Tema 8

Redes Neurais Artificiais

Professora:
Ariane Machado Lima

Alguns slides baseados nos slides da Profa. Patrícia Rufino

Vídeo 1

A inspiração biológica

Introdução

- Modelos que incorporam funções matemáticas (complexas)
  - Podem ser usadas para a tarefa de regressão (aproximam uma função)

- Podem ser usadas para a tarefa de classificação:
  - tomam uma instância como entrada e produzem uma saída, que é interpretada como a classe estimada pelo modelo
  - cada categoria é dada por um número
  - ou por um intervalo de valores reais (por ex., 0.5 – 0.9)

Aprendizado de funções

- Exemplos de aprendizado de funções:
  - Amostra de treinamento = {(1,1), (2,4), (3, 9), (4, 16)}
    - Aqui, o conceito a ser aprendido é o quadrado dos números inteiros.
  - Amostra de treinamento = $\{((1,2,3)^T,1), ((2,3,4)^T,5), ((3,4,5)^T,11), ((4,5,6)^T, 19)\}$
    - Aqui, o conceito é: [a,b,c] -> a*c - b

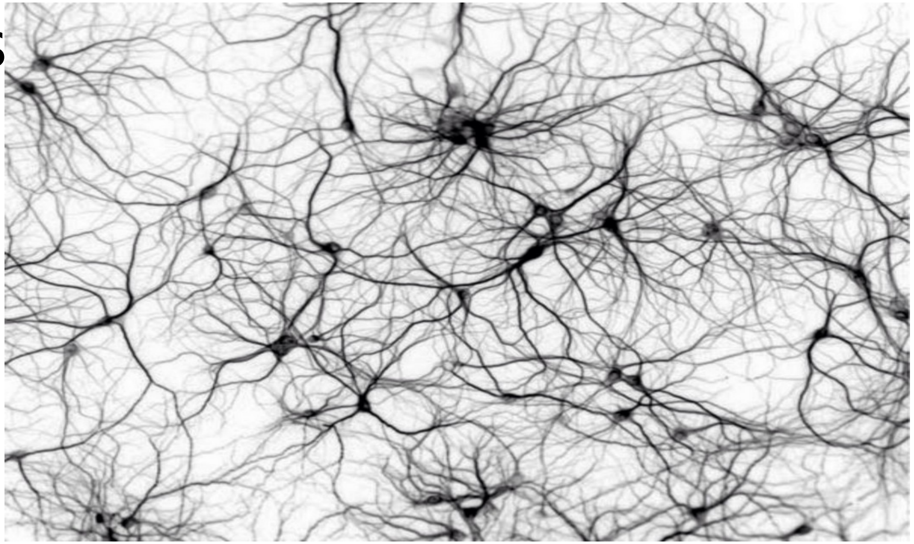
Exemplo: Classificando Veículos

- Entrada para a função: dados de pixels obtidos de imagens de veículos.
  - Saída: números: 1 para carro; 2 para ônibus; 3 para tanque

INPUT	INPUT	INPUT	INPUT
			
OUTPUT = 3	OUTPUT = 2	OUTPUT = 1	OUTPUT=1

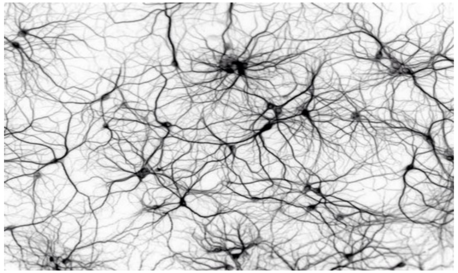
Por que usar Redes Neurais?

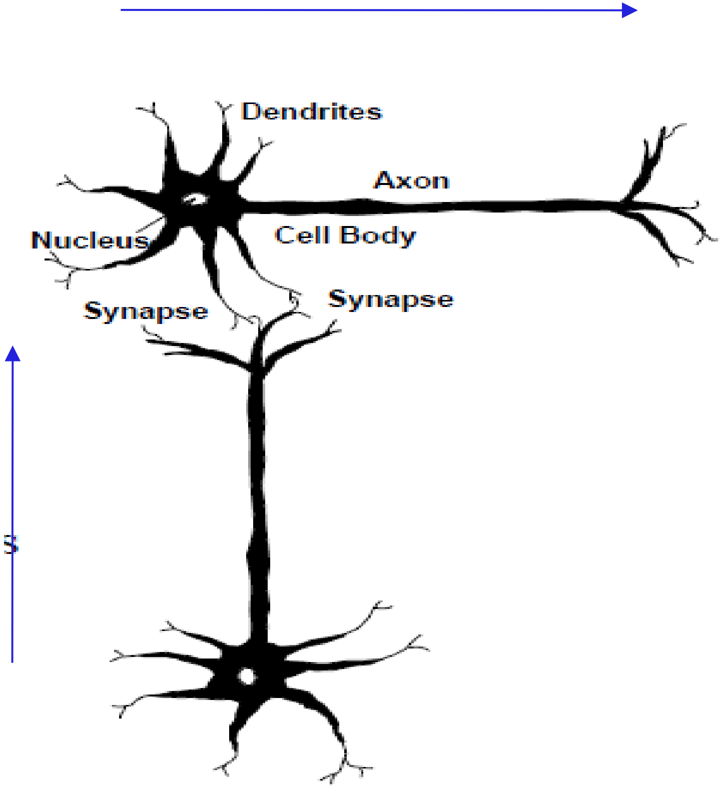
- Motivação biológica:
  - O cérebro faz com que tarefas de classificação pareçam fáceis.
  - O processamento cerebral é realizado por redes de neurônios.
  - Cada neurônio é conectado a vários outros neurônios.



Por que usar Redes Neurais?

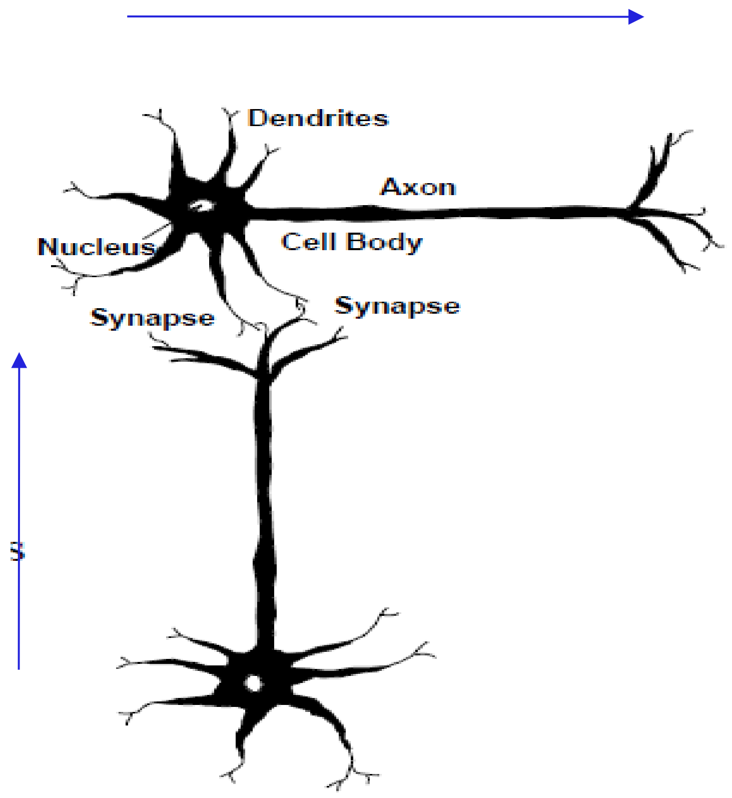
- Redes Neurais “Naturais”:
  - A entrada de um neurônio é formada pelas saídas de vários outros neurônios.
  - Um neurônio é ativado se a soma ponderada de suas entradas > limiar.





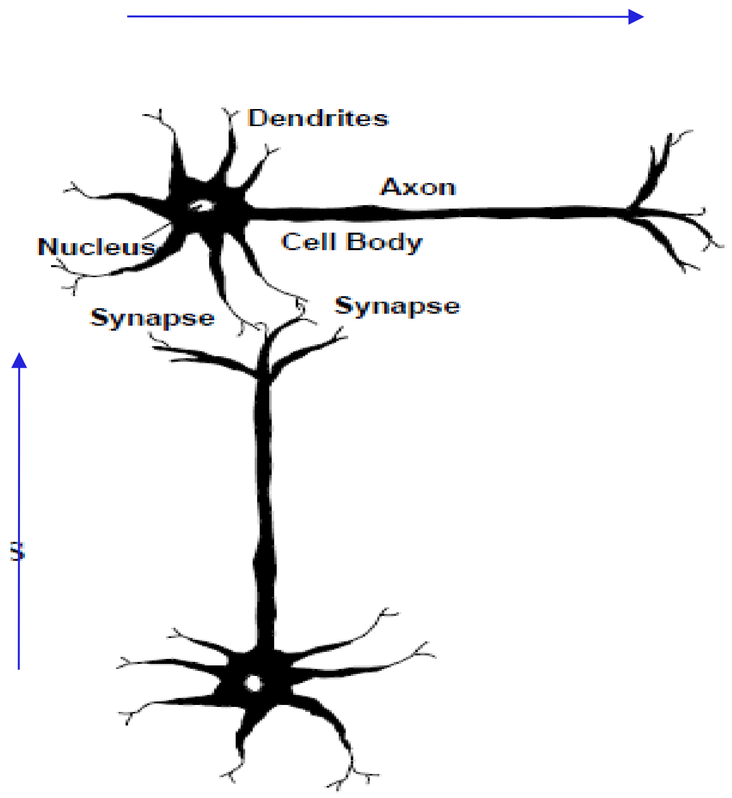
O neurônio biológico

- O neurônio recebe impulsos (sinais) de outros neurônios por meio dos seus dendritos.
- O neurônio envia impulsos para outros neurônios por meio do seu axônio.
- O axônio termina num tipo de contato chamado sinapse, que conecta-o com o dendrito de outro neurônio.



O neurônio biológico

- A sinapse libera substâncias químicas, chamadas de <u>neurotransmissores</u>, em função do pulso elétrico disparado pelo axônio.
- O neurônio envia impulsos para outros neurônios por meio do seu <u>axônio</u>.
- O fluxo de neurotransmissores nas sinapses pode ter um efeito excitatório ou inibitório sobre o neurônio receptor.



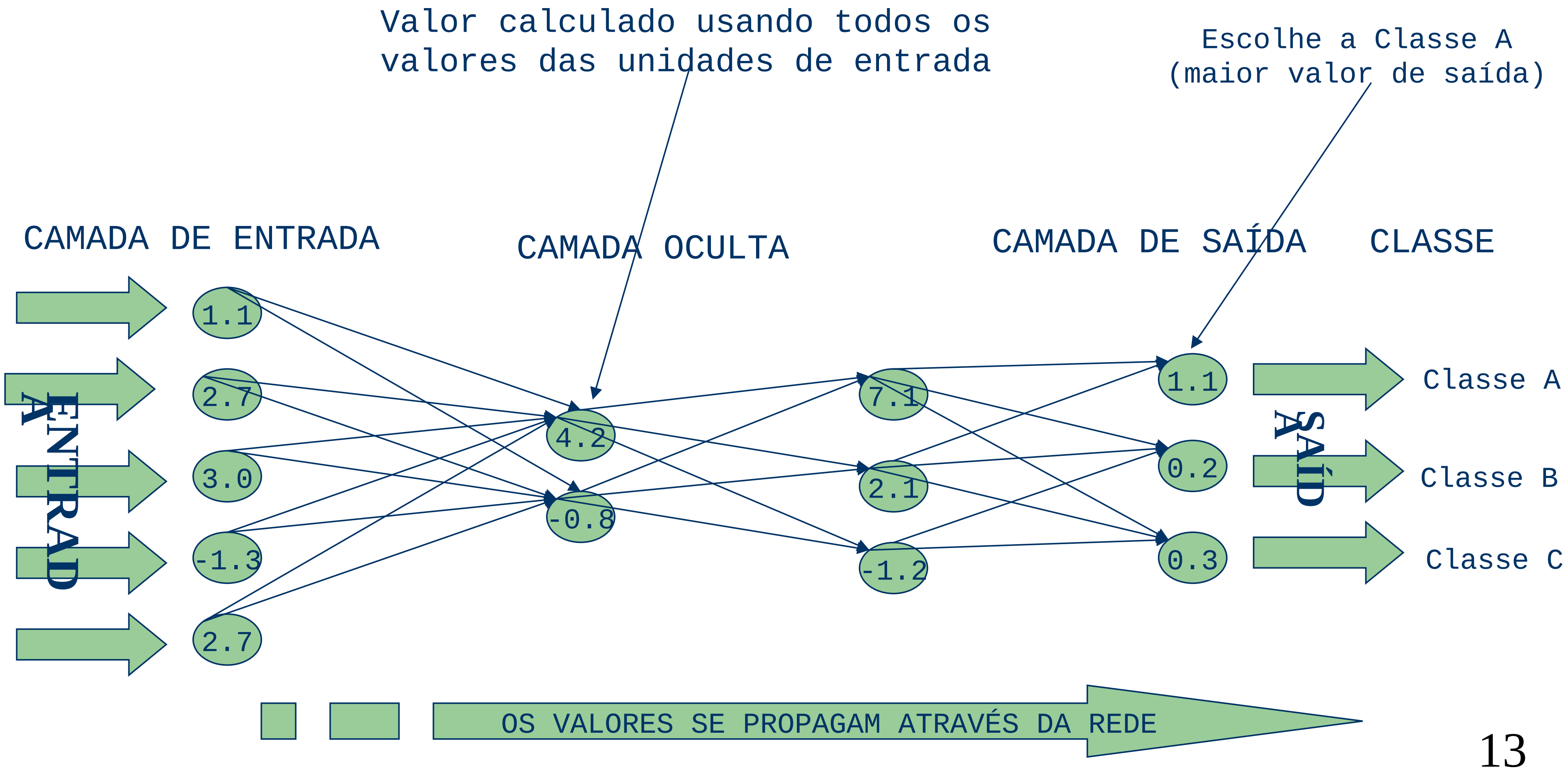
Processo de aprendizado

- O aprendizado ocorre por sucessivas modificações nas sinapses que interconectam os neurônios, em função da maior ou menor liberação de neurotransmissores.
- À medida que novos eventos ocorrem, determinadas ligações entre neurônios são reforçadas, enquanto outras enfraquecidas.
- Este ajuste nas ligações entre os neurônios durante o processo de aprendizado é uma das mais importantes características das redes neurais artificiais.

