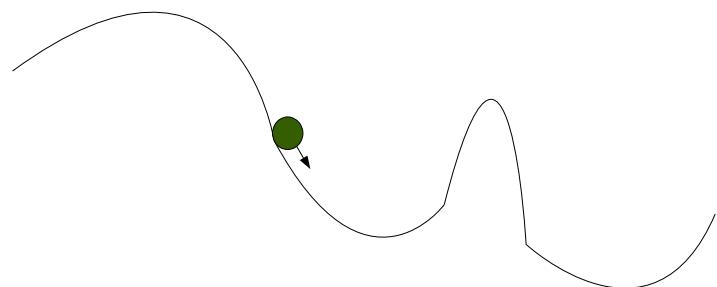
Rede Neural de Hopfield Contínua

- Implementar o Caixeiro Viajante usando Hopfield Contínua

Boltzmann Machine

- BAM e Hopfield apresentam um problema em comum:

- Retroalimentação dessas redes neurais busca por um mínimo local

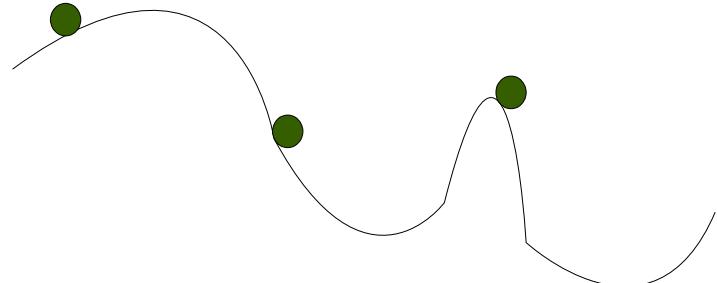
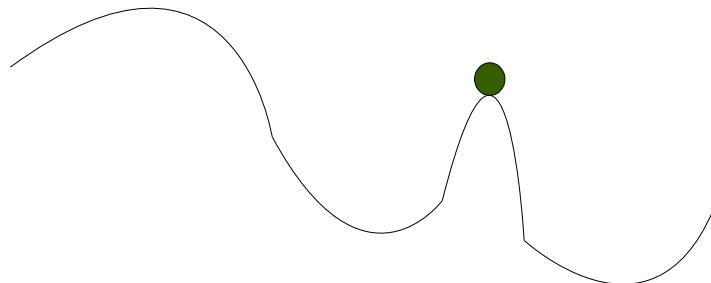


Boltzmann Machine

- BAM e Hopfield apresentam um problema em comum:
 - Se a “bola” estiver exatamente no pico???
 - Por isso há certas entradas que podem fazer com que BAM e Hopfield fiquem “travadas” e não busquem por um mínimo

Boltzmann Machine

- Máquina de Boltzmann agraga probabilidades
 - Permite “varrer” melhor a superfície
 - Isso se dá por meio do arrefecimento simulado
 - Simulated Annealing

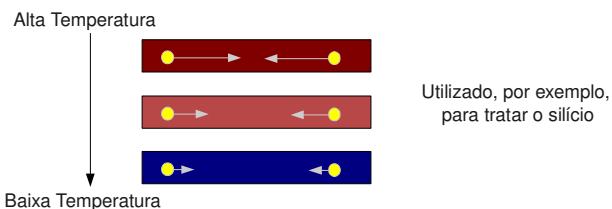


Boltzmann Machine

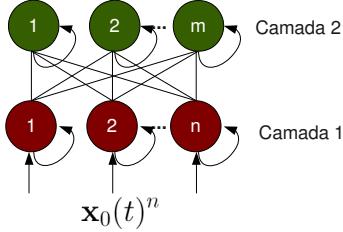
- O que Simulated Annealing?
 - Baseado no Conceito de Annealing da Metalurgia
 - Aplica-se uma energia sobre um corpo na forma de calor
 - Quando aquecido, as moléculas desse corpo podem se mover e reorganizar a matéria
 - Reduz-se a energia (calor) aplicada aos poucos e aguarda-se, novamente, a reorganização das moléculas