Redes Neurais Artificiais (RNAs)

- Redes Neurais Artificiais (RNAs)
  - Hierarquia similar ao funcionamento do sistema biológico.
    - Neurônios que podem ser ativados por estímulos de entrada (função de ativação)
  - Mas essa analogia não vai muito longe...
    - Cérebro humano: aproximadamente 100.000.000.000 de neurônios.
    - RNAs: < 1000 geralmente

Ideia Geral



Valor calculado usando todos os valores das unidades de entrada
Escolhe a Classe A (maior valor de saída)
CAMADA DE ENTRADA
CAMADA OCULTA
CAMADA DE SAÍDA
CLASSE
ENTRADA
1.1
2.7
3.0
-1.3
2.7
4.2
-0.8
7.1
2.1
-1.2
1.1
0.2
0.3
SAÍDA
Classe A
Classe B
Classe C
OS VALORES SE PROPAGAM ATRAVÉS DA REDE

Processamento das RNAs

- Cada unidade (da mesma camada) da rede realiza o mesmo cálculo.
  - geralmente baseado na soma ponderada das entradas na unidade.
- O conhecimento obtido pela rede fica armazenado nos pesos correspondentes a cada uma de suas unidades (neurônios).
- Representação “Caixa Preta”:
  - É difícil extrair o conhecimento sobre o conceito aprendido.

Aprendizado Supervisionado em RNAs

- Dados: conjunto de exemplos rotulados e representados numericamente.
- Tarefa: treinar uma rede neural usando esses exemplos.
  - O desempenho deve ser medido pela capacidade da rede em produzir saídas corretas para dados não contidos no conjunto de treinamento.

Aprendizado Supervisionado em RNAs

- Etapas preliminares ao treinamento:
  - escolha da arquitetura de rede correta
    - número de neurônios
    - número de camadas ocultas
  - escolha da função de ativação (a mesma para cada neurônio de uma mesma camada).
- A etapa de treinamento resume-se a:
  - ajustar os pesos das conexões entre as unidades para que a rede produza saídas corretas.

Fim do vídeo 1

A inspiração biológica

Vídeo 2

Perceptrons

Perceptrons

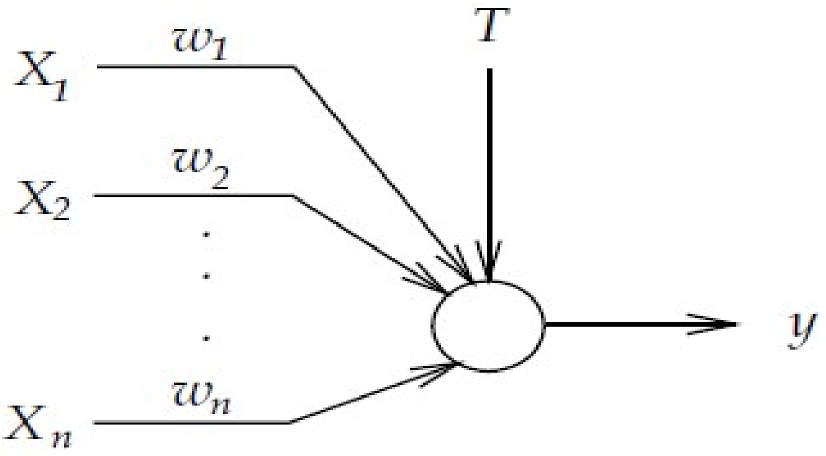
- O tipo mais simples de Rede Neural.
- Proposta por Rosenblat (1959)

Perceptrons

- O tipo mais simples de Rede Neural.
- Possui um único neurônio de saída.
  - Considera uma soma ponderada das entradas.
  - A função de ativação da unidade calcula a saída da rede.
  - Exemplo: unidade com threshold (limiar) linear.

– $Netinput = \sum_{i=1}^{n} x_i w_i$

– $if\ Netinput > T\ then\ y = 1\ else\ y = 0$

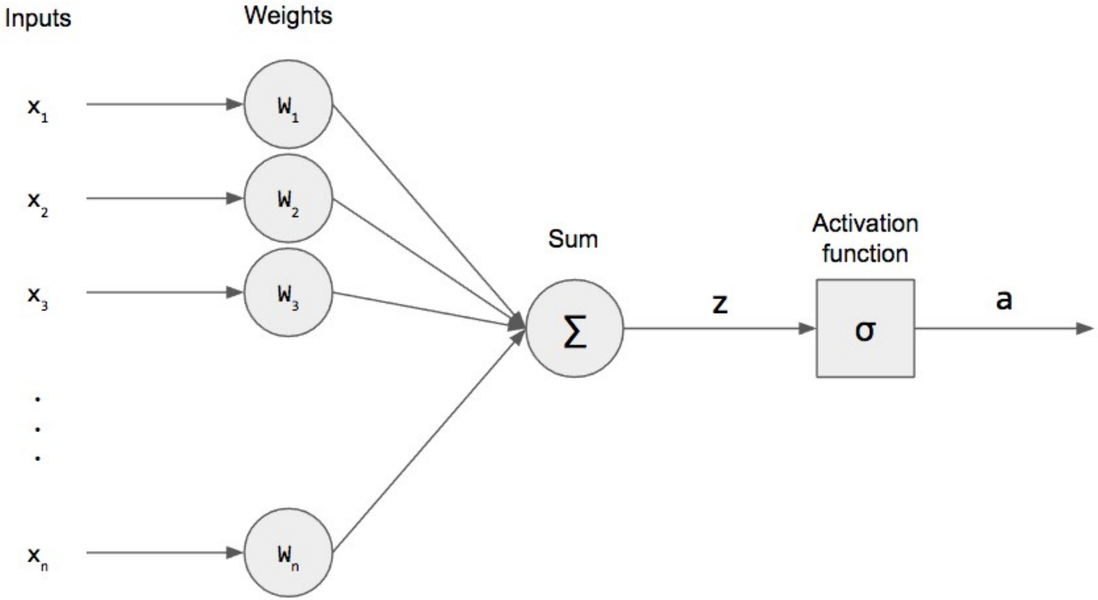


Perceptrons

- O tipo mais simples de Rede Neural.
- Possui um único neurônio de saída.
  - Considera uma soma ponderada das entradas.
  - A função de ativação da unidade calcula a saída da rede.
  - Exemplo: unidade com threshold (limiar) linear.

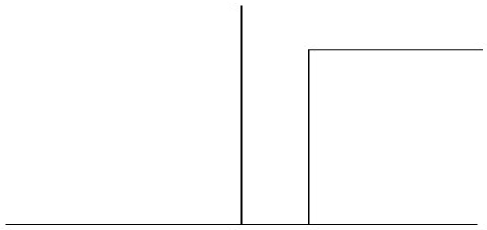
– $Netinput = \sum_{i=1}^{n} x_i w_i$

– $if\ Netinput > T\ then\ y = 1\ else\ y = 0$

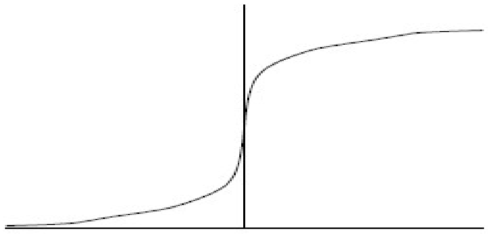


Perceptrons

- Algumas funções de ativação:



Função threshold



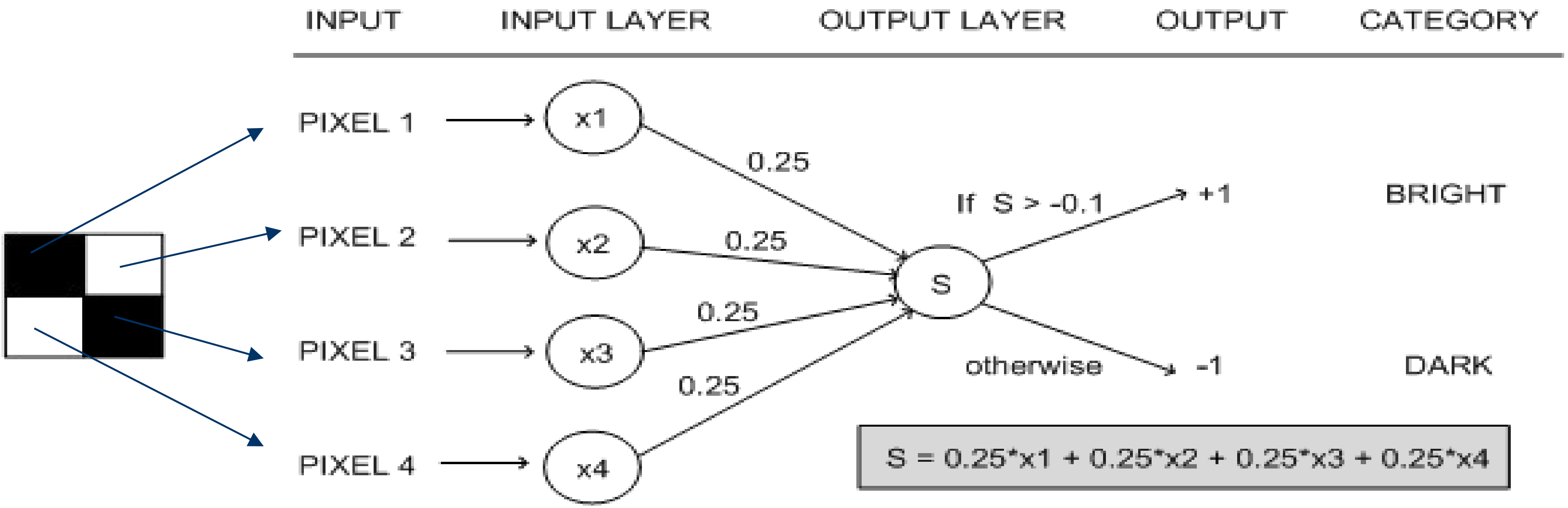
Função logística/sigmóide

- Função Step (degrau):
  - Saída +1 se *Netinput* > *Threshold* T
  - Saída –1 caso contrário
    - Aqui, os dados binários são representados por +1 e -1
- <u>Problema principal</u>: como aprender os valores dos pesos da rede?

Perceptrons: Exemplo

- Classificação de imagens preto e branco representadas por uma matriz de pixels 2x2.
  - Em “clara” ou “escura”.
  - Pixels brancos = 1 e pixels pretos = -1
- Pode-se representar o problema por essa regra:
  - Se apresentar 2, 3 ou 4 pixels brancos, é “clara”.
  - Se apresentar 0 ou 1 pixel branco, é “escura”.
- Arquitetura do Perceptron:
  - Quatro unidades de entrada, uma para cada pixel.
  - Uma unidade de saída: +1 para “clara”, -1 para “escura”.

Perceptrons: Exemplo de funcionamento (já treinada)

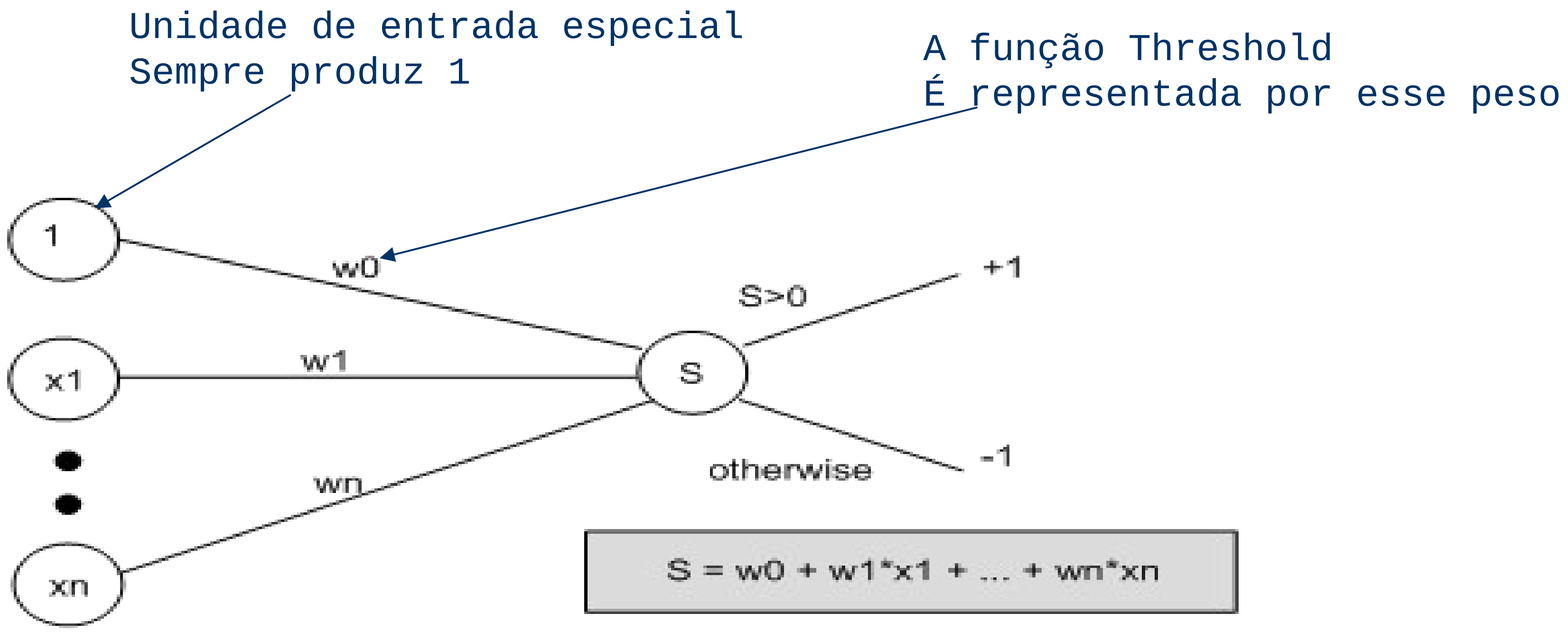


- Exemplo de entrada: x_1=-1, x_2=1, x_3=1, x_4=-1
  - S = 0.25*(-1) + 0.25*(1) + 0.25*(1) + 0.25*(-1) = 0
- 0 > -0.1, portanto a saída para a rede é +1
  - A imagem é classificada como “clara”

Aprendizagem em Perceptrons

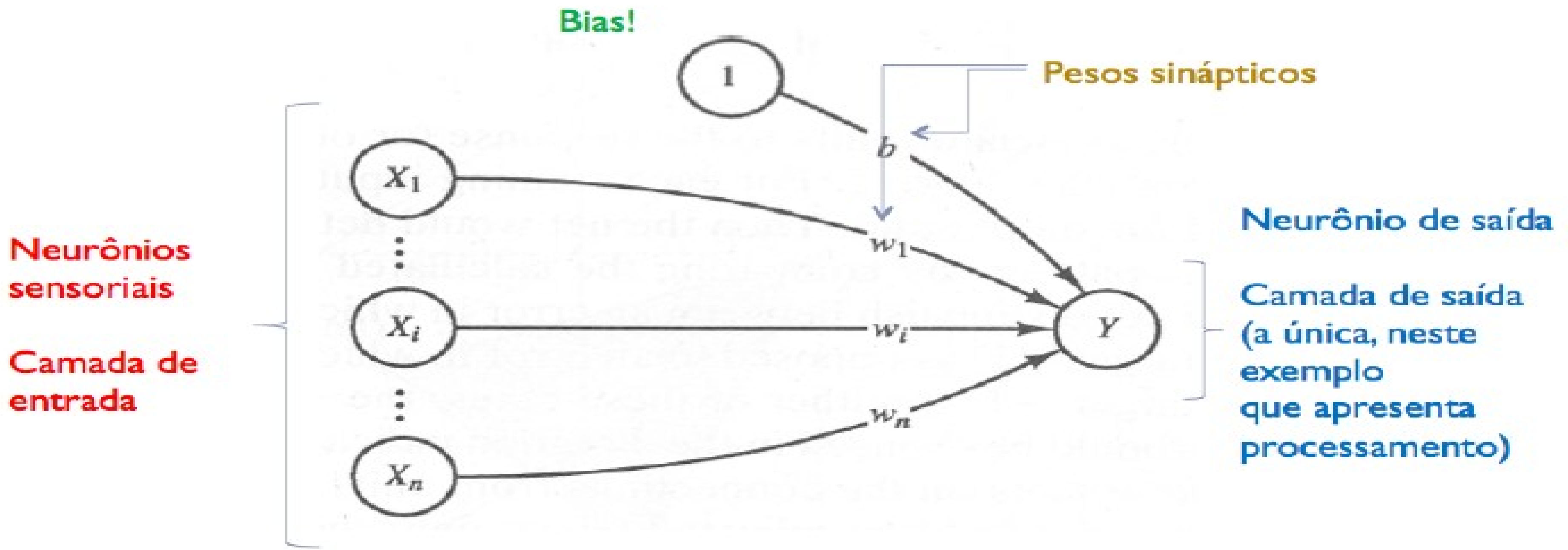
- É necessário aprender:
  - Os pesos entre as unidades de entrada e saída.
  - O valor do *threshold*.
- Para tornar os cálculos mais fáceis:
  - Considera-se o *threshold* como um peso referente a uma unidade de entrada especial, cujo sinal é sempre 1 (ou -1).
    - Agora, o único objetivo resume-se a aprender os pesos da rede.

Representação Alternativa para Perceptrons



Perceptrons

Arquitetura



Perceptrons: Algoritmo de Aprendizagem

- Os valores dos pesos são inicializados aleatoriamente, geralmente no intervalo (-1, 1).
- Para cada exemplo de treinamento E:
  - Calcule a saída observada da rede o(E).
  - Se a saída desejada t(E) for diferente de o(E):
    - Ajuste os pesos da rede, para que o(E) chegue mais próximo de t(E).
    - Isso é feito aplicando-se a regra de aprendizado do Perceptron.

Perceptrons: Algoritmo de Aprendizagem

- O processo de aprendizado não pára necessariamente depois de todos os exemplos terem sido apresentados.
  - Repita o ciclo novamente (uma “época”).
  - Até que a rede produza saídas corretas (ou boas o suficiente) – convergência.
    - Considerando todos os exemplos no conjunto de treinamento.

Regra de Aprendizagem para Perceptrons

- Quando t(E) for diferente de o(E)
  - Adicione Δ_i ao peso w_i
    - Em que $\Delta_i = \eta(t(E) - o(E))x_i$
  - Faça isso para todos os pesos da rede.

Regra de Aprendizagem para Perceptrons

Adicione Δ_i ao peso w_i em que $\Delta_i = \eta(t(E) - o(E))x_i$

- Interpretação:

Regra de Aprendizagem para Perceptrons

Adicione Δ_i ao peso w_i em que $\Delta_i = \eta(t(E) - o(E))x_i$

Interpretação:

$(t(E) - o(E))$ será igual a 0, +2 ou –2 (considerando saídas +1 ou -1 apenas)

– Portanto, pode-se pensar na adição de Δ_i como uma movimentação do peso em uma determinada direção.
  - que irá melhorar o desempenho da rede com relação a E.

Regra de Aprendizagem para Perceptrons

Adicione Δ_i ao peso w_i em que $\Delta_i = \eta(t(E) - o(E))x_i$

Interpretação:

- Multiplicação por x_i : o movimento aumenta proporcionalmente ao sinal de entrada.

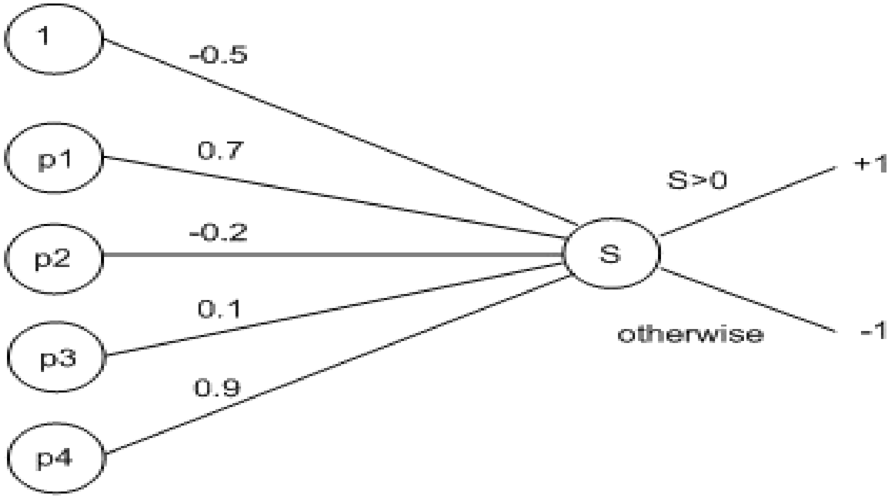
Regra de Aprendizagem para Perceptrons

Adicione Δ_i ao peso w_i em que $\Delta_i = \eta(t(E) - o(E))x_i$

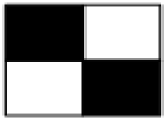
- Interpretação:
- O parâmetro η é chamado de taxa de aprendizagem.
- Geralmente escolhido como uma pequena constante entre 0 e 1 (por exemplo, 0.1).
- Controla o movimento dos pesos.
- Não permite haver uma mudança grande para um único exemplo.
- Se uma mudança grande for mesmo necessária para que os pesos classifiquem corretamente um determinado exemplo, essa deve ocorrer graduamente, em várias épocas.

Exemplo anterior: classificar uma imagem em “clara” ou “escura”

- 1) Suponha que a rede Percepton em treinamento apresente, em um dado instante de tempo, o seguinte conjunto de pesos:

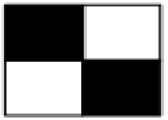


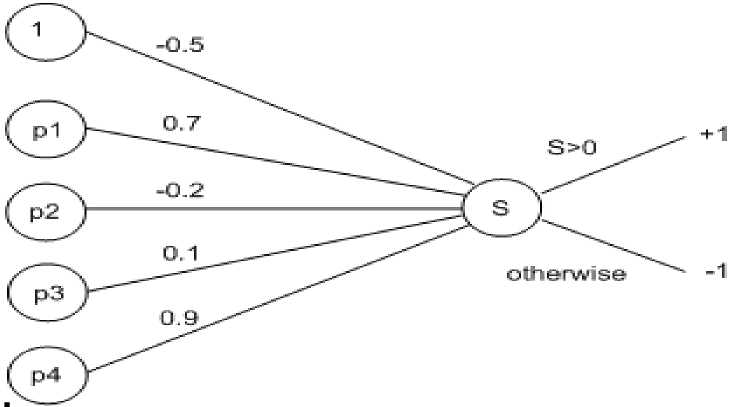
- 2) Use o exemplo de treinamento, e_1, abaixo, para atualizar os pesos da rede:



  – Use a taxa de aprendizado $\eta = 0.1$

Exemplo anterior: classificar uma imagem em “clara” ou “escura”





- <u>Solução</u>:

- Aqui, x_1=-1, x_2=1, x_3=1, x_4=-1
- Propagando essa informação através da rede:
    - S = (-0.5 * 1) + (0.7 * -1) + (-0.2 * +1) + (0.1 * +1) + (0.9 * -1) = -2.2
- Portanto, a saída da rede é $o(e_1)$ = -1 (“escura”)
- Mas deveria ter sido +1 (“clara”)
    - Portanto $t(e_1)$ = +1

Exemplo anterior: classificar uma imagem em “clara” ou “escura”

Cálculo dos valores de erro: ($t(e_1) = +1$, $o(e_1) = -1$ e $\eta = 0.1$)

- $\Delta_0 = \eta(t(E)\text{-}o(E))x_0$

  $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$
- $\Delta_1 = \eta(t(E)\text{-}o(E))x_1$

  $= 0.1 * (1 - (-1)) * (-1) = 0.1 * (-2) = -0.2$
- $\Delta_2 = \eta(t(E)\text{-}o(E))x_2$

  $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$
- $\Delta_3 = \eta(t(E)\text{-}o(E))x_3$

  $= 0.1 * (1 - (-1)) * (1) = 0.1 * (2) = 0.2$
- $\Delta_4 = \eta(t(E)\text{-}o(E))x_4$