Boltzmann Machine

- O que Simulated Annealing?
 - Como visualizar em termos da superfície?
 - Em um primeiro momento, quando energia alta, aceitase soluções piores com grande facilidade
 - Conforme energia é reduzida
 - Focamos em uma região e a exploramos melhor
 - Por que isso é bom?
 - Pois permite “varrer” melhor o espaço de busca ou superfície que define a qualidade das soluções



- Como a Máquina de Boltzmann permite isso?
- Arquitetura similar à BAM e Hopfield
 - Definimos uma entrada x definida como $\{-1, +1\}$ ou $\{0, 1\}$
- Camada 1 contém neurônios visíveis
- Camada 2 contém neurônios escondidos
 - A segunda camada é ativada de maneira estocástica



- Para melhor compreender, consideremos uma Máquina de Boltzmann treinada para os seguintes vetores:

$$\mathbf{x}_1 = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \text{ e } \mathbf{y}_1 = (1, -1, -1, -1, -1, 1)^t$$

$$\mathbf{x}_2 = (1, 1, 1, -1, -1, 1, 1, -1, -1)^t \text{ e } \mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t$$

- Temos os pesos, tal como na BAM e Hopfield:

$$\mathbf{w} = \mathbf{y}_1 \mathbf{x}_1^t + \mathbf{y}_2 \mathbf{x}_2^t$$

$$\mathbf{w} = \begin{bmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{bmatrix}$$

- Agora considere um vetor de entrada:

$$x(0) = (1, -1, -1, 1, -1, 1, 1, -1, -1, -1)$$

- E qualquer vetor de saída aleatório, tal como:

$$y(0) = (1, -1, -1, -1, -1, -1)$$

- Na Máquina de Boltzmann:

- Alguns dos neurônios (um número n qualquer entre 1 e o número total de neurônios na Camada 2) da camada escondida são aleatoriamente escolhidos para computar net
- Considere que somente o neurônio 6 da Camada 2 foi escolhido, então:

$$\text{net}_6 = \sum_{i=1}^{10} w[6, i]x(0) = 8$$

- Na Máquina de Boltzmann:

- Aplica-se, então, a função de ativação para esse neurônio:

$$\text{net}_6 = \sum_{i=1}^{10} w[6, i]x(0) = 8$$

- A função de ativação é dada por:

$$P(y_i = +1) = \frac{1}{1 + e^{-\text{net}/T}}$$

- Logo:

$$\begin{aligned} \text{if } z_i \leq P(y_i = +1) \\ y_i = +1 \\ \text{else} \\ y_i = -1 \end{aligned}$$

z é um número aleatório entre [0,1]

- Na Máquina de Boltzmann:

- Aplica-se, então, a função de ativação para esse neurônio para $T=10$ (temperatura inicial):

$$\text{net}_6 = \sum_{i=1}^{10} w[6, i]x(0) = 8$$

$$\text{Suponha:}$$

$$z_6 = 0,2$$

- Na Máquina de Boltzmann:

- Retroalimentando $y(1)$ para gerar $x(1)$, temos:

$$y(1)w = (4, -8, -8, 8, -4, 8, 4, -8, -4, 8)$$

- Logo:

$$x(1) = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)$$

- Logo:

$$y_6 = +1$$

- Reconstruímos, assim o vetor de entrada x_1

$$\mathbf{x}_1 = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \text{ e } \mathbf{y}_1 = (1, -1, -1, -1, -1, 1)^t$$

$$\mathbf{x}_2 = (1, 1, 1, -1, -1, 1, 1, -1, -1)^t \text{ e } \mathbf{y}_2 = (1, 1, 1, 1, -1, -1)^t$$

- Assim: $y(0) = (1, -1, -1, -1, -1, -1)$

$$y(1) = (1, -1, -1, -1, -1, 1)$$

- Pode-se, reduzir a temperatura T e continuar com o processamento (arrefecimento)