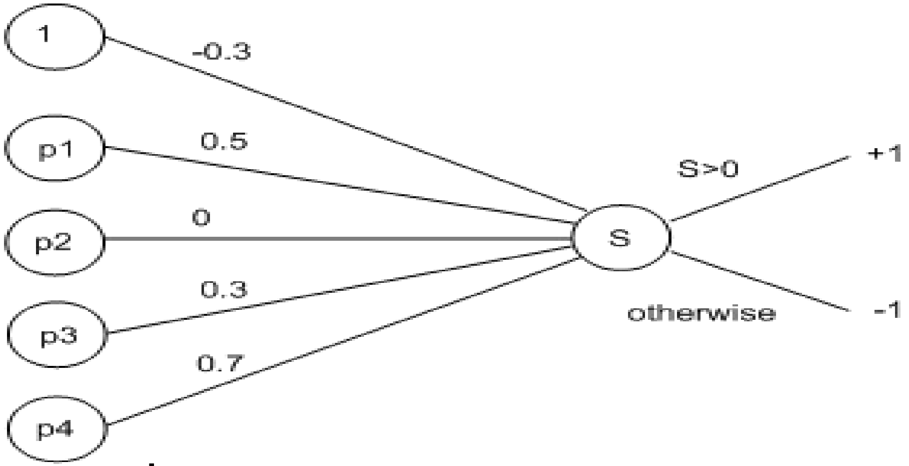
  $= 0.1 * (1 - (-1)) * (-1) = 0.1 * (-2) = -0.2$

Exemplo anterior: classificar uma imagem em “clara” ou “escura”

- <u>Ajuste dos pesos</u>:
  - $w'_0 = -0.5 + \Delta_0 = -0.5 + 0.2 = -0.3$
  - $w'_1 = 0.7 + \Delta_1 = 0.7 + -0.2 = 0.5$
  - $w'_2 = -0.2 + \Delta_2 = -0.2 + 0.2 = 0$
  - $w'_3 = 0.1 + \Delta_3 = 0.1 + 0.2 = 0.3$
  - $w'_4 = 0.9 + \Delta_4 = 0.9 - 0.2 = 0.7$

Exemplo anterior: classificar uma imagem em “clara” ou “escura”

- <u>Nova configuração da rede</u>:



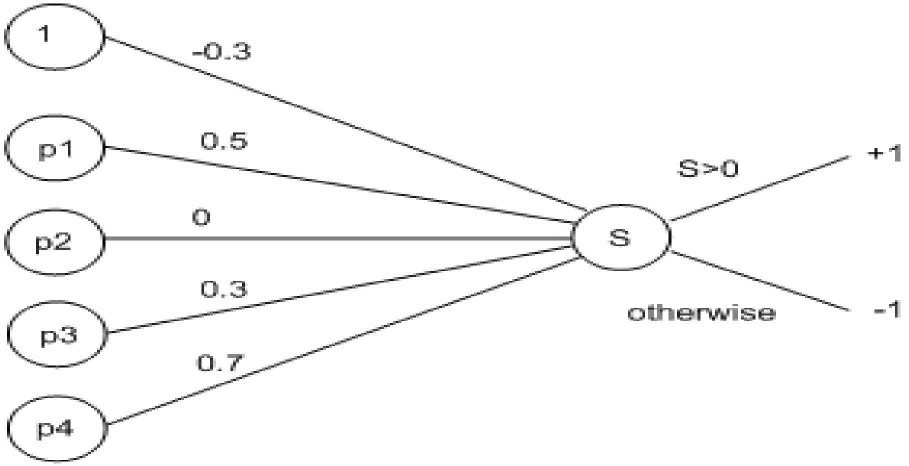
- Calcule a saída para o exemplo, e_1, novamente:

$$S = (-0.3 * 1) + (0.5 * -1) + (0 * +1) + (0.3 * +1) + (0.7 * -1) = -1.2$$

- Portanto, a nova saída da rede é o(e1) = -1 (“escura”)
- Ainda resulta em classificação errada.
- Mas o valor de S já está mais próximo de zero (de -2.2 para -1.2)
- Em poucas épocas, esse exemplo será classificado corretamente.

Exemplo anterior: classificar uma imagem em “clara” ou “escura”

- <u>Observação</u>:



- Neste exemplo: apenas 1 exemplo de treinamento
- No mundo real: cada época avalia todos os exemplos de treinamento até alcançar um critério de parada, por exemplo:
  - Somente um baixo nr de exemplos ainda é erroneamente classificado
  - Os valores de Δ_i são muito pequenos
  - Número máximo de épocas

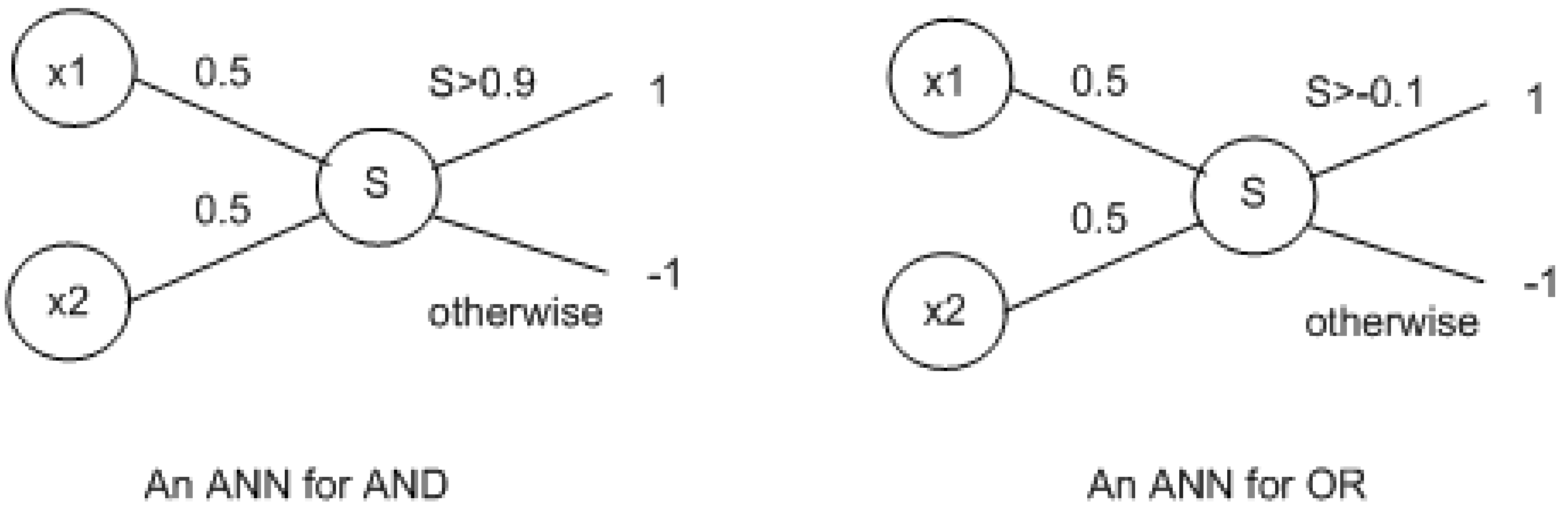
Exemplo: Aprendizado de Funções Booleanas

- Entradas assumem dois valores posssíveis (+1 ou -1).
- Produz um valor como saída (+1 ou -1).

  – Exemplo 1: Função AND
    - Produz +1 somente se ambas as entradas forem iguais a +1.

  – Exemplo 2: Função OR
    - Produz +1 se pelo menos uma das entradas for igual a +1.

Exemplo: Aprendizado de Funções Booleanas



Capacidade de Aprendizado da Rede Perceptron

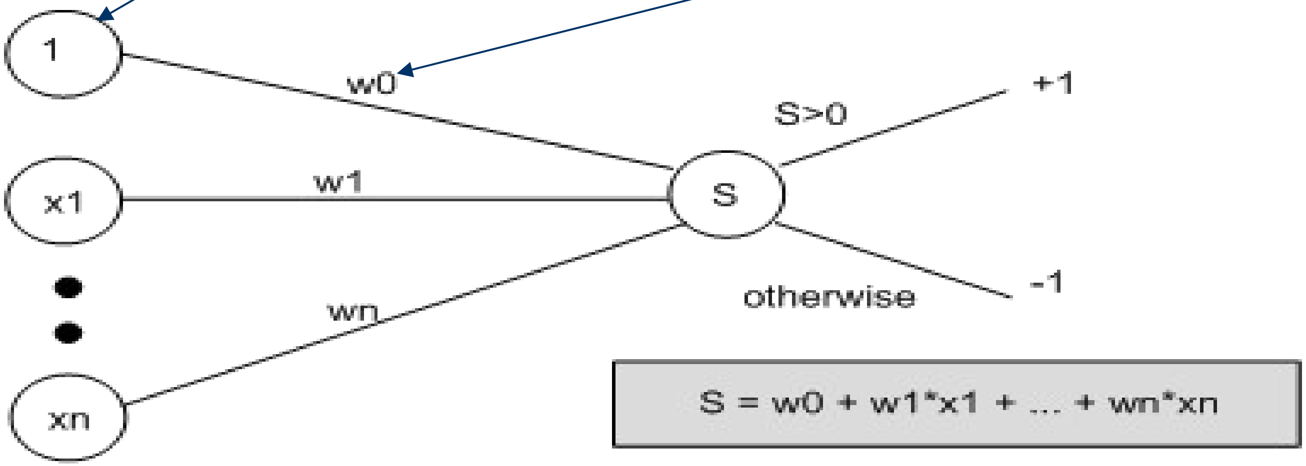
- O que a rede neural Perceptron é capaz de aprender?

Capacidade de Aprendizado da Rede Perceptron

- O que a rede neural Perceptron é capaz de aprender?

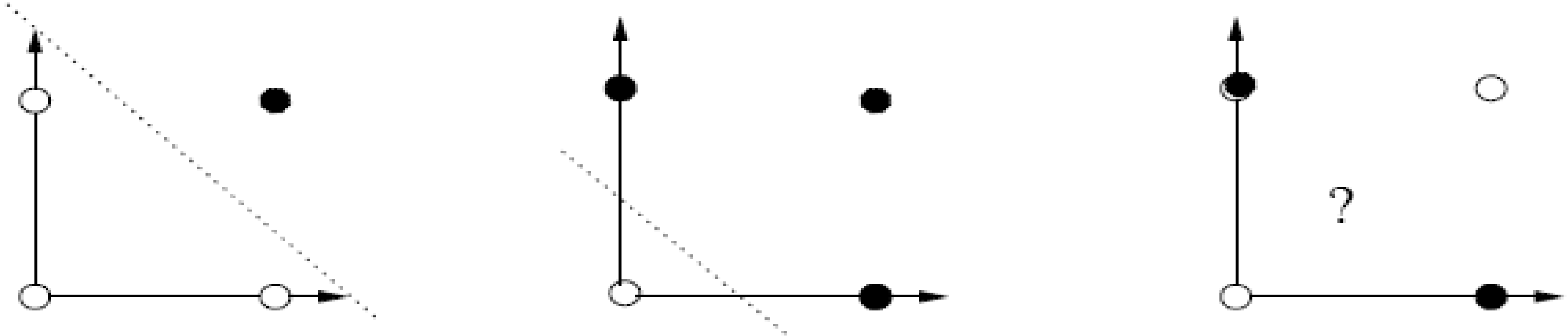
Unidade de entrada especial
Sempre produz 1

A função Threshold
É representada por esse peso



Capacidade de Aprendizado da Rede Perceptron

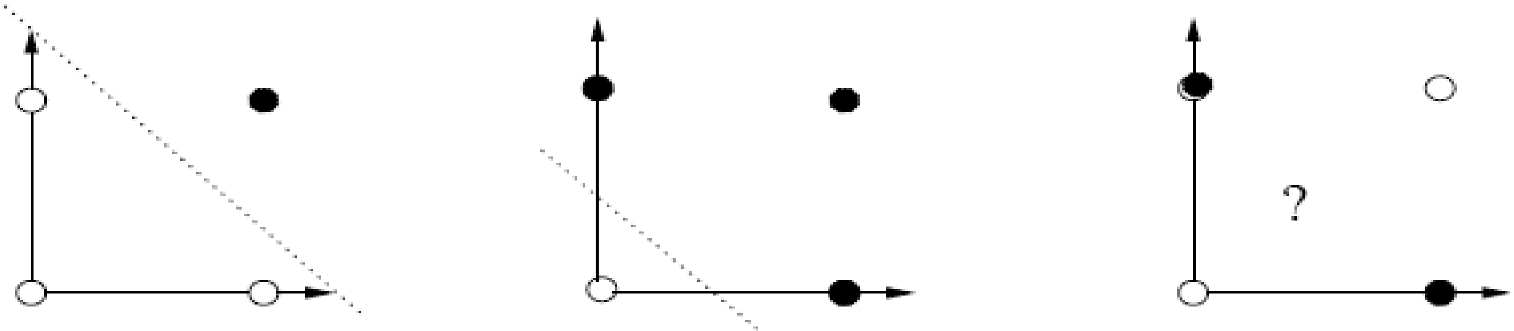
- O que a rede neural Perceptron é capaz de aprender?
  - somente a discriminação de classes que sejam linearmente separáveis.



(a) and (2) or (3) xor

Capacidade de Aprendizado da Rede Perceptron

- Redes Perceptron não conseguem aprender a função XOR.
  - provado em 1969 por Minsky e Papert.
- A função XOR não é linearmente separável.
  - Não é possível traçar uma linha divisória que classifique corretamente todos os pontos.