– Neste caso simples, já convergiu na primeira iteração

- Na Máquina de Boltzmann:

- Geralmente, reduz-se a temperatura utilizando a função:

$$T(\text{iteração}) = \frac{T_0}{1 + \ln(\text{iteração})}$$

- Implementação

- Vamos implementar a Máquina de Boltzmann para reconhecer um conjunto de padrões de entrada
- Podemos usar as saídas iguais às entradas
 - Memória associativa

- Outra situação:

- Suponha que não tenhamos, ainda, a matrix de pesos
- Portanto, precisamos realizar o treinamento da Máquina de Boltzmann para obter w
 - Para isso podemos usar duas abordagens:
 - Least-Mean-Squared Learning (LMS)
 - Aprendizado baseado em probabilidades

- Algoritmo do Aprendizado baseado em LMS:

- Para cada época
- Para cada vetor de entrada
- Enquanto ($T > \text{temperatura_minima}$)
 - Aplique um vetor de entrada $x(0)$ na Boltzmann
 - A Boltzmann irá selecionar, aleatoriamente neurônios da Camada 2 para processar $x(0)$ e gerar $y(1)$
 - Retroalimente $y(1)$ para gerar $x(1)$
 - Reduza temperatura T
 - Considere $x(1)$ como entrada e $y(1)$ como saída
- Calcule os erros:
 - $y_{\text{error}} = y_{\text{esperado}} - y_{\text{obtido}}$
 - $x_{\text{error}} = x_{\text{esperado}} - x_{\text{obtido}}$
- Atualize $w = w + \eta(y_{\text{error}} \ x_{\text{esperado}}^t)$
 $w = w + \eta(y_{\text{esperado}} \ x_{\text{error}}^t)$

- Implementação
- Aprendizado LMS para a Máquina de Boltzmann

- Há outra técnica que emprega os mesmos conceitos:

- Simulated Annealing
 - Também inspirada do tratamento térmico que permite reorganização molecular
 - Essa reorganização é usada em metalurgia para levar o material a um estado mais estável
 - Remover tensões internas do material
 - Kirkpatrick (1983)
 - Observou que o algoritmo de Metropolis (1953), usado para simular esse tratamento térmico, poderia, também, ser utilizado como uma meta-heurística de otimização
 - Heurística – oferece boas soluções para um problema específico sem prova de corretude
 - Meta-heurística – oferece boas soluções para diversos problemas

Simulated Annealing

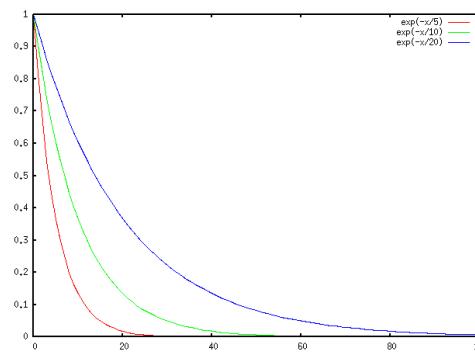
Algorithm 2 Simulated Annealing Algorithm

```

1: {Input: initial temperature  $T_0$ , learning rate  $\alpha$ , cooling
   schedule function  $\phi(k)$  and minimal number of iterations
    $k_{min}$ }
2: {Output:  $s$ }
3:  $k = 0, T = T_0$ 
4: randomly defines initial solution  $s$ 
5: while ( $T > 0$ ) do
6:   {system is warm}
7:   define a number of evaluations  $nrep$  for the current  $T$ 
8:   for each  $nrep$  do
9:     generate a new solution  $s'$ 
10:    compute  $\Delta f = f(s') - f(s)$ 
11:    if  $\Delta f < 0$  then
12:       $s = s'$ 
13:    else
14:      generate a random value  $\mu$  in range  $[0, 1]$ 
15:      if  $\exp(-\frac{f(s') - f(s)}{T}) > \mu$  then
16:         $s = s'$ 
17:      end if
18:    end if
19:   end for
20:   decrease the temperature ( $T = \phi(k)$ )
21:    $k = k + 1$ 
22: end while