(a) and (2) or (3) xor

Capacidade de Aprendizado da Rede Perceptron

- Redes Perceptron não conseguem aprender a função XOR.
  - provado em 1969 por Minsky e Papert.
- Decepção na época...
- ... até meados da década de 80

Redes Perceptron Multicamadas

- Redes Perceptron não são capazes de aprender conceitos complexos.
- Porém, os perceptrons formam a base para a construção de um tipo de rede que pode aprender conceitos mais sofisticados.
  - Redes Perceptron Multicamadas (*Multilayer Perceptron* – MLP).
  - Pode-se pensar nesse modelo como sendo uma rede formada por vários neurônios similares ao “tipo perceptron”.

Fim do vídeo 2

Perceptrons

Vídeo 3

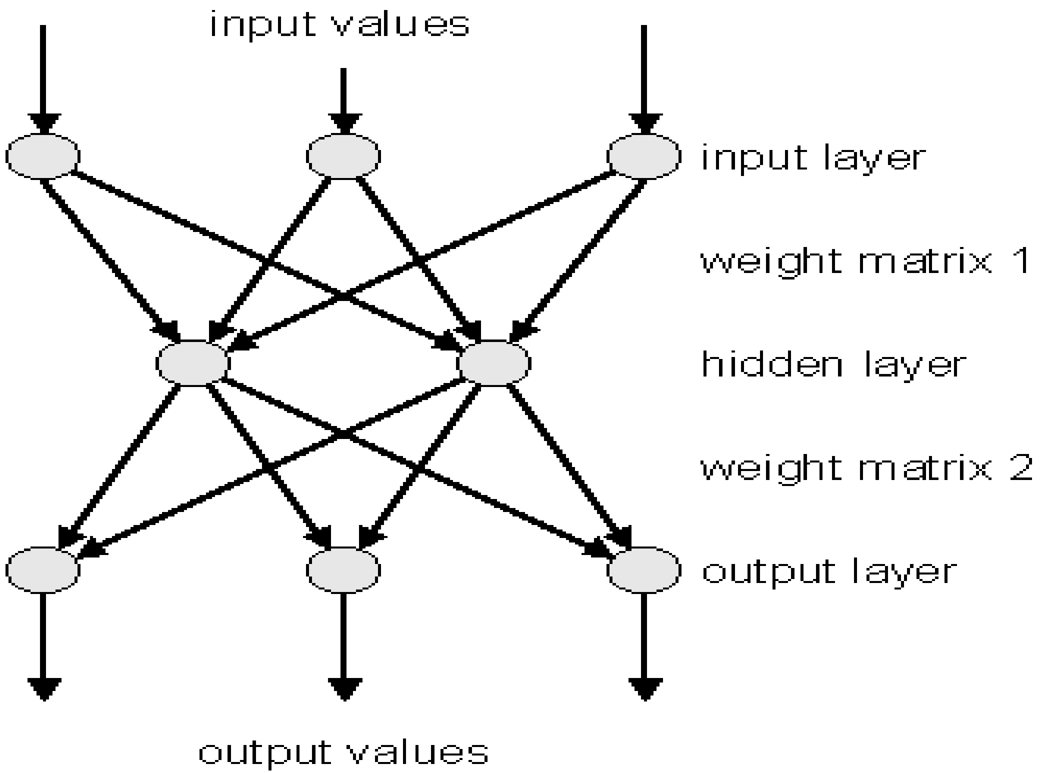
Redes perceptron multicamadas

Redes Perceptron Multicamadas

- Redes Perceptron não são capazes de aprender conceitos complexos.
- Porém, os perceptrons formam a base para a construção de um tipo de rede que pode aprender conceitos mais sofisticados.
  - Redes Perceptron Multicamadas (*Multilayer Perceptron* – MLP).
  - Pode-se pensar nesse modelo como sendo uma rede formada por vários neurônios similares ao “tipo perceptron”.

Redes Perceptron Multicamadas

Na verdade várias camadas ocultas podem existir



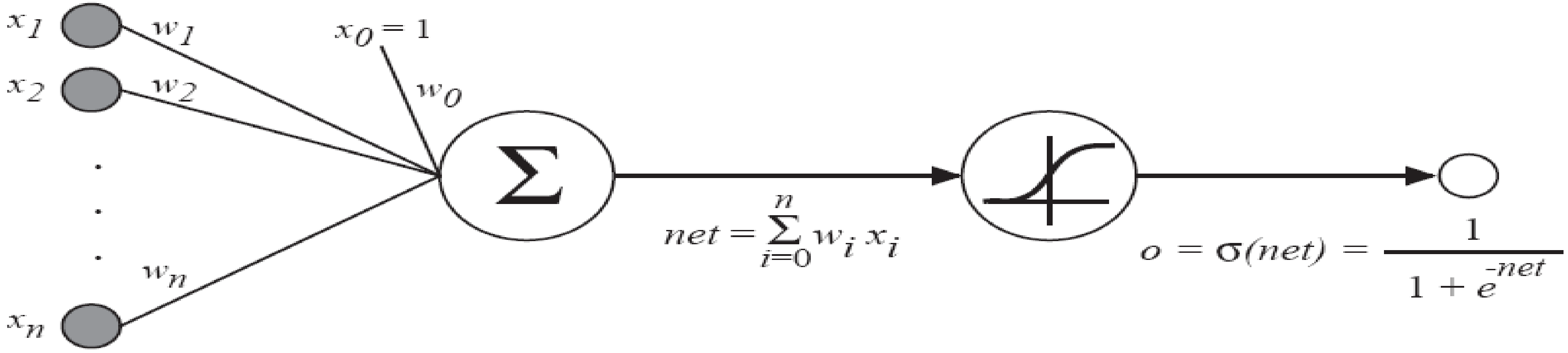
http://fbim.fh-regensburg.de/~saj39122/jfroehl/diplom/e-12-text.html

Redes Perceptron Multicamadas

- Limitações
  - A regra de aprendizado na MLP baseia-se em cálculo diferencial.
  - Funções do tipo degrau não são diferenciáveis.
    - Não são contínuas no valor do threshold.
  - Uma função de ativação alternativa deve ser considerada.
    - Tem que ser diferenciável.

Unidades com função sigmóide

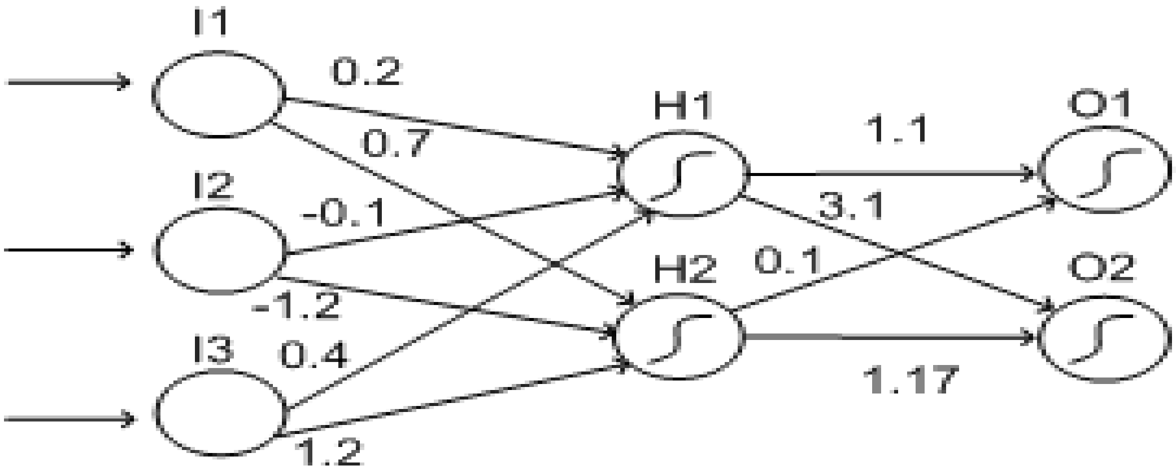
- Unidades com função de ativação sigmóide podem ser usadas em redes MLP.



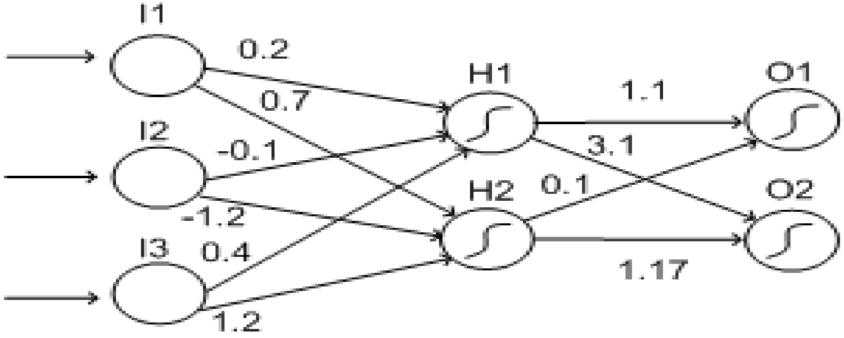
- Função sigmóide: $$\sigma(x) = \frac{1}{1 + e^{-x}}$$
  - Derivada da função sigmóide: $$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

Exemplo de MLP

- Considere a seguinte MLP já treinada e que classifica um exemplo como sendo da classe 1 se O1 > O2 e da classe 2, caso contrário.
- Qual a classe estimada pela rede para o exemplo: [10, 30, 20]?



Exemplo de MLP



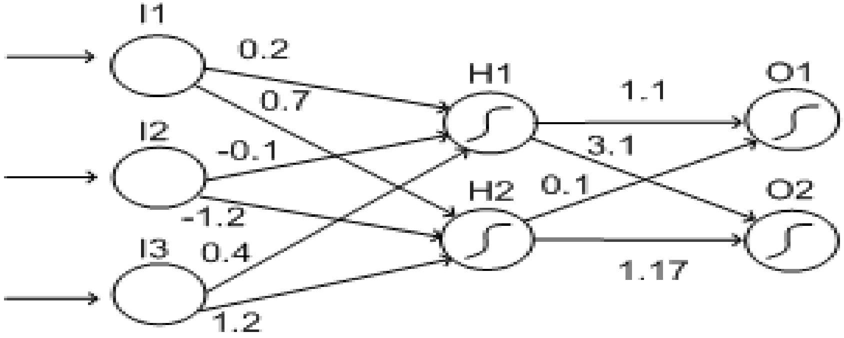
- Primeiro, calcule as somas ponderadas para a camada oculta:

  $S_{H1} = (0.2*10) + (-0.1*30) + (0.4*20) = 2-3+8 = 7$

  $S_{H2} = (0.7*10) + (-1.2*30) + (1.2*20) = 7-6+24= -5$

- A seguir, calcule a saída da camada oculta:
  - Usando: $h = \sigma(S) = 1/(1 + e^{-S})$
    - $h1 = \sigma(S_{H1}) = 1/(1 + e^{-7}) = 1/(1+0.000912) = 0.999$
    - $h2 = \sigma(S_{H2}) = 1/(1 + e^{5}) = 1/(1+148.4) = 0.0067$

Exemplo de MLP



$h1 = 0.999; h2 = 0.0067$

- A seguir, calcule as somas ponderadas para a camada de saída:

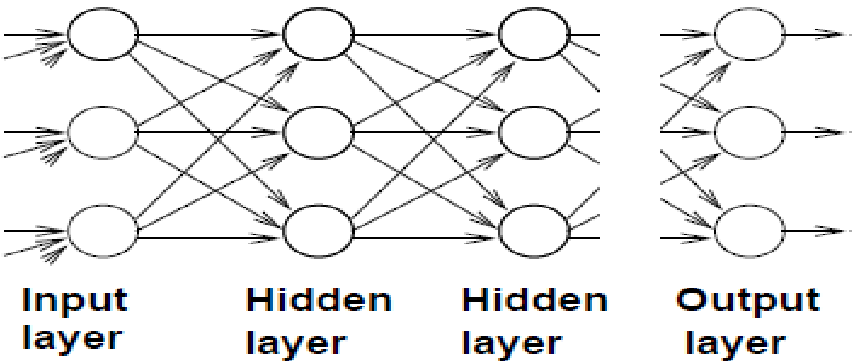
  $S_{O1} = (1.1 * 0.999) + (0.1 * 0.0067) = 1.0996$

  $S_{O2} = (3.1 * 0.999) + (1.17 * 0.0067) = 3.1047$

- Finalmente, calcule a saída da rede:
  - Usando: $\sigma(S) = 1/(1 + e^{-S})$
    - $o1 = \sigma(S_{O1}) = 1/(1 + e^{-1.0996}) = 1/(1+0.333) = 0.750$
    - $o2 = \sigma(S_{O2}) = 1/(1 + e^{-3.1047}) = 1/(1+0.045) = 0.957$
- Como a saída do neurônio O2 > saída do neurônio O1
  - a classe estimada para o exemplo é a classe 2.

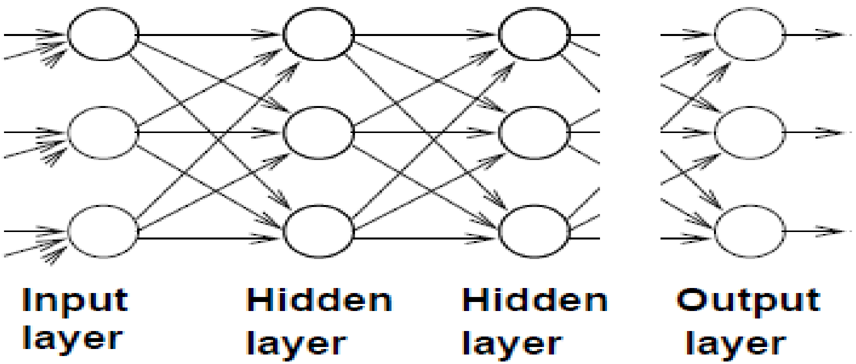
Características da MLP

- Rede Neural do tipo “feedforward”:
  - Alimentação de entradas pela camada mais à esquerda;
  - Propagação dos sinais para frente através da rede;
  - Neurônios entre camadas vizinhas estão completamente conectados.



Características da MLP

- Camada de entrada: exemplos (sinais) de entrada.
- Camada(s) oculta(s): necessária(s) para o aprendizado de funções complexas.
- Camada de saída: saídas da rede.



Aprendizado de redes MLP

Seja S uma amostra de treinamento

1) Gere um conjunto de pesos com valores aleatórios para a rede (por exemplo entre -1 e 1).

2) Enquanto o critério de convergência não for alcançado, faça:

3) Para cada exemplo *e* de S:

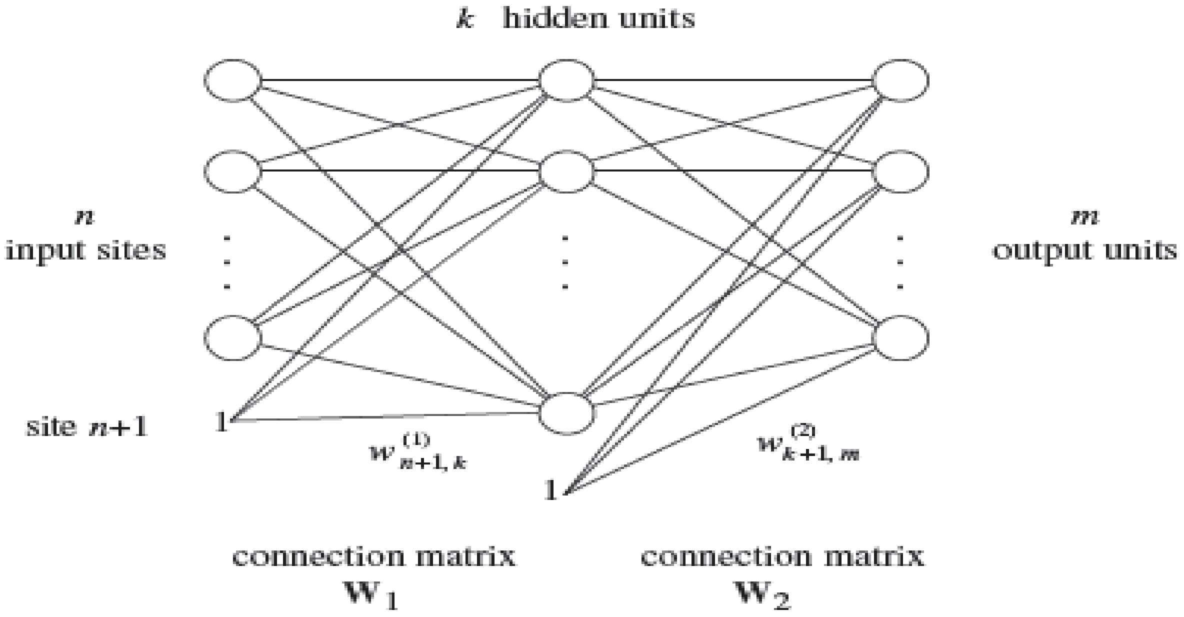
3.1) Apresente o exemplo *e* (vetor de características) para a rede e calcule as suas saídas

3.2) A diferença entre a saída da rede e a saída desejada é considerada como o valor de erro para essa iteração

3.3) Ajuste os pesos da rede

Algoritmo Backpropagation para **ajuste dos pesos**

- Obs: aqui considera-se uma MLP com apenas uma camada oculta.



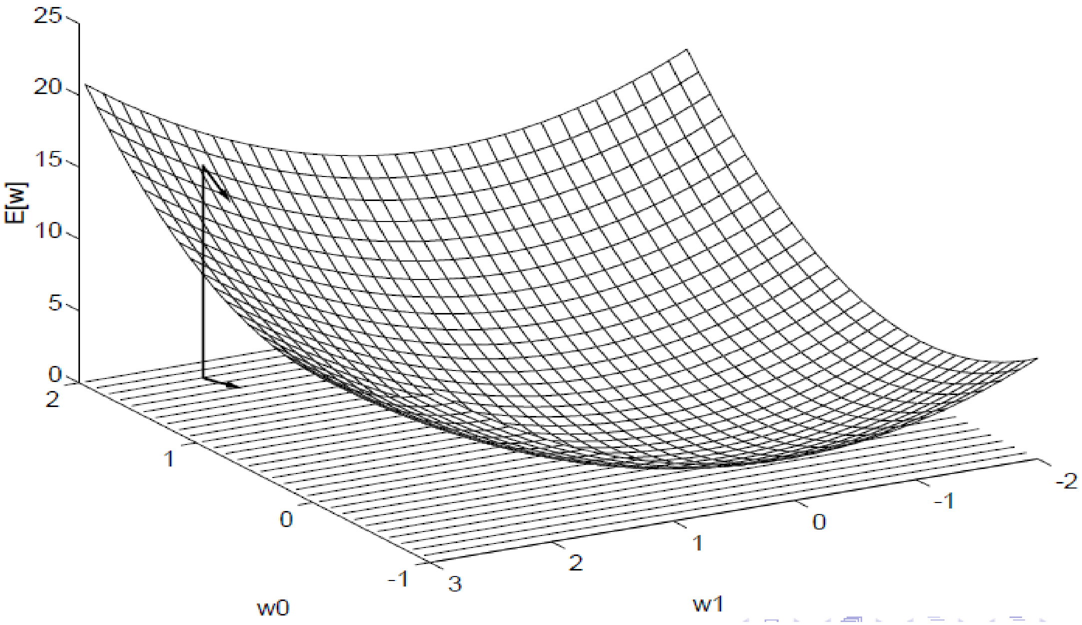
Regra de Aprendizagem para **Perceptrons**

- Quando $t(E)$ for diferente de $o(E)$
  - Adicione Δ_i ao peso w_i
    - Em que $\Delta_i = \eta(t(E) - o(E))x_i$

      ERRO
  - Faça isso para todos os pesos da rede.

Algoritmo Backpropagation – Idéia Geral

- O ajuste de pesos em uma rede MLP se dá por meio da aplicação do algoritmo de aprendizagem Backpropagation.
- Idéia geral: *Erro* = f(**w**)
  - Deseja-se minimizar o valor de Erro.
    - Problema de otimização multidimensional



Ajuste dos pesos via Backpropagation

- A notação w_{ij} é usada para:
  - especificar o peso da conexão entre o neurônio i e o neurônio j.
- Para cada exemplo, calcule o ajuste Δw_{ij} para cada peso w_{ij} da rede.
  - e depois adicione Δw_{ij} à w_{ij}
- Para calcular os ajustes, é necessário calcular os termos de erro δ^k_i para cada i-ésimo neurônio da camada k.
  - Primeiro, calcula-se o termo de erro para as unidades na camada de saída (2);
  - Depois, essas informações são usadas para calcular os termos de erro para as unidades da camada oculta (1).
- Dessa forma, os erros são propagados de volta através da rede.

Regra de Aprendizagem para **Perceptrons**

- Quando $t(E)$ for diferente de $o(E)$
  - Adicione Δ_i ao peso w_i
    - Em que $\Delta_i = \eta(t(E) - o(E))x_i$

      ERRO
  - Faça isso para todos os pesos da rede.

Algoritmo Backpropagation

- Inicialize todos os pesos da rede com pequenos valores aleatórios.
- Enquanto o critério de convergência não for alcançado, faça:
  - Para cada exemplo *e* do conjunto de treinamento, faça:

1) Apresente o exemplo *e* para a rede e calcule as suas saídas

2) Para cada neurônio j, da camada de saída (2) faça:

Calcule $\delta_j^{(2)}$

3) Para cada neurônio j, da camada oculta (1) faça:

Calcule $\delta_j^{(1)}$

4) Ajuste cada peso wij da rede: $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$ sendo

$\Delta w_{ij}^{(2)} = \eta h_i \delta_j^{(2)}$, para $i = 1, \ldots, k+1$; $j = 1, \ldots, m$, h_i = saída do neurônio oculto i

e

$\Delta w_{ij}^{(1)} = \eta x_i \delta_j^{(1)}$, para $i = 1, \ldots, n+1$; $j = 1, \ldots, k$, x_i = entrada i

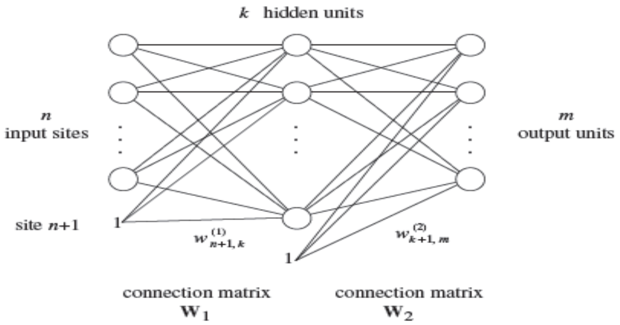
Algoritmo Backpropagation

- Inicialize todos os pesos da rede com pequenos valores aleatórios.
- Enquanto o critério de convergência não for alcançado, faça:
  - Para cada exemplo *e* do conjunto de treinamento, faça:

  1) Apresente o exemplo *e* para a rede e calcule as suas saídas

  2) Para cada neurônio j, da camada de saída (2) faça:

$$\delta_j^{(2)} = (t_j - o_j^{(2)})\, o_j^{(2)}\, (1 - o_j^{(2)})$$



  3) Para cada neurônio j, da camada oculta (1) faça:

$$\delta_j^{(1)} = o_j^{(1)}\, (1 - o_j^{(1)})\, \Sigma_s\ \delta_s^{(2)}\ w_{js}^{(2)}$$

  4) Ajuste cada peso wij da rede: wij ← wij + Δwij sendo

$\Delta w_{ij}^{(2)} = \eta h_i \delta_j^{(2)}$, para i = 1, ..., k+1; j = 1, ..., m, h_i = saída do neurônio oculto i

e

$\Delta w_{ij}^{(1)} = \eta x_i \delta_j^{(1)}$, para i = 1, ..., n+1; j = 1, ..., k, x_i = entrada i

Algoritmo Backpropagation

Como calcular $\delta_j^{(2)}$ e $\delta_j^{(1)}$:

Considerando o método de otimização por mínimos quadrados, deseja-se minimizar:

$E = \text{Erro}(\mathbf{w}) = \Sigma_p \,||\, \mathbf{t}^P - f\,(\mathbf{x}^p; \mathbf{w}) \,||^{\,2}$, onde P é o índice do exemplo,

$$\mathbf{y} = \mathbf{o}^{(2)} = f\,(\mathbf{x}^p; \mathbf{w})$$

Para um dado exemplo:

Erro associado a cada neurônio de saída j : $e_j = t_j - o_j$

Erro da rede: $E = \frac{1}{2}\, \Sigma_j \; e_j^{\,2}$

Queremos que Δw_{ij} seja proporcional a $\partial E / \partial w_{ij}$ (para minimizar o erro via técnica de gradiente descendente)

Vamos chamar de v_j a entrada do neurônio j (soma ponderada das entradas)

Algoritmo Backpropagation

$\partial E / \partial w_{ij} = \partial E / \partial e_j \ \partial e_j / \partial o_j \ \partial o_j / \partial v_j \ \partial v_j / \partial w_{ij}$

$\partial E / \partial e_j = e_j$

$\partial e_j / \partial o_j = -1$

$\partial o_j / \partial v_j = f'_j(v_j)$ (f = função de ativação)

$\partial v_j / \partial w_{ij} = g_i$, na qual g_i é a saída do neurônio i (da camada anterior) que entra no neurônio j ($v_j = \Sigma_s\, g_s * w_{sj}$)

$\partial E / \partial w_{ij} = - e_j\, f'_j(v_j)\, g_i$

$\Delta w_{ij} = - \eta\, \partial E / \partial w_{ij}$ ("-" por ser gradiente DESCENDENTE para reduzir o valor de E)

$\Delta w_{ij} = - \eta\, \partial E / \partial v_j \ \partial v_j / \partial w_{ij} = \ \eta\, \delta_j\, g_i \longrightarrow$

x_i se o neurônio j for da camada oculta

h_i se o neurônio j for da camada de saída

$\delta_j = - \ \partial E / \partial v_j \ = - \partial E / \partial e_j \ \partial e_j / \partial o_j \ \partial o_j / \partial v_j = \ e_j\, f'_j(v_j)$

Algoritmo Backpropagation

$\partial E / \partial w_{ij} = \partial E/\partial e_j \ \partial e_j/\partial o_j \ \partial o_j/\partial v_j \ \partial v_j/\partial w_{ij}$

$\partial E/\partial e_j = e_j$

$\partial e_j/\partial o_j = -1$

$\partial o_j/\partial v_j = f'_j(v_j)$ (f = função de ativação)

$\partial v_j/\partial w_{ij} = g_i$, na qual g_i é a saída do neurônio i (da camada anterior) que entra no neurônio j ($v_j = \Sigma_s \, g_s * w_{sj}$)

$\partial E / \partial w_{ij} = - e_j \, f'_j(v_j) \, g_i$

$\Delta w_{ij} = - \eta \, \partial E/\partial w_{ij}$ ("-" por ser gradiente DESCENDENTE para reduzir o valor de E)

$\Delta w_{ij} = - \eta \, \partial E/\partial v_j \ \partial v_j/\partial w_{ij} = \eta \, \delta_j \, g_i$ $\longrightarrow$?

$\delta_j = - \ \partial E/\partial v_j = - \partial E/\partial e_j \ \partial e_j/\partial o_j \ \partial o_j/\partial v_j = e_j \, f'_j(v_j)$

Determinado se o neurônio j for da camada de saída

Mas se o neurônio j for da camada oculta, como calcular o erro se não há saída esperada?

Algoritmo Backpropagation

Para a camada de saída:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

$\Delta w_{ij} = \eta\, \delta_j^{(2)}\, h_i$

$\delta_j^{(2)} = e_j\, f'_j(v_j)$

Para f = função logística ($\sigma(x)$) :

$\delta_j^{(2)} = e_j\, f'_j(v_j) = e_j\, f(v_j)(1 - f(v_j)) = (t_j - o_j^{(2)})\, o_j^{(2)}\, (1 - o_j^{(2)})$

Algoritmo Backpropagation

O erro de um neurônio da camada oculta terá que ser calculado recursivamente a partir dos erros de todos os neurônios a que ele está diretamente conectado (da camada seguinte)

Algoritmo Backpropagation

Para um neurônio j da camada oculta:

Podemos redefinir δ_j:

$\boldsymbol{\delta_j} = - \partial E/\partial e_j \;\; \partial e_j/\partial o_j \;\; \partial o_j/\partial v_j = - \partial E/\partial o_j \;\; \partial o_j/\partial v_j = \;\; \boldsymbol{- \partial E/\partial o_j \, f'_j(v_j)}$

Lembrando que $E = \frac{1}{2} \Sigma_k \; e_k^2$ para k sendo um neurônio de saída:

$\partial E/\partial o_j = \Sigma_k \; e_k \;\; \partial e_k/\partial o_j \quad = \Sigma_k \; e_k \;\; \partial e_k/\partial v_k \;\; \partial v_k/\partial o_j$ (usando a regra da cadeia)

$e_k = t_k - o_k = t_k - f_k(v_k) \quad => \quad \partial e_k/\partial v_k \; = - f'_k(v_k)$

$v_k = \Sigma_j \; w_{jk} o_j$

$\partial v_k/\partial o_j = w_{jk}$

$\partial E/\partial o_j = - \Sigma_k \; e_k \; f'_k(v_k) \, w_{jk} = - \Sigma_k \; \delta_k \, w_{jk}$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$$

$\boldsymbol{\delta_j} = f'_j(v_j) \, \Sigma_k \; \delta_k \; w_{jk}$

Para f = função logística ($\sigma(x)$) : $\delta_j = f_j(v_j) \, (1-f_j(v_j)) \, \Sigma_k \; \delta_k \; w_{jk}$

$$\delta_j^{(1)} = o_j^{(1)} \, (1-o_j^{(1)}) \, \Sigma_k \; \delta_k^{(2)} \; w_{jk}^{(2)}$$

Algoritmo Backpropagation

- Inicialize todos os pesos da rede com pequenos valores aleatórios.
- Enquanto o critério de convergência não for alcançado, faça:
  - Para todos os exemplos no conjunto de treinamento, faça:

1) Apresente um exemplo para a rede e calcule as suas saídas.

2) Para cada neurônio j, da camada de saída (2) faça:

$$\delta_j^{(2)} = (t_j - o_j^{(2)})\, o_j^{(2)}\, (1 - o_j^{(2)})$$

3) Para cada neurônio j, da camada oculta(1) faça:

$$\delta_j^{(1)} = o_j^{(1)}\, (1-o_j^{(1)})\, \Sigma_s\ \delta_s^{(2)}\ w_{js}^{(2)}$$

4) Ajuste cada peso wij da rede: wij ← wij + Δwij onde

$\Delta w_{ij}^{(2)} = \eta h_i \delta_j^{(2)}$, para i = 1, ..., k+1; j = 1, ..., m; $h_i = o_i^{(1)}$ = saída do neurônio oculto i

e

$\Delta w_{ij}^{(1)} = \eta x_i \delta_j^{(1)}$, para i = 1, ..., n+1; j = 1, ..., k , x_i = entrada i

Algoritmo Backpropagation - Critério de Parada

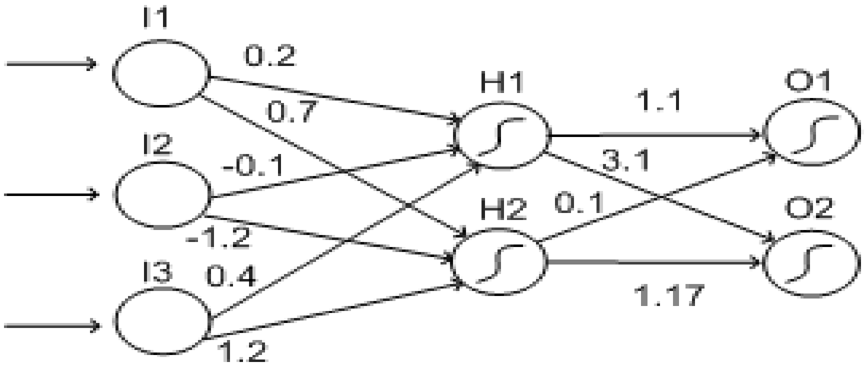
- Várias condições podem ser usadas como critério de parada.
  - Pode-se optar por parar depois de um número fixo de iterações.
    - em que uma iteração é definida pela apresentação completa do conjunto de treinamento (“época”).
  - Ou até que o erro sobre o conjunto de treinamento esteja abaixo de um determinado limiar (“threshold”).
    - Em que o erro é definido como:

$$\frac{1}{2}\sum_{E \in examples}\left(\sum_{k \in outputs}(t_k(E) - o_k(E))^2\right)$$

  - Ou até que o erro sobre um conjunto de validação esteja abaixo de um determinado limiar (“threshold”).

Algoritmo Backpropagation - Exemplo

- Considere uma MLP com a seguinte configuração:



- Atualize os pesos dessa rede, supondo que o exemplo $e = (10,30,20)^T$ seja dado como entrada para o modelo acima.

Considere ainda que:

– *e* deva ser classificado como sendo da classe 1 (ie: $t_1(e) = 1$ e $t_2(e) = 0$)

– a taxa de aprendizagem seja $\eta = 0.1$

Algoritmo Backpropagation - Exemplo

- A propagação do exemplo *e* através da rede é resumida pelos seguintes cálculos (slides 54 a 56):

Input units		Hidden units			Output units		
Unit	Output	Unit	Weighted Sum Input	Output	Unit	Weighted Sum Input	Output
I1	10	H1	7	0.999	O1	1.0996	0.750
I2	30	H2	-5	0.0067	O2	3.1047	0.957
I3	20						

- Ainda:
  - $t_1(e) = 1$ e $t_2(e) = 0$

$o_1(e) = o_1^{(2)}(e) = 0.750$ e $o_2(e) = o_2^{(2)}(e) = 0.957$

Algoritmo Backpropagation - Exemplo

- Dado que:

  $t_1(e) = 1$ e $t_2(e) = 0$

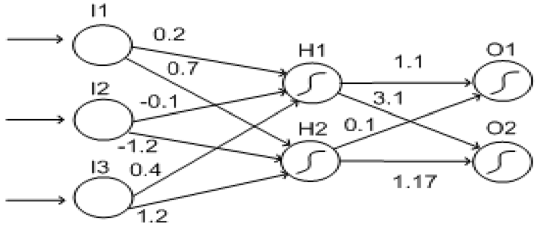
  $o_1(e) = o_1^{(2)}(e) = 0.750$ e $o_2(e) = o_2^{(2)}(e) = 0.957$

- Os termos de erro para os neurônios de saída, são calculados da seguinte forma:

$$\delta_1^{(2)} = (t_1 - o_1^{(2)})\, o_1^{(2)} (1 - o_1^{(2)}) = (1 - 0.750)\, 0.750\, (1 - 0.750) = 0.0469$$

$$\delta_2^{(2)} = (t_2 - o_2^{(2)})\, o_2^{(2)} (1 - o_2^{(2)}) = (0 - 0.957)\, 0.957\, (1 - 0.957) = -0.0394$$

Algoritmo Backpropagation - Exemplo



- Dado que:

$\delta_1^{(2)} = 0.0469$ e $\delta_2^{(2)} = -0.0394$

$h_1 = o_1^{(1)} = 0.999$ e $h_2 = o_2^{(1)} = 0.0067$

Os termos de erro para os neurônios ocultos, são calculados da seguinte forma: ($\delta_j^{(1)} = o_j^{(1)} (1-o_j^{(1)}) \Sigma_s \; w_{js}^{(2)} \delta_s^{(2)}$)

– Para H1, realiza-se a somatória:

$(w_{11} * \delta_1^{(2)}) + (w_{12} * \delta_2^{(2)}) = (1.1*0.0469)+(3.1*-0.0394) = -0.0706$

– E depois multiplica-se o resultado acima por $o_1^{(1)} (1-o_1^{(1)})$:

$-0.0706 * (0.999 * (1-0.999)) = 0.0000705 = \delta_{H1} = \delta_1^{(1)}$

Algoritmo Backpropagation - Exemplo

- Para H2, realiza-se a somatória:

$(w_{21}*\delta_1^{(2)}) + (w_{22}*\delta_2^{(2)}) = (0.1*0.0469)+(1.17*-0.0394) = -0.0414$

E depois multiplica-se o resultado acima por $o_2^{(1)}(e)(1-o_2^{(1)}(e))$:

$-0.0414 * (0.067 * (1-0.067)) = -0.00259 = \delta_{H2} = \delta_2^{(1)}$

Algoritmo Backpropagation - Exemplo

- Os cálculos das mudanças de pesos para as conexões entre a camada de entrada e a camada oculta estão resumidos na tabela:

Input unit	Hidden unit	η	δ_H	x_i	$\Delta = \eta * \delta_H * x_i$	Old weight	New weight
I1	H1	0.1	-0.0000705	10	-0.0000705	0.2	0.1999295
I1	H2	0.1	-0.00259	10	-0.00259	0.7	0.69741
I2	H1	0.1	-0.0000705	30	-0.0002115	-0.1	-0.1002115
I2	H2	0.1	-0.00259	30	-0.00777	-1.2	-1.20777
I3	H1	0.1	-0.0000705	20	-0.000141	0.4	0.39999
I3	H2	0.1	-0.00259	20	-0.00518	1.2	1.1948

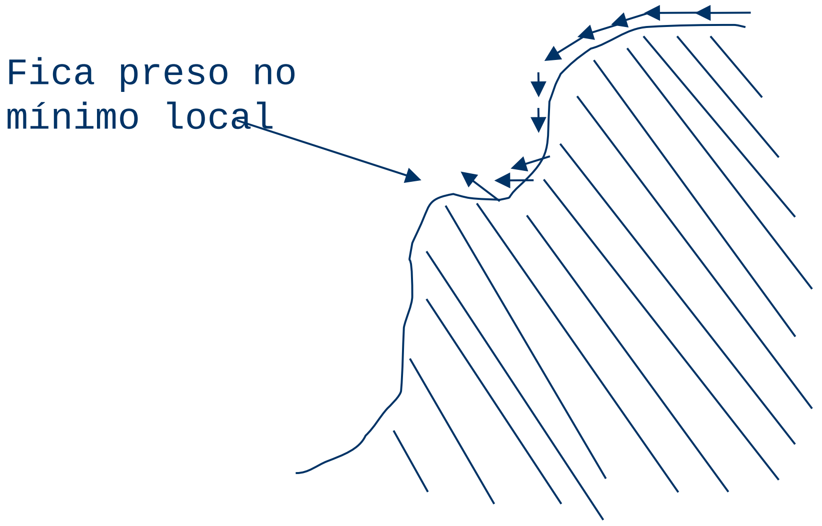
Algoritmo Backpropagation - Exemplo

- Os cálculos das mudanças de pesos para as conexões entre a camada oculta e a camada de saída estão resumidos na tabela:

Hidden unit	Output unit	η	δ_O	$h_i(E)$	$\Delta = \eta * \delta_O * h_i(E)$	Old weight	New weight
H1	O1	0.1	0.0469	0.999	0.000469	1.1	1.100469
H1	O2	0.1	-0.0394	0.999	-0.00394	3.1	3.0961
H2	O1	0.1	0.0469	0.0067	0.00314	0.1	0.10314
H2	O2	0.1	-0.0394	0.0067	-0.0000264	1.17	1.16998

Algoritmo Backpropagation – Convergência e Mínimos Locais

- O backpropagation implementa uma busca por descida do gradiente (*gradient descent*) através do espaço dos possíveis pesos da rede.
  - iterativamente reduz o erro entre os valores esperados e os obtidos pela rede.
- Uma vez que a superfície de erro pode conter vários mínimos locais diferentes, o método pode ficar "preso" em um desses pontos.
  - Não há garantias de que o mínimo global será alcançado.



Algoritmo Backpropagation – Termo Momentum

- Uma das possíveis soluções utilizadas para tentar evitar o problema dos mínimos locais é adicionar um <u>termo momentum</u> à regra de aprendizado dos pesos da rede.

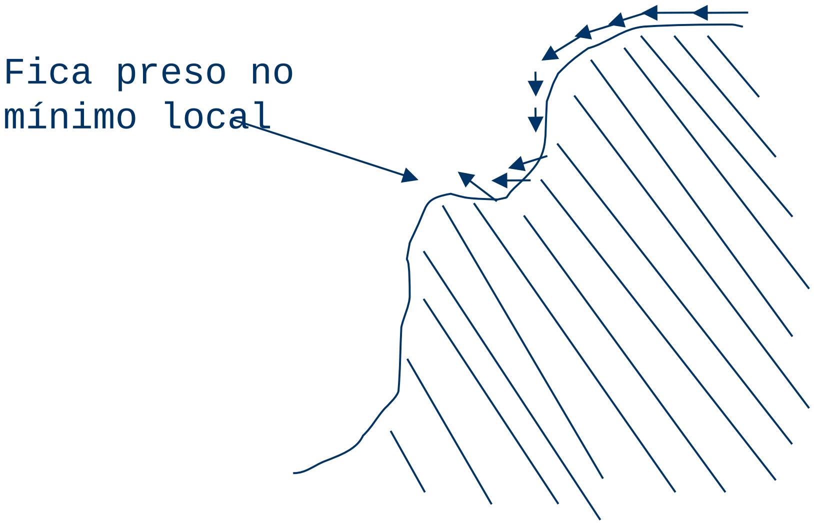
$$\Delta w_{ij}(n) = \eta \delta_j x_{ij} + \alpha \Delta w_{ij}(n-1)$$

- Esse termo faz com que a atualização dos pesos na n-ésima iteração dependa parcialmente da atualização ocorrida na iteração anterior (n-1).
  - α é uma constante no intervalo (0, 1).

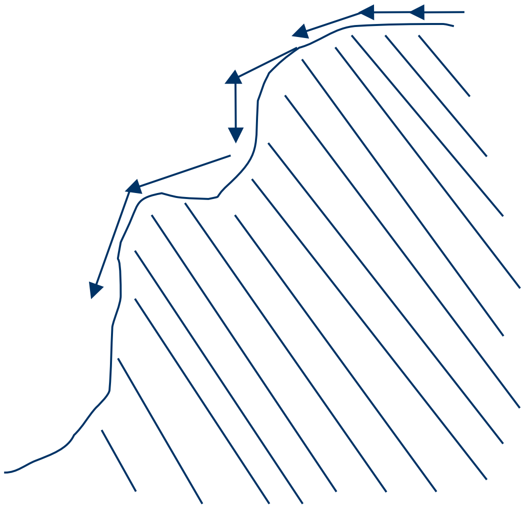
Algoritmo Backpropagation – Termo Momentum

- A trajetória de busca do *gradient descent* é análoga a de uma bola rolando para baixo na superfície de erro.
- O efeito do momentum é tentar conservar essa “bola” rolando na mesma direção entre uma iteração e a subsequente a esta.

Sem Momentum



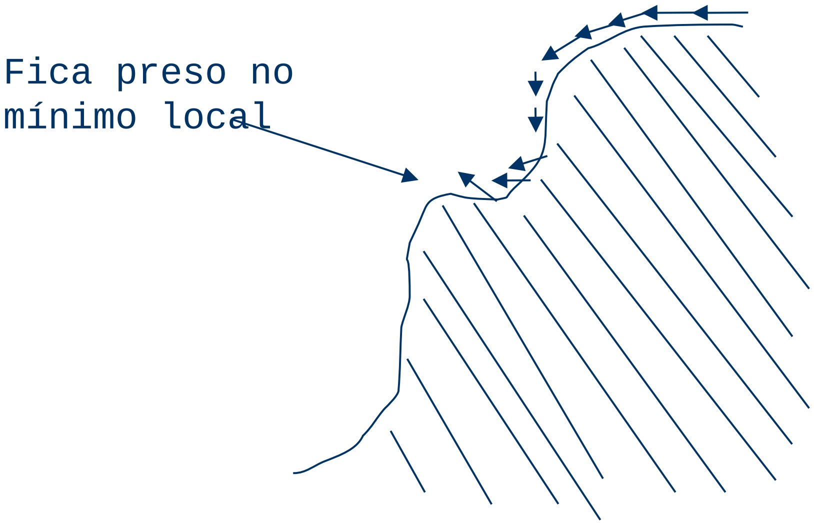
Com Momentum



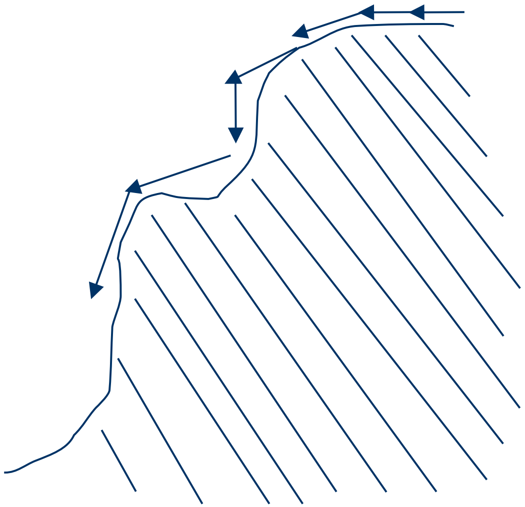
Algoritmo Backpropagation – Termo Momentum

- O termo momentum também possui o efeito de aumentar gradualmente o tamanho do passo da busca
  - Acelera a convergência do algoritmo.

Sem Momentum



Com Momentum



Observações

- Erro quadrático (função custo): $\frac{1}{2}\sum_{E \in examples}\left(\sum_{k \in outputs}(t_k(E) - o_k(E))^2\right)$

Funções custo alternativas são exploradas para melhorar as características de convergência (entropia cruzada)

- Regularização - para evitar *overfitting*:
  - L2: adicionar ao erro (função custo) um λ Σ(wijk)2
  - Ex: Em Python, λ (chamado alpha) te, valor default de 0,0001
- Função custo entropia cruzada + regularização L2:

$$Loss(\hat{y}, y, W) = -y \ln \hat{y} - (1-y)\ln(1-\hat{y}) + \alpha||W||_2^2$$

- Dropout: durante o treinamento zerar (0.2 a 0.5) dos neurônios (também para evitar *overfitting*)

Poder de MLPs

Teoricamente, redes neurais do tipo MLPs podem aproximar qualquer função computável, desde que disponível um número suficientemente grande de unidades ocultas.

[HORNIK, 1991]

Possíveis problemas de MLPs

Treinamento lento

Quanto maior a rede, maior o número de parâmetros para serem estimados (maior erro de estimação, perigo de *overfitting*)

Sensível à variação dos valores de inicialização

Não interpretável

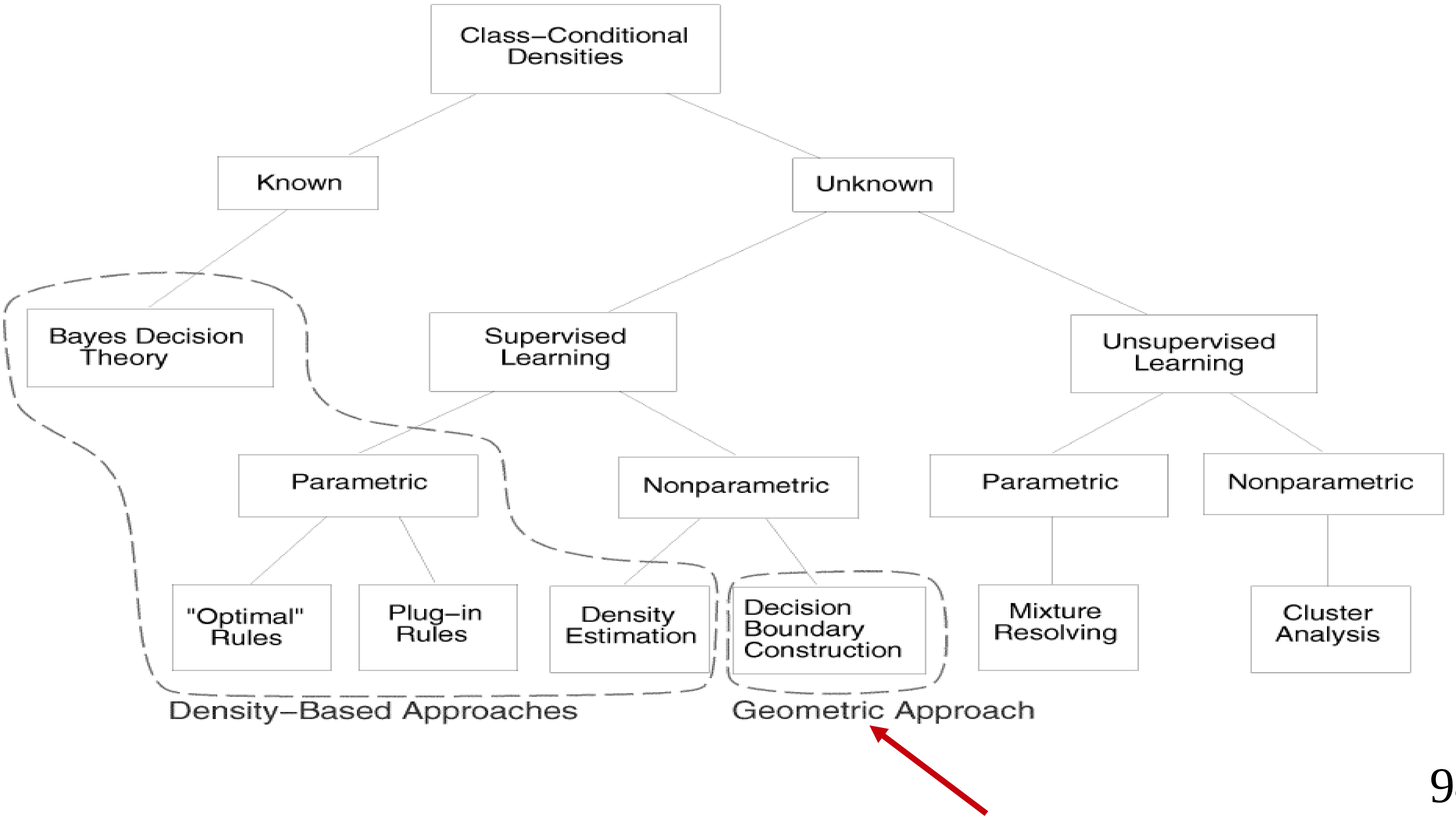
Número e magnitude de outliers

Referências

- MITCHELL, T. **Machine Learning**. McGraw Hill, 1997, cap 4
- BISHOP, C. M. **Neural Networks for Pattern Regnition**. Oxford University Press, 2005, cap 3 e 4
- HORNIK, K. Approximation Capabilities of Multilayer Feedforward Networks. **Neural Networks** v.4, p. 251-257, 1991

Fim do vídeo 3

Redes perceptron multicamadas


Class–Conditional Densities
Known
Unknown
Bayes Decision Theory
Supervised Learning
Unsupervised Learning
Parametric
Nonparametric
Parametric
Nonparametric
"Optimal" Rules
Plug–in Rules
Density Estimation
Decision Boundary Construction
Mixture Resolving
Cluster Analysis
Density–Based Approaches
Geometric Approach

Vídeo 4

Atividade 8

Atividade 8 (para 19/06)

- Seguir o mesmo padrão das atividades 6 e 7, mas agora para redes MLP
- Realizar validação cruzada para testar redes neurais (mesmos subconjuntos utilizados em SVMs) utilizando:

- todas a características
- apenas com os componentes principais
- apenas com as características selecionadas pelo selecionador 1
- apenas com as características selecionadas pelo selecionador 2 (opcional)

Em cada um, calibrar os parâmetros (número de neurônios da camada oculta (ex: 10, raiz quadrada do nr de características, um nr maior), taxa de aprendizado (ex: 0,1; 0,01), função de ativação (sigmoide na camada de saída, mas na camada oculta testar tangente hiperbólica e leaky relu (ou relu)) e reportar os valores médios de precisão, revocação e acurácia para o que apresentou melhor acurácia (com os respectivos intervalos de confiança)

vídeo 5: outras funções de ativação

Fim do vídeo 4

Atividade 8