```

- $f(s)$ é uma função que mede o custo da solução s
- Quanto maior o custo, pior a solução
- No entanto, quando em alta temperatura:
 - Dá-se chance para a escolha de soluções piores
 - Isso tende a auxiliar que nos retiremos de mínimos locais
- Observe que a função exponencial é muito parecida com a utilizada na Máquina de Boltzmann



- Repare que quanto menor a temperatura, mais rápida a queda da função
- Portanto, se iniciamos em baixa temperatura, o algoritmo executa por menos iterações
- Podemos reduzir a temperatura, da mesma forma que em Boltzmann

$$T(\text{iteração}) = \frac{T_0}{1 + \ln(\text{iteração})}$$

Simulated Annealing

- Mas de onde surgiu a idéia dessa função?

$$T(\text{iteração}) = \frac{T_0}{1 + \ln(\text{iteração})}$$

- Conceito da Mecânica Estatística:

- Ramo da Física que estuda sistemas compostos por um grande número de partículas
- Uma mudança em uma partícula, não afeta o sistema como um todo

- Exemplo: Lago e Garrafas



<http://ja.fotopedia.com/items/flickr-4013276093>



<http://suttonhoo.blogspot.com/2009/02/gulp.html>

Simulated Annealing

- Implementação
- Problemas de Otimização para resolver:
 - Caixeiro Viajante
 - Escalonamento
 - Tarefas, Processos, etc.

Simulated Annealing

- Proposta por Kohonen

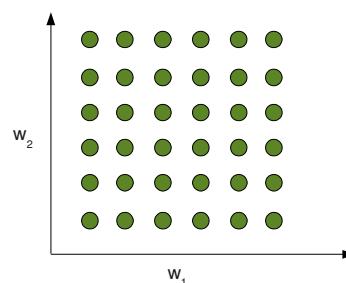
- Baseada no contexto do cérebro humano
 - Cérebro tem áreas com funções específicas
 - Neurônios biológicos vizinhos “respondem” de maneira similar a determinadas situações (entradas)

- SOM:

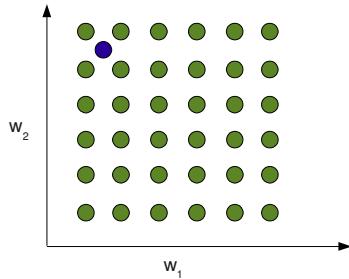
- Aprendizado não supervisionado
 - Não há um conjunto de saídas esperado
 - A técnica busca representar as entradas
- Enquanto:
 - Aprendizado supervisionado é usado para Classificação de padrões
 - Aprendizado não supervisionado é utilizado para agrupamento de padrões

- Neurônios recebem pesos aleatórios

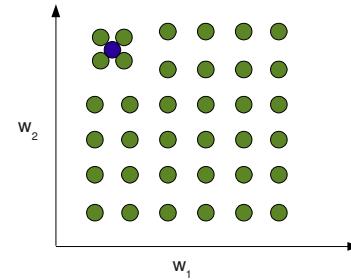
- Consideremos uma plana R^2 para isso, portanto vetor de pesos de cada neurônio tem 2 elementos



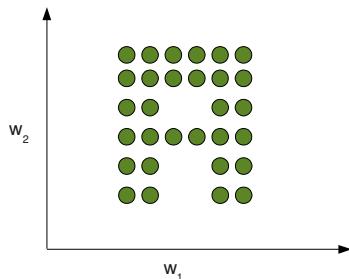
- Suponhamos que um vetor de entrada foi submetido à SOM
 - Como ocorre o treinamento?



- Suponhamos que um vetor de entrada foi submetido à SOM
 - Neurônios se movem para perto do padrão de entrada

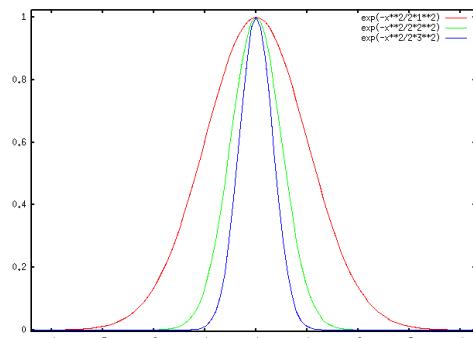


- Suponhamos que um vetor de entrada foi submetido à SOM
 - Após várias iterações poderíamos ter algo como



- Quais equações utilizamos para treinamento?
 - Permite verificar a ativação de um padrão de entrada em relação a um neurônio

$$t_i(x) = e^{-\frac{\vec{x} - \vec{w}_i}{2\sigma^2}}$$



- Quais equações utilizamos para treinamento?
 - Usamos a ativação para adaptar neurônios
- $$\vec{w}_i(t+1) = \vec{w}_i(t) + \eta t_i(\vec{x})(\vec{x} - \vec{w}_i)$$
- w representa o vetor de pesos
 - x é o vetor de entrada
- Podemos usar a SOM em dois estágios:
 - Primeiro – modelar vetores de entrada (conjunto de treinamento)
 - Segundo – Para um conjunto de testes, podemos indicar qual neurônio é o que melhor representa a entrada

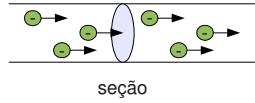
- Implementação:
 - Problemas possíveis:
 - Representação de caracteres
 - Agrupamento para quantização de áudio
 - Uso em Codecs de áudio e Voip

- Freeman and Skapura, Neural Networks: Algorithms, Applications and Programming Techniques, Addison-Wesley, 1991
- HAYKIN, S. "Neural Networks. A Comprehensive Foundation", New Jersey, Prentice Hall, 2 ed., 1999
- BRAGA,A.; CARVALHO, A.; LUDERMIR, T. "Redes Neurais Artificiais: Teoria e Aplicações", Livro Técnico e Científico, Rio de Janeiro, 2000
- Wikipedia, Neural Network
- Wikipedia, Perceptron

Rede Neural de Hopfield Contínua

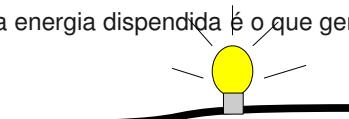
- Considera conceitos de eletricidade
 - Intensidade de corrente elétrica
 - Quantidade de carga Q (em módulo) que passa pela seção em um intervalo de tempo

$$I = \frac{\Delta Q}{\Delta t} \quad 1 \text{ Ampére} = \frac{1 \text{ Coulomb}}{1 \text{ Segundo}}$$



Rede Neural de Hopfield Contínua

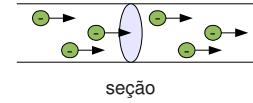
- Considera conceito de Resistência Elétrica:
 - Medida de quão difícil é para a corrente elétrica fluir por um material
 - Materiais como o cobre e o alumínio apresentam baixa resistência, ou seja, permitem que a corrente elétrica flua facilmente
 - Tungstênio tem alta resistência, por isso é usado em lâmpadas de filamento
 - Assim a eletricidade deve dispensar muita energia para sair de um ponto e chegar em outro
 - Essa energia dispendida é o que gera luz e calor



Rede Neural de Hopfield Contínua

- Considera conceitos de eletricidade
 - Intensidade de corrente elétrica
 - Quantidade de carga Q (em módulo) que passa pela seção em um intervalo de tempo

$$I = \frac{\Delta Q}{\Delta t} \quad 1 \text{ Ampére} = \frac{1 \text{ Coulomb}}{1 \text{ Segundo}}$$



- A quantidade de carga Q é dada pelo produto do número de elétrons (n) pela carga individual carregada por 1 elétron (e)

$$\Delta Q = ne$$

Rede Neural de Hopfield Contínua

- Considera conceito de Resistência Elétrica:
 - Pois membranas de neurônios apresentam tal propriedade

$$R = \frac{V}{I}$$

- Em que:
 - V é medida em volt
 - Diferença de energia entre um ponto A e outro B
 - I é medida em Ampére
 - R é medida em Ohm

- Considera conceito de Resistência Elétrica:

- Pois membranas de neurônios apresentam tal propriedade

$$R = \frac{V}{I} = \frac{\text{Volt}}{\text{Ampére}} = \frac{\frac{\text{Watt}}{\text{Segundo}}}{\frac{n \text{ Coulomb}_i}{\text{Segundo}}} = \frac{\text{Watt}}{n \text{ Coulomb}_i}$$

em que: Coulomb_i é a energia carregada por um elétron

- Watt pode ser visto como a energia gasta na realização de uma tarefa
 - Exemplo, exercício físico (bicicletas de academia)
- Logo:
 - Se há mais elétrons, a resistência é menor
 - Pois mais elementos dividem a carga para produzir a

- A Resistência Elétrica também pode ser medida como:

$$R = \rho \frac{\ell}{A}$$

- Em que:

- ρ é relativo à resistência do material
- ℓ é o comprimento do condutor em metros
- A é a área de seção do condutor

- Para Hopfield Contínua assume-se:

$$\frac{\ell}{A} = \frac{\sum_{i=1}^j R_{ij}}{1}$$

- Logo:

- O comprimento do condutor é relacionado às resistências individuais de outros neurônios
- Quanto maior o somatório de resistências individuais
 - Maior resistência para j

- Logo a resistência de um neurônio para ser disparado é dada por:

$$R_j = \rho \sum_{i=1}^j R_{ij}$$

- Assim como um neurônio “resiste” à ativação, ele também “guarda” ativações prévias de acordo com a Capacitância
- Capacitância é a energia acumulada por um corpo

$$C = \frac{Q}{V}$$

- Em que Q é medida em Coulomb e V em Volt

- Assim podemos:

$$C = \frac{Q}{V}$$

$$C = \frac{\frac{\Delta Q}{\Delta t}}{\frac{\Delta V}{\Delta t}}$$

Logo:

$$\frac{C}{\Delta t} = \frac{\frac{\Delta Q}{\Delta t}}{\frac{\Delta V}{\Delta t}} = \frac{\frac{\text{Coulomb}}{\text{Segundo}}}{\frac{\text{Volt}}{\text{Segundo}}} = \frac{\text{Coulomb}}{\text{Volt Segundo}}$$

- Sendo:

$$R = \frac{V}{I} = \frac{\text{Watt}}{\text{Coulomb}} = \frac{\text{Volt Segundo}}{\text{Coulomb}}$$

$$\frac{C}{\Delta t} = \frac{\text{Coulomb}}{\text{Volt Segundo}}$$

- Assim:

$$\frac{C}{\Delta t} = \frac{1}{R}$$

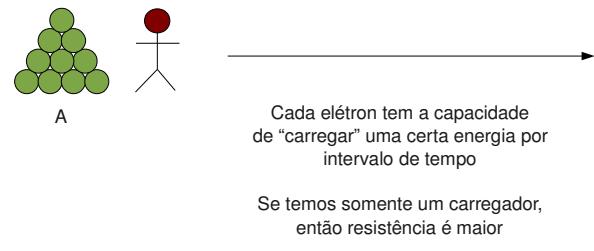
- Ou seja:

- A energia que um corpo mantém no tempo é inversamente proporcional à sua resistência

- Logo a seguinte relação vale para explicar a :

$$\frac{1}{R_j} = \frac{1}{\rho} + \frac{1}{\sum_{i=1}^n R_{ij}}$$

- De maneira prática:



Rede Neural de Hopfield Contínua

- De maneira prática:

