

Universidade de Fortaleza - UNIFOR

Inteligência Artificial Computacional

T296

Msc. Prof. Paulo Cirillo Souza Barbosa

Centro de Ciências Tecnológicas - CCT

Fortaleza, Ceará, Brasil





1 Redes Neurais Artificiais (RNA).

- 1.1 Introdução.
- 1.2 Neurônio Biológico.
- 1.3 Neurônio Artificial

2 Redes RBF

- 2.1 Projeto da camada oculta

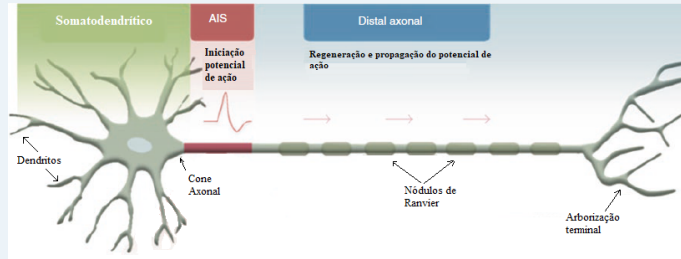


Introdução.

- São modelos computacionais inspirados no sistema nervoso de seres vivos.
- São definidas por um conjunto de unidades de processamento, caracterizadas por **neurônios artificiais** que são interconectados através de uma matriz de pesos (sinapses artificiais).
- Características principais:
 - ① Adaptação por experiência.
 - ② Capacidade de aprendizado.
 - ③ Habilidade de generalização.
 - ④ Organização de dados.
 - ⑤ Tolerância a falhas.
 - ⑥ Armazenamento distribuído.
 - ⑦ Facilidade de prototipagem.
- Aplicações:
 - ① Aproximador universal de funções.
 - ② Controle de processos.
 - ③ Reconhecimento/classificação de padrões.
 - ④ Agrupamento de dados.
 - ⑤ Sistemas de previsão.
 - ⑥ Otimização de sistemas.
 - ⑦ Memórias Associativas.



Neurônio Biológico.



- O neurônio nada mais é do que uma célula que consegue conduzir estímulos elétricos advindos de reações físico-químicas.
 - 1 Dendritos.
 - 2 Soma ou Corpo Celular.
 - 3 Axônio.
 - 4 Sinapses.



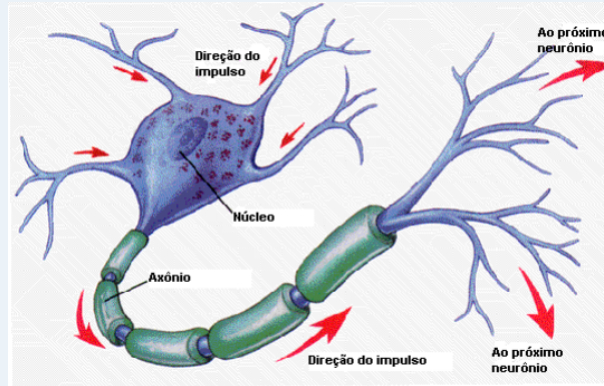
Neurônio Biológico.

- **Dendritos** – Ramificações correspondentes aos canais de entrada de informação (sinais elétricos, escala mV).
- **Corpo Celular** – Local onde é feito o balanço energético da célula nervosa (soma das contribuições de energia).
- **Axônios** – Canal de saída do neurônio, ou seja, caminho de propagação dos impulsos nervosos em direção a outros neurônios ou músculos.
- **Sinapses** – Pontos de contato entre neurônios onde há passagem de neurotransmissores do axônio de um neurônio para os dendritos de outro neurônio.



Neurônio Biológico.

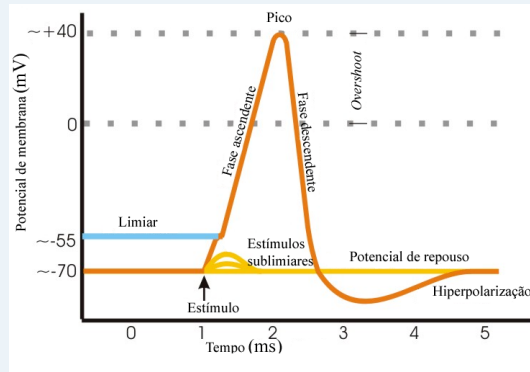
- O Fluxo da informação ocorre sempre no sentido: **Dendritos** \Rightarrow **Corpo Celular** \Rightarrow **Axônio**.





Neurônio Biológico.

- O axônio emite um impulso elétrico (potencial de ação) apenas se o balanço energético realizado no corpo celular for maior que um certo limiar. Neste caso, diz-se que o neurônio disparou ou está ativado.





Neurônio Biológico.

- A chegada deste sinal de disparo no terminal do axônio, faz com que neurotransmissores sejam liberados na fenda sináptica.
- Sinapses podem ser excitatórias (facilitam a passagem do potencial de ação) ou inibitórias (inibem a passagem do potencial de ação).
- Neurônios podem fazer conexões:
 - 1 com outros neurônios.
 - 2 com os músculos diretamente.
 - 3 com os órgãos sensoriais.



Curiosidades!

- Um comparativo pode ser feito com relação às portas lógicas, que operam na ordem dos nanossegundos.
- Um neurônio biológico opera na ordem dos milissegundos.
- Como computadores tem características "inferiores" ao cérebro humano?



Curiosidades!

- Um comparativo pode ser feito com relação às portas lógicas, que operam na ordem dos nanossegundos.
- Um neurônio biológico opera na ordem dos milissegundos.
- Como computadores tem características "inferiores" ao cérebro humano?
- Há cerca de 10 Bilhões de neurônios no cortex cerebral (massa cinzenta).
- O córtex é a estrutura responsável pelas habilidades cognitivas superiores, tais como memória, raciocínio lógico, linguagem, consciência, dentre outras.



Neurônio de McCulloch - Pitts.

- Modelo proposto em: MCCULLOCH, Warren S.; PITTS, Walter. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115-133, 1943.



Neurônio de McCulloch - Pitts.

- Modelo proposto em: MCCULLOCH, Warren S.; PITTS, Walter. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115-133, 1943.
- Faz-se necessário **destacar** que se trata de um modelo, ou seja, é uma aproximação do neurônio natural.



Neurônio de McCulloch - Pitts.

- Modelo proposto em: MCCULLOCH, Warren S.; PITTS, Walter. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115-133, 1943.
- Faz-se necessário **destacar** que se trata de um modelo, ou seja, é uma aproximação do neurônio natural.
- Portanto, o neurônio M-P é uma aproximação útil do neurônio real, pois, serve até hoje como bloco construtivo básico de algoritmos de RNA.



Neurônio de McCulloch - Pitts.

- Modelo proposto em: MCCULLOCH, Warren S.; PITTS, Walter. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115-133, 1943.
- Faz-se necessário **destacar** que se trata de um modelo, ou seja, é uma aproximação do neurônio natural.
- Portanto, o neurônio M-P é uma aproximação útil do neurônio real, pois, serve até hoje como bloco construtivo básico de algoritmos de RNA.
- A modelagem realizada está ligada aos aspectos do **processamento da informação** em um neurônio biológico, ou seja, os caminhos e etapas pelas quais passam os potenciais de ação que trafegam:



Neurônio de McCulloch - Pitts.

- Modelo proposto em: MCCULLOCH, Warren S.; PITTS, Walter. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115-133, 1943.
- Faz-se necessário **destacar** que se trata de um modelo, ou seja, é uma aproximação do neurônio natural.
- Portanto, o neurônio M-P é uma aproximação útil do neurônio real, pois, serve até hoje como bloco construtivo básico de algoritmos de RNA.
- A modelagem realizada está ligada aos aspectos do **processamento da informação** em um neurônio biológico, ou seja, os caminhos e etapas pelas quais passam os potenciais de ação que trafegam:
 - ① de um neurônio a outro neurônio,



Neurônio de McCulloch - Pitts.

- Modelo proposto em: MCCULLOCH, Warren S.; PITTS, Walter. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115-133, 1943.
- Faz-se necessário **destacar** que se trata de um modelo, ou seja, é uma aproximação do neurônio natural.
- Portanto, o neurônio M-P é uma aproximação útil do neurônio real, pois, serve até hoje como bloco construtivo básico de algoritmos de RNA.
- A modelagem realizada está ligada aos aspectos do **processamento da informação** em um neurônio biológico, ou seja, os caminhos e etapas pelas quais passam os potenciais de ação que trafegam:
 - 1 de um neurônio a outro neurônio,
 - 2 receptores sensoriais a um neurônio, ou



Neurônio de McCulloch - Pitts.

- Modelo proposto em: MCCULLOCH, Warren S.; PITTS, Walter. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115-133, 1943.
- Faz-se necessário **destacar** que se trata de um modelo, ou seja, é uma aproximação do neurônio natural.
- Portanto, o neurônio M-P é uma aproximação útil do neurônio real, pois, serve até hoje como bloco construtivo básico de algoritmos de RNA.
- A modelagem realizada está ligada aos aspectos do **processamento da informação** em um neurônio biológico, ou seja, os caminhos e etapas pelas quais passam os potenciais de ação que trafegam:
 - 1 de um neurônio a outro neurônio,
 - 2 receptores sensoriais a um neurônio, ou
 - 3 de um neurônio a um atuador (e.g. músculo).

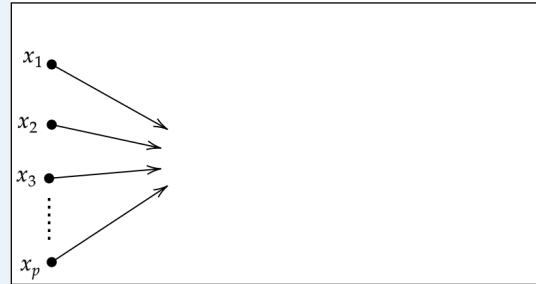
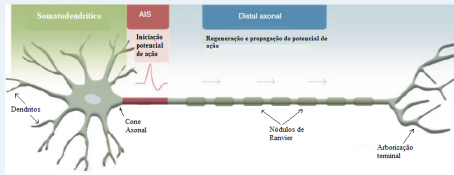


Neurônio de McCulloch - Pitts.

- Modelo proposto em: MCCULLOCH, Warren S.; PITTS, Walter. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115-133, 1943.
- Faz-se necessário **destacar** que se trata de um modelo, ou seja, é uma aproximação do neurônio natural.
- Portanto, o neurônio M-P é uma aproximação útil do neurônio real, pois, serve até hoje como bloco construtivo básico de algoritmos de RNA.
- A modelagem realizada está ligada aos aspectos do **processamento da informação** em um neurônio biológico, ou seja, os caminhos e etapas pelas quais passam os potenciais de ação que trafegam:
 - 1 de um neurônio a outro neurônio,
 - 2 receptores sensoriais a um neurônio, ou
 - 3 de um neurônio a um atuador (e.g. músculo).
- Deseja-se portanto, desenvolver modelos matemáticos que representem os **dendritos**, as **sinapses**, o **corpo celular** e o **axônio**.



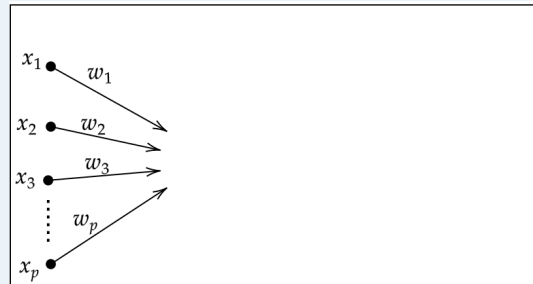
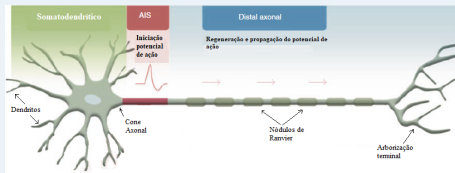
Neurônio de Mculloch - Pitts.



- Cada ramo dendrítico é modelado como um canal, pelo qual flui a informação de entrada ($x_j, j = 1, \dots, p$).



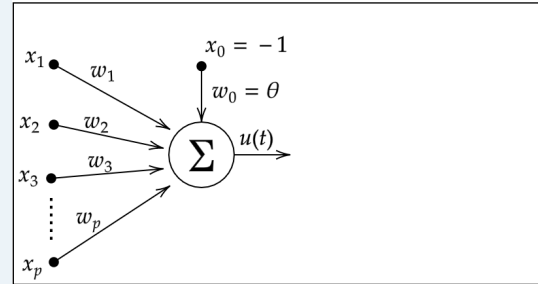
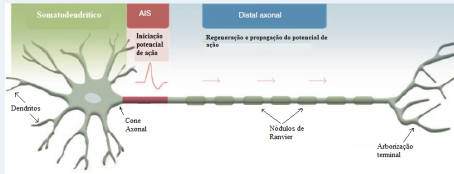
Neurônio de Mculloch - Pitts.



- A força (ou eficiência) das conexões sinápticas de uma certa árvore dendrítica é modelada como um fator (peso sináptico), cujo papel é modular o fluxo de sinais passando por uma certa árvore dendrítica.



Neurônio de Mculloch - Pitts.

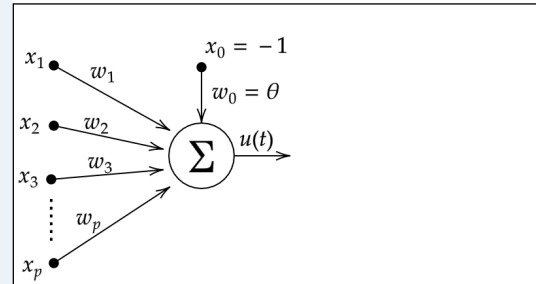
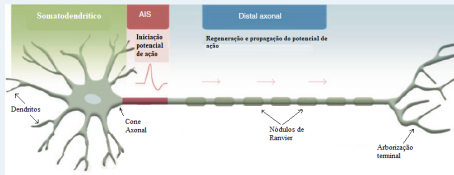


- A função do corpo celular de realizar o balanço ou acúmulo energético é modelada por uma operação de somatório sobre as entradas moduladas pelos pesos sinápticos.

$$u =$$



Neurônio de Mculloch - Pitts.

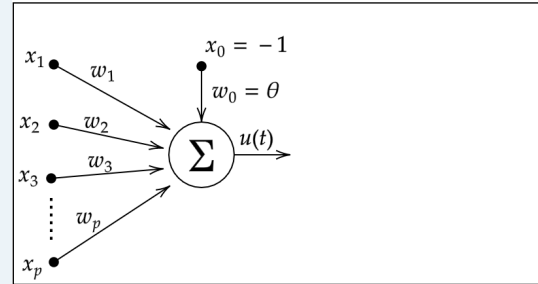
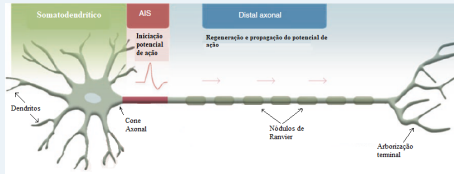


- A função do corpo celular de realizar o balanço ou acúmulo energético é modelada por uma operação de somatório sobre as entradas moduladas pelos pesos sinápticos.

$$u = w_1x_1 + w_2x_2 + \cdots w_px_p - \theta$$



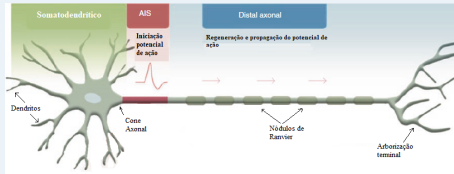
Neurônio de Mculloch - Pitts.



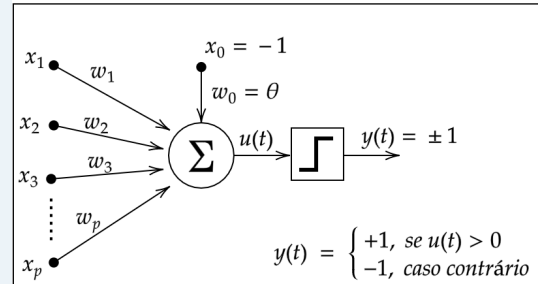
- x_1, x_2 são as entradas.
- w_1, w_2 os pesos sinápticos.
- θ representa o limiar (*threshold*)
- -1 é o viés *bias*
- u ativação.



Neurônio de Mculloch - Pitts.



- x_1, x_2 são as entradas.
- w_1, w_2 os pesos sinápticos.
- θ o limiar (bias, viés, *threshold*)
- u ativação.

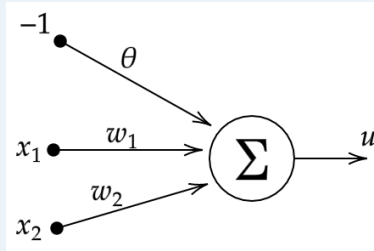


$$u = w_1 x_1 + w_2 x_2 + \dots + w_p x_p - \theta$$



Neurônio de Mculloch - Pitts.

- Dado o seguinte neurônio, com duas entradas x_1 e x_2 , o seu modelo pode ser escrito como:

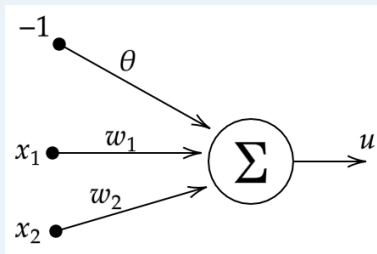


- A combinação linear u é dada por:



Neurônio de Mculloch - Pitts.

- Dado o seguinte neurônio, com duas entradas x_1 e x_2 , o seu modelo pode ser escrito como:



- A combinação linear u é dada por:

$$u = w_1x_1 + w_2x_2 - \theta$$

- Para fins de classificação, pode-se trabalhar no plano (x_1, x_2) , ou seja, $u = 0$.



Neurônio de Mculloch - Pitts.

$$u = w_1x_1 + w_2x_2 - \theta = 0$$

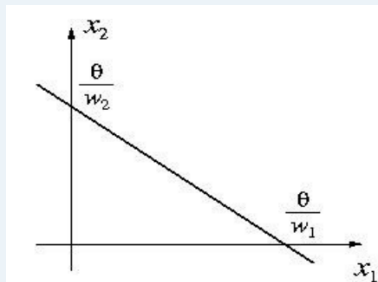


Neurônio de Mculloch - Pitts.

$$u = w_1x_1 + w_2x_2 - \theta = 0$$

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}$$

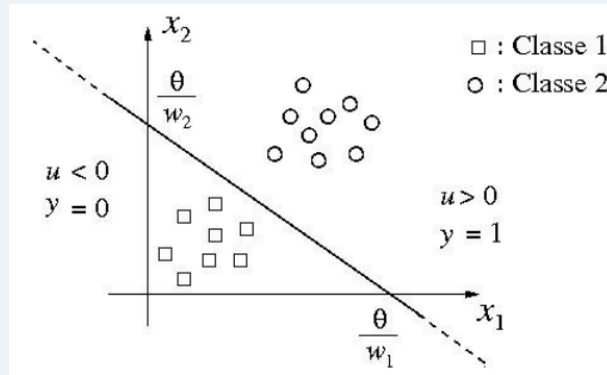
- Esta equação define a seguinte reta em (x_1, x_2) .





Neurônio de Mculloch - Pitts.

- Assim, um neurônio M-P pode ser usado para separar com eficiência, duas classes que estejam bem isoladas uma da outra.

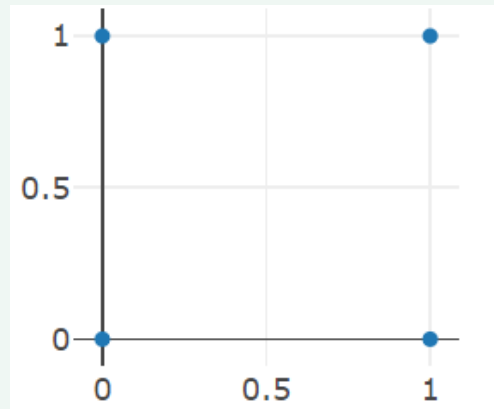




Exemplo 1: Implementação das portas **OR**, **AND** e **NOT**.

- **OR**: É possível encontrar uma reta que separe os pontos da Classe **UM** ($y=1$) dos da Classe **DOIS** ($y=0$)?

x_1	x_1	y
0	0	
0	1	
1	0	
1	1	

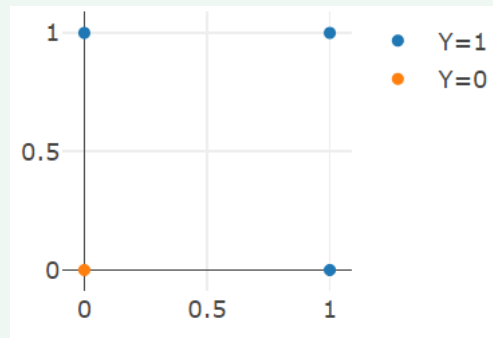




Exemplo 1: Implementação das portas **OR**, **AND** e **NOT**.

- **OR**: É possível encontrar uma reta que separe os pontos da Classe **UM** ($y=1$) dos da Classe **DOIS** ($y=0$)?
- É possível encontrar mais de uma reta?

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1





Exemplo 1: Implementação das portas **OR**, **AND** e **NOT**.

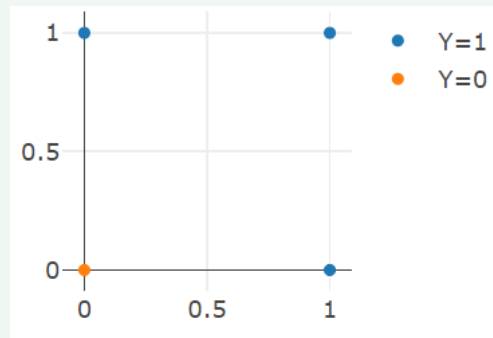
- **OR**: É possível encontrar uma reta que separe os pontos da Classe **UM** ($y=1$) dos da Classe **DOIS** ($y=0$)? SIM
- É possível encontrar mais de uma reta? Quantas?



Exemplo 1: Implementação das portas **OR**, **AND** e **NOT**.

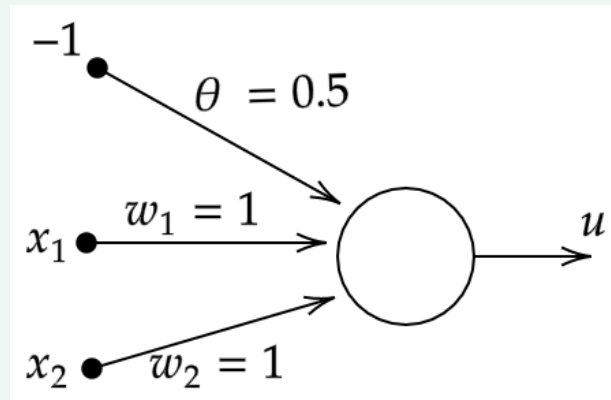
- **OR**: É possível encontrar uma reta que separe os pontos da Classe **UM** ($y=1$) dos da Classe **DOIS** ($y=0$)? **SIM**
- É possível encontrar mais de uma reta? Quantas? Infinitas.

x_1	x_1	y
0	0	0
0	1	1
1	0	1
1	1	1





Exemplo 1: neurônio MP das portas OR.



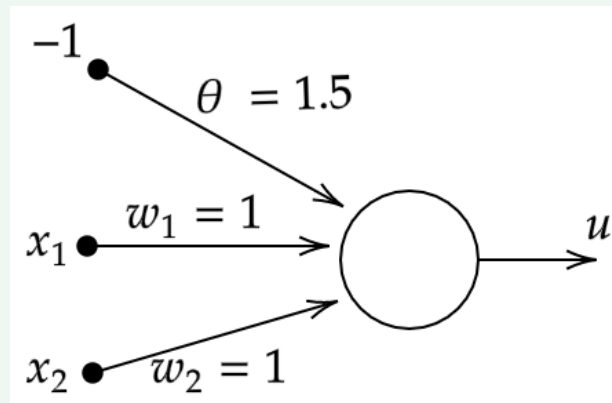
$$w_1 = w_2 = 1 \text{ e } \theta = 0.5$$

$$y = 1, \text{ se } u \geq 0$$

$$y = 0, \text{ se } u < 0$$



Exemplo 1: neurônio MP das portas AND.



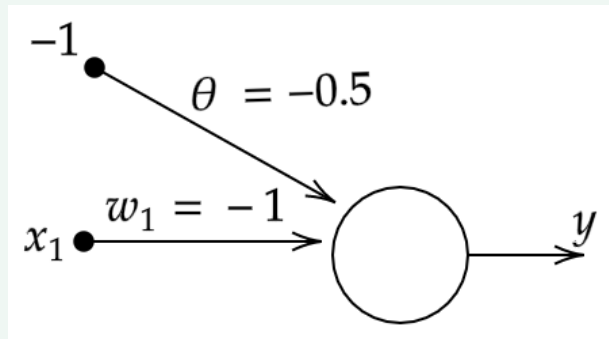
$$w_1 = w_2 = 1 \text{ e } \theta = 1.5$$

$$y = 1, \text{ se } u \geq 0$$

$$y = 0, \text{ se } u < 0$$



Exemplo 1: neurônio MP das portas NOT.



$$w_1 = -1 \text{ e } \theta = -0.5$$

$$y = 1, \text{ se } u \geq 0$$

$$y = 0, \text{ se } u < 0$$



Notas importantes.

- O neurônio MP pode ser usado para implementar as portas lógicas AND, OR e NOT porque estas, do ponto de vista geométrico, podem ser interpretadas como um problema de classificação binária (duas categorias).



Notas importantes.

- O neurônio MP pode ser usado para implementar as portas lógicas AND, OR e NOT porque estas, do ponto de vista geométrico, podem ser interpretadas como um problema de classificação binária (duas categorias).
- O neurônio MP, do ponto de vista geométrico, pode ser interpretado como uma reta (2D), ou um plano (3D) ou ainda um hiperplano ($> 3D$), que é usado para separar duas categorias de dados distintas.



Notas importantes.

- O neurônio MP pode ser usado para implementar as portas lógicas AND, OR e NOT porque estas, do ponto de vista geométrico, podem ser interpretadas como um problema de classificação binária (duas categorias).
- O neurônio MP, do ponto de vista geométrico, pode ser interpretado como uma reta (2D), ou um plano (3D) ou ainda um hiperplano ($> 3D$), que é usado para separar duas categorias de dados distintas.
- Na implementação das portas lógicas AND, OR e NOT, os valores dos pesos e do limiar foram determinados pelo projetista com base na análise geométrica do problema.



Notas importantes.

- O neurônio MP pode ser usado para implementar as portas lógicas AND, OR e NOT porque estas, do ponto de vista geométrico, podem ser interpretadas como um problema de classificação binária (duas categorias).
- O neurônio MP, do ponto de vista geométrico, pode ser interpretado como uma reta (2D), ou um plano (3D) ou ainda um hiperplano ($> 3D$), que é usado para separar duas categorias de dados distintas.
- Na implementação das portas lógicas AND, OR e NOT, os valores dos pesos e do limiar foram determinados pelo projetista com base na análise geométrica do problema.
- Como fazer com que o neurônio M-P determine de forma automática os valores dos pesos e do limiar para um problema específico?



Notas importantes.

- O neurônio MP pode ser usado para implementar as portas lógicas AND, OR e NOT porque estas, do ponto de vista geométrico, podem ser interpretadas como um problema de classificação binária (duas categorias).
- O neurônio MP, do ponto de vista geométrico, pode ser interpretado como uma reta (2D), ou um plano (3D) ou ainda um hiperplano ($> 3D$), que é usado para separar duas categorias de dados distintas.
- Na implementação das portas lógicas AND, OR e NOT, os valores dos pesos e do limiar foram determinados pelo projetista com base na análise geométrica do problema.
- Como fazer com que o neurônio M-P determine de forma automática os valores dos pesos e do limiar para um problema específico?
- Para que o neurônio M-P seja capaz de aprender sozinho a resolver um problema de classificação é necessário dotá-lo de uma **regra de aprendizagem**.



Notas importantes.

- O neurônio MP pode ser usado para implementar as portas lógicas AND, OR e NOT porque estas, do ponto de vista geométrico, podem ser interpretadas como um problema de classificação binária (duas categorias).
- O neurônio MP, do ponto de vista geométrico, pode ser interpretado como uma reta (2D), ou um plano (3D) ou ainda um hiperplano ($> 3D$), que é usado para separar duas categorias de dados distintas.
- Na implementação das portas lógicas AND, OR e NOT, os valores dos pesos e do limiar foram determinados pelo projetista com base na análise geométrica do problema.
- Como fazer com que o neurônio M-P determine de forma automática os valores dos pesos e do limiar para um problema específico?
- Para que o neurônio M-P seja capaz de aprender sozinho a resolver um problema de classificação é necessário dotá-lo de uma **regra de aprendizagem**.
- Uma regra de aprendizagem nada mais é do que uma equação que altera os valores dos pesos e do limiar em função dos erros cometidos durante a execução da tarefa de classificação.



Algoritmo do Perceptron Simples (Definições Preliminares).

- Vamos assumir que existe uma lei matemática, ou função $\mathbf{H}(\cdot)$. Tal função chamada de mapeamento, que relaciona um vetor de entrada $\mathbf{x} \in \mathbb{R}^{p+1}$ qualquer com um vetor de saída $\mathbf{y} \in \mathbb{R}^c$. Matematicamente essa relação pode ser descrita como $\mathbf{y} = \mathbf{H}(\mathbf{x})$.
- Porém, $\mathbf{H}(\cdot)$ é



Algoritmo do Perceptron Simples (Definições Preliminares).

- Vamos assumir que existe uma lei matemática, ou função $\mathbf{H}(\cdot)$. Tal função chamada de mapeamento, que relaciona um vetor de entrada $\mathbf{x} \in \mathbb{R}^{p+1}$ qualquer com um vetor de saída $\mathbf{y} \in \mathbb{R}^c$. Matematicamente essa relação pode ser descrita como $\mathbf{y} = \mathbf{H}(\mathbf{x})$.
- Porém, $\mathbf{H}(\cdot)$ é desconhecida.
- Esse mapeamento pode representar diversos problemas de interesse prático.
 - ① Aproximação de Função.



Algoritmo do Perceptron Simples (Definições Preliminares).

- Vamos assumir que existe uma lei matemática, ou função $\mathbf{H}(\cdot)$. Tal função chamada de mapeamento, que relaciona um vetor de entrada $\mathbf{x} \in \mathbb{R}^{p+1}$ qualquer com um vetor de saída $\mathbf{y} \in \mathbb{R}^c$. Matematicamente essa relação pode ser descrita como $\mathbf{y} = \mathbf{H}(\mathbf{x})$.
- Porém, $\mathbf{H}(\cdot)$ é desconhecida.
- Esse mapeamento pode representar diversos problemas de interesse prático.
 - 1 Aproximação de Função.
 - 2 Classificação de padrões.



Algoritmo do Perceptron Simples (Definições Preliminares).

- Vamos assumir que existe uma lei matemática, ou função $\mathbf{H}(\cdot)$. Tal função chamada de mapeamento, que relaciona um vetor de entrada $\mathbf{x} \in \mathbb{R}^{p+1}$ qualquer com um vetor de saída $\mathbf{y} \in \mathbb{R}^c$. Matematicamente essa relação pode ser descrita como $\mathbf{y} = \mathbf{H}(\mathbf{x})$.
- Porém, $\mathbf{H}(\cdot)$ é desconhecida.
- Esse mapeamento pode representar diversos problemas de interesse prático.
 - 1 Aproximação de Função.
 - 2 Classificação de padrões.
- Aproximação de função, a saída é quantitativa e é normalmente dada por números reais.
- Para classificação, a saída é qualitativa, muitas vezes representadas por +1s e -1s.
- Independente da aplicação, é de desejo a construção de um modelo adaptativo que aproxime a função \mathbf{H} a partir dos pares entrada-saída.



Algoritmo do Perceptron Simples.

- A rede Perceptron Simples (PS) é considerada o primeiro algoritmo de redes neurais artificiais.
- Proposta em 1958 por Frank Rosenblatt em: *ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, v. 65, n. 6, p. 386, 1958.*
- Em sua versão mais simples, trata-se de um neurônio de M-P dotado de



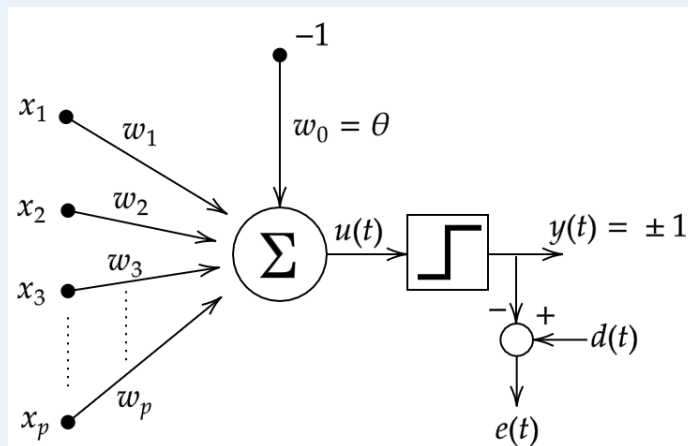
Algoritmo do Perceptron Simples.

- A rede Perceptron Simples (PS) é considerada o primeiro algoritmo de redes neurais artificiais.
- Proposta em 1958 por Frank Rosenblatt em: *ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, v. 65, n. 6, p. 386, 1958.*
- Em sua versão mais simples, trata-se de um neurônio de M-P dotado de uma regra de aprendizagem ou algoritmo de aprendizagem.
- Tal regra é o mecanismo que torna a rede PS um dispositivo inteligente.



Algoritmo do Perceptron Simples.

- A arquitetura do neurônio da rede PS é dada por





Algoritmo do Perceptron Simples.

- Este modelo, significa que para um problema de classificação binário, tem-se:

$$\sum_{j=1}^p w_j x_j \geq \text{limiar} \longrightarrow \textit{Classe1}.$$

$$\sum_{j=1}^p w_j x_j < \text{limiar} \longrightarrow \textit{Classe2}.$$

- Ou então, este pode ser reescrito em uma única equação:



Algoritmo do Perceptron Simples.

- Este modelo, significa que para um problema de classificação binário, tem-se:

$$\sum_{j=1}^p w_j x_j \geq \text{limiar} \longrightarrow \textit{Classe1}.$$

$$\sum_{j=1}^p w_j x_j < \text{limiar} \longrightarrow \textit{Classe2}.$$

- Ou então, este pode ser reescrito em uma única equação:

$$\begin{aligned} y(t) &= \textit{sinal}(u(t)) \\ &= \textit{sinal} \left(\left(\sum_{j=1}^p w_j x_j \right) - \textit{limiar} \right) \end{aligned}$$



Algoritmo do Perceptron Simples.

$$= \text{sinal} \left(\left(\sum_{j=1}^p w_j x_j \right) - \text{limiar} \right)$$



Algoritmo do Perceptron Simples.

$$= \text{sinal} \left(\left(\sum_{j=1}^p w_j x_j \right) - \text{limiar} \right)$$

- Pode-se reescrever $\text{limiar} = \theta = w_0$ e adicionar o artifício $x_0 = -1$.

$$\begin{aligned} y(t) &= \text{sinal} \left(\left(\sum_{j=1}^p w_j x_j \right) + \theta(-1) \right) \\ &= \text{sinal} \left(\left(\sum_{j=1}^p w_j x_j \right) + w_0 x_0 \right) \\ &= \text{sinal} \left(\sum_{j=0}^p w_j x_j \right) = \text{sinal}(\mathbf{w}^T \mathbf{x}) = \text{sinal}(u(t)) \end{aligned}$$



Algoritmo do Perceptron Simples (Discriminante).

- Nesta, os vetores \mathbf{x} e \mathbf{w} são definidos como:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} -1 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$

- Do ponto de vista geométrico, o que representa $u(t)$?



Algoritmo do Perceptron Simples (Discriminante).

- Nesta, os vetores \mathbf{x} e \mathbf{w} são definidos como:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} -1 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$

- Do ponto de vista geométrico, o que representa $u(t)$? Exato!!!



Algoritmo do Perceptron Simples (Discriminante).

- Nesta, os vetores \mathbf{x} e \mathbf{w} são definidos como:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} -1 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$

- Do ponto de vista geométrico, o que representa $u(t)$? Exato!!! Uma **SIMILARIDADE**.
- Sabendo disto, se o ângulo entre \mathbf{x} e \mathbf{w} for menor que 90° , o que dizer sobre $u(t)$?



Algoritmo do Perceptron Simples (Discriminante).

- Nesta, os vetores \mathbf{x} e \mathbf{w} são definidos como:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} -1 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$

- Do ponto de vista geométrico, o que representa $u(t)$? Exato!!! Uma **SIMILARIDADE**.
- Sabendo disto, se o ângulo entre \mathbf{x} e \mathbf{w} for menor que 90° , o que dizer sobre $u(t)$?
- Se o ângulo entre \mathbf{x} e \mathbf{w} for maior que 90° , o que dizer sobre $u(t)$?



Algoritmo do Perceptron Simples (Discriminante).

- Nesta, os vetores \mathbf{x} e \mathbf{w} são definidos como:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} = \begin{bmatrix} -1 \\ x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$

- Do ponto de vista geométrico, o que representa $u(t)$? Exato!!! Uma **SIMILARIDADE**.
- Sabendo disto, se o ângulo entre \mathbf{x} e \mathbf{w} for menor que 90° , o que dizer sobre $u(t)$?
- Se o ângulo entre \mathbf{x} e \mathbf{w} for maior que 90° , o que dizer sobre $u(t)$?

$$y(t) = \text{sin}al(u(t)) = \begin{cases} +1, & u(t) \geq 0 \\ -1, & u(t) < 0 \end{cases}$$



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- O processo de aprendizagem consiste na modificação (ou ajuste) dos parâmetros do neurônio M-P, até que se consiga resolver o problema de interesse ou que se chegue ao final do período de aprendizagem. **Pergunta:** Quais são os parâmetros do modelo?



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- O processo de aprendizagem consiste na modificação (ou ajuste) dos parâmetros do neurônio M-P, até que se consiga resolver o problema de interesse ou que se chegue ao final do período de aprendizagem. **Pergunta:** Quais são os parâmetros do modelo? **Pesos e o limiar**
- A regra de aprendizagem é uma função de dois fatores:
 - 1 Erro entre a saída desejada $d(t)$ e a saída gerada pela rede $y(t)$. Logo, $e(t) = d(t) - y(t)$.
 - 2 Informação fornecida pelo vetor de entrada x .
- O processo de aprendizagem, ou seja, o ajuste dos parâmetros do neurônio M-P, é guiado pelo erro $e(t)$ e pelo vetor de entrada x .



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- Como projetar a regra de aprendizagem?
- Uma regra de aprendizagem pode ser projetada com base em:
 - 1 Argumentos geométricos ou empíricos.
 - 2 Critério de otimização de função-custo.
- Em geral, uma regra de aprendizagem tem a seguinte forma:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta\mathbf{w}(t)$$

- Em que:
 - 1 $\mathbf{w}(t)$ é o conhecimento atual (ou memória).
 - 2 $\Delta\mathbf{w}(t)$ informação adquirida (ou incremento na memória).
 - 3 $\mathbf{w}(t+1)$ memória é modificada com o acréscimo de nova informação.

$$\Delta\mathbf{w}(t) = \mathbf{F}(e(t), \mathbf{x}(t))$$



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- Utilizaremos argumentos **geométricos** para obter a regra de aprendizagem:
- Portanto, quais os possíveis valores que a variável erro $e(t)$ pode assumir?



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- Utilizaremos argumentos **geométricos** para obter a regra de aprendizagem:
- Portanto, quais os possíveis valores que a variável erro $e(t)$ pode assumir?
 - ① $e(t) = d(t) - y(t) = 2$ ($d = +1$ e $y = -1$).



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- Utilizaremos argumentos **geométricos** para obter a regra de aprendizagem:
- Portanto, quais os possíveis valores que a variável erro $e(t)$ pode assumir?
 - 1 $e(t) = d(t) - y(t) = 2$ ($d = +1$ e $y = -1$).
 - 2 $e(t) = d(t) - y(t) = -2$ ($d = -1$ e $y = +1$).



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- Utilizaremos argumentos **geométricos** para obter a regra de aprendizagem:
- Portanto, quais os possíveis valores que a variável erro $e(t)$ pode assumir?
 - ① $e(t) = d(t) - y(t) = 2$ ($d = +1$ e $y = -1$).
 - ② $e(t) = d(t) - y(t) = -2$ ($d = -1$ e $y = +1$).
 - ③ $e(t) = d(t) - y(t) = 0$ ($d = -1$ e $y = -1$) ou ($d = +1$ e $y = +1$).
- Com valores iniciais para \mathbf{w} , o algoritmo deve testar: dado $\mathbf{x}(t)$ se $\text{sin}(\mathbf{u}(t)) \neq d(t)$ é verdadeiro.
- Então ajustar o vetor de pesos de acordo com:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}(t)$$



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- Utilizaremos argumentos **geométricos** para obter a regra de aprendizagem:
- Portanto, quais os possíveis valores que a variável erro $e(t)$ pode assumir?
 - ① $e(t) = d(t) - y(t) = 2$ ($d = +1$ e $y = -1$).
 - ② $e(t) = d(t) - y(t) = -2$ ($d = -1$ e $y = +1$).
 - ③ $e(t) = d(t) - y(t) = 0$ ($d = -1$ e $y = -1$) ou ($d = +1$ e $y = +1$).
- Com valores iniciais para \mathbf{w} , o algoritmo deve testar: dado $\mathbf{x}(t)$ se $\text{sinál}(u(t)) \neq d(t)$ é verdadeiro.
- Então ajustar o vetor de pesos de acordo com:

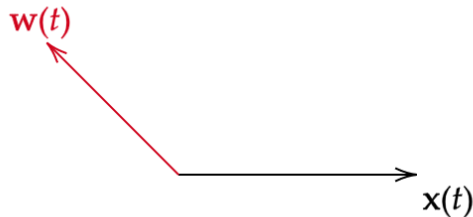
$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}(t)$$

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \Delta \mathbf{w}(t)$$



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

Caso 1: $d = +1$ e $y = -1$

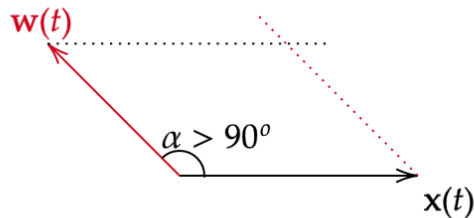


- O que pode ser feito para que y seja igual a d ??



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

Caso 1: $d = +1$ e $y = -1$



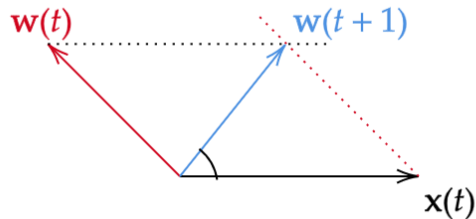
$$u < 0$$
$$y = -1$$

- O que pode ser feito para que y seja igual a d ??



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

Caso 1: $d = +1$ e $y = -1$



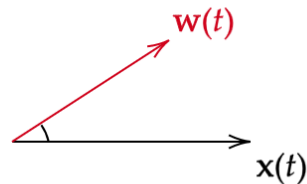
$$u < 0$$
$$y = -1$$

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



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

Caso 2: $d = -1$ e $y = +1$

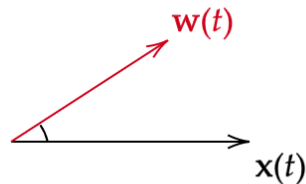


- O que pode ser feito para que y seja igual a d ??



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

Caso 2: $d = -1$ e $y = +1$



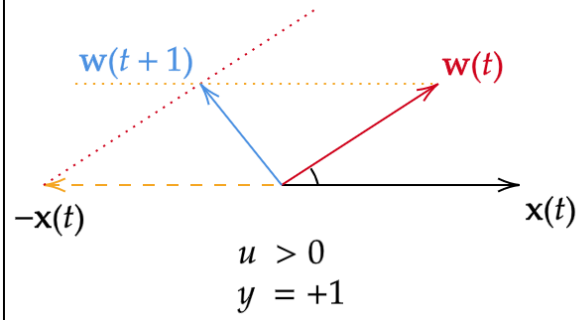
$$\begin{aligned} u &> 0 \\ y &= +1 \end{aligned}$$

- O que pode ser feito para que y seja igual a d ??



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

Caso 2: $d = -1$ e $y = +1$

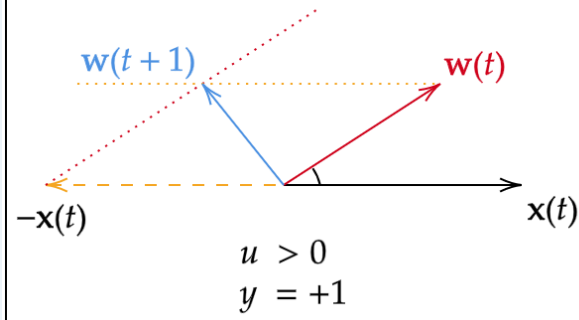


$$w(t+1) = w(t) - x(t)$$



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

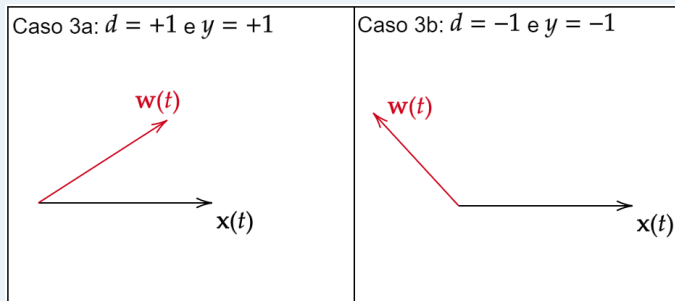
Caso 2: $d = -1$ e $y = +1$



$$w(t+1) = w(t) - x(t)$$



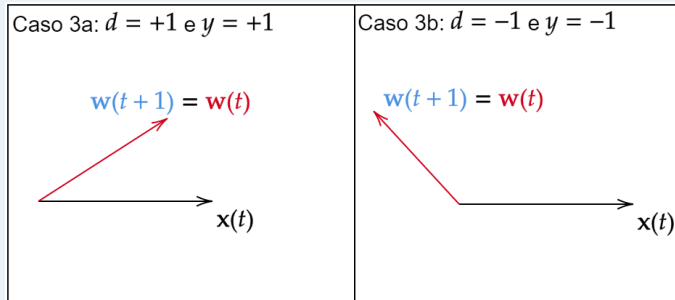
Algoritmo do Perceptron Simples (Regra de Aprendizagem).



- O que deve ser feito nesses casos?



Algoritmo do Perceptron Simples (Regra de Aprendizagem).



$$w(t+1) = w(t)$$



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- As equações mostradas na interpretação geométrica, podem ser combinadas em uma única equação dependente do erro e do vetor de entrada:



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- As equações mostradas na interpretação geométrica, podem ser combinadas em uma única equação dependente do erro e do vetor de entrada:

$$\mathbf{w}(t + 1) = \mathbf{w}(t) + e(t)\mathbf{x}(t)$$

- Qual problemática desta abordagem?



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- As equações mostradas na interpretação geométrica, podem ser combinadas em uma única equação dependente do erro e do vetor de entrada:

$$\mathbf{w}(t + 1) = \mathbf{w}(t) + e(t)\mathbf{x}(t)$$

- Qual problemática desta abordagem?
- Há, portanto, uma maneira de tornar o processo de ajuste do vetor \mathbf{w} mais estável.
- Isto pode ser realizado, ao adicionar um fator de escala η , comumente conhecido como passo, ou taxa de aprendizagem.



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- As equações mostradas na interpretação geométrica, podem ser combinadas em uma única equação dependente do erro e do vetor de entrada:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + e(t)\mathbf{x}(t)$$

- Qual problemática desta abordagem?
- Há, portanto, uma maneira de tornar o processo de ajuste do vetor \mathbf{w} mais estável.
- Isto pode ser realizado, ao adicionar um fator de escala η , comumente conhecido como passo, ou taxa de aprendizagem.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \cdot e(t)\mathbf{x}(t)$$



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- As equações mostradas na interpretação geométrica, podem ser combinadas em uma única equação dependente do erro e do vetor de entrada:

$$\mathbf{w}(t + 1) = \mathbf{w}(t) + e(t)\mathbf{x}(t)$$

- Qual problemática desta abordagem?
- Há, portanto, uma maneira de tornar o processo de ajuste do vetor \mathbf{w} mais estável.
- Isto pode ser realizado, ao adicionar um fator de escala η , comumente conhecido como passo, ou taxa de aprendizagem.

$$\mathbf{w}(t + 1) = \mathbf{w}(t) + \eta \cdot e(t)\mathbf{x}(t)$$

- Em que $0 < \eta \leq 1$.



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

- As equações mostradas na interpretação geométrica, podem ser combinadas em uma única equação dependente do erro e do vetor de entrada:

$$\mathbf{w}(t + 1) = \mathbf{w}(t) + e(t)\mathbf{x}(t)$$

- Qual problemática desta abordagem?
- Há, portanto, uma maneira de tornar o processo de ajuste do vetor \mathbf{w} mais estável.
- Isto pode ser realizado, ao adicionar um fator de escala η , comumente conhecido como passo, ou taxa de aprendizagem.

$$\mathbf{w}(t + 1) = \mathbf{w}(t) + \eta \cdot e(t)\mathbf{x}(t)$$

- Em que $0 < \eta \leq 1$.



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

Algorithm 1: Pseudocódigo para ajuste (fase de treinamento), do perceptron.

```
1: Início ( $t = 0$ )
2: Definir o valor de  $\eta$  entre 0 e 1.
3: Inicializar o vetor de pesos  $\mathbf{w}(t)$  com valores nulos ou aleatórios.
4: ERRO  $\leftarrow$  EXISTE
5: while ERRO == 'EXISTENTE' do
6:   ERRO  $\leftarrow$  'INEXISTE'.
7:   for Todas amostras em  $\mathbf{x}$  do
8:      $u(t) \leftarrow \mathbf{w}^T(t)\mathbf{x}(t)$ 
9:      $y(t) \leftarrow \text{signal}(u(t))$ 
10:     $\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + \eta(d(t) - y(t))\mathbf{x}(t)$ 
11:    if  $d(t) \neq y(t)$  then
12:      ERRO  $\leftarrow$  'EXISTENTE'
13:    end if
14:  end for
15:   $t \leftarrow t + 1$ 
16: end while
17: FIM TREINAMENTO
```



Algoritmo do Perceptron Simples (Regra de Aprendizagem).

Algorithm 2: Pseudocódigo para operação (fase de teste), do perceptron.

- 1: Obter uma amostra ($\mathbf{x}_{desconhecido}$) a ser classificada
- 2: Utilizar o vetor \mathbf{w} já estimado
- 3: Realizar as seguintes operações:
- 4: $u \leftarrow \mathbf{w}^T \mathbf{x}_{desconhecido}$
- 5: $y(t) \leftarrow \text{signal}(u(t))$
- 6: **if** $y == -1$ **then**
- 7: amostra pertence a classe A
- 8: **else**
- 9: amostra pertence a classe B
- 10: **end if**



Exemplo

- Considere o conjunto de dados fornecido

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \\ x_{3,1} & x_{3,2} \\ x_{4,1} & x_{4,2} \\ x_{5,1} & x_{5,2} \\ x_{6,1} & x_{6,2} \\ x_{7,1} & x_{7,2} \\ x_{8,1} & x_{8,2} \\ x_{9,1} & x_{9,2} \\ x_{10,1} & x_{10,2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 2 \\ 1 & 0 \\ 2 & 2 \\ 4 & 1.5 \\ 1.5 & 6 \\ 3 & 5 \\ 3 & 3 \\ 6 & 4 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$



Modelo ADALINE.

- Trata-se de um tipo elementar de algoritmo adaptativo.
- Seu nome original é *ADaptive LINear Element* (ou em português, Elemento Linear Adaptativo).
- O presente modelo foi proposto por Widrow & Hoff (1960).
- O modelo ADALINE têm seus parâmetros ajustados por meio de uma regra de atualização recursiva:



Modelo ADALINE.

- Trata-se de um tipo elementar de algoritmo adaptativo.
- Seu nome original é *ADaptive LINear Element* (ou em português, Elemento Linear Adaptativo).
- O presente modelo foi proposto por Widrow & Hoff (1960).
- O modelo ADALINE têm seus parâmetros ajustados por meio de uma regra de atualização recursiva:
 - 1 Regra de Widrow-Hoff.



Modelo ADALINE.

- Trata-se de um tipo elementar de algoritmo adaptativo.
- Seu nome original é *ADaptive LINear Element* (ou em português, Elemento Linear Adaptativo).
- O presente modelo foi proposto por Widrow & Hoff (1960).
- O modelo ADALINE têm seus parâmetros ajustados por meio de uma regra de atualização recursiva:
 - 1 Regra de Widrow-Hoff.
 - 2 Regra Delta.

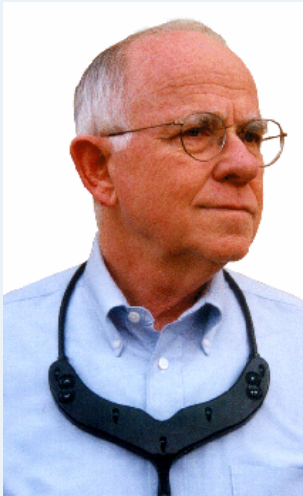


Modelo ADALINE.

- Trata-se de um tipo elementar de algoritmo adaptativo.
- Seu nome original é *ADaptive LINear Element* (ou em português, Elemento Linear Adaptativo).
- O presente modelo foi proposto por Widrow & Hoff (1960).
- O modelo ADALINE têm seus parâmetros ajustados por meio de uma regra de atualização recursiva:
 - 1 Regra de Widrow-Hoff.
 - 2 Regra Delta.
 - 3 Algoritmos de adaptação LMS (Least Mean Squares).

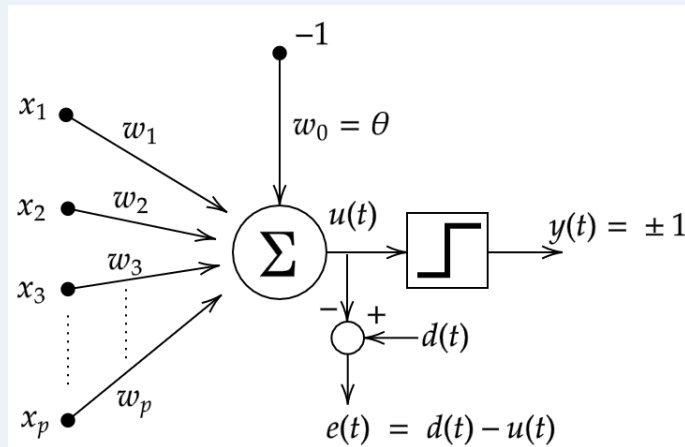


Widrow-Hoff.





Modelo ADALINE.



- Há diferença entre o perceptron?



Modelo ADALINE.

- Em que o vetor de entradas e o de pesos, são definidos como:

$$\mathbf{x}(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \\ x_2(t) \\ \vdots \\ x_p(t) \end{bmatrix} = \begin{bmatrix} -1 \\ x_1(t) \\ x_2(t) \\ \vdots \\ x_p(t) \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$

- A saída desejada $d(t)$ do presente modelo tem qual ordem?



Modelo ADALINE.

- Em que o vetor de entradas e o de pesos, são definidos como:

$$\mathbf{x}(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \\ x_2(t) \\ \vdots \\ x_p(t) \end{bmatrix} = \begin{bmatrix} -1 \\ x_1(t) \\ x_2(t) \\ \vdots \\ x_p(t) \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} \theta \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix}$$

- A saída desejada $d(t)$ do presente modelo tem qual ordem?
- Isto porque se trata apenas de **UM** único neurônio. Para uma rede com múltiplos neurônios, o modelo passa a se chamar MADALINE.
- Em princípio a saída desejada pode assumir qualquer valor real.
- Contudo, em problemas de classificação de padrões a saída desejada assume geralmente apenas dois valores $d \in \{-1, +1\}$



Modelo ADALINE.

- Quais são os parâmetros ajustáveis do modelo ADALINE?



Modelo ADALINE.

- Quais são os parâmetros ajustáveis do modelo ADALINE?
- O vetor de pesos \mathbf{w} .
- Qual diferença entre o PS e o ADALINE?



Modelo ADALINE.

- Quais são os parâmetros ajustáveis do modelo ADALINE?
- O vetor de pesos \mathbf{w} .
- Qual diferença entre o PS e o ADALINE?

$$u(t) = \left(\sum_{j=1}^p w_j(t)x_j(t) \right) - \theta = \left(\sum_{j=1}^p w_j(t)x_j(t) \right) - w_0x_0 = \sum_{j=0}^p w_j(t)x_j(t) = \mathbf{w}^T(t)\mathbf{x}(t)$$

- Assim, $u(t)$ é simplesmente o produto escalar



Modelo ADALINE.

- Quais são os parâmetros ajustáveis do modelo ADALINE?
- O vetor de pesos \mathbf{w} .
- Qual diferença entre o PS e o ADALINE?

$$u(t) = \left(\sum_{j=1}^p w_j(t)x_j(t) \right) - \theta = \left(\sum_{j=1}^p w_j(t)x_j(t) \right) - w_0x_0 = \sum_{j=0}^p w_j(t)x_j(t) = \mathbf{w}^T(t)\mathbf{x}(t)$$

- Assim, $u(t)$ é simplesmente o produto escalar do vetor de entradas $\mathbf{x}(t)$ com o vetor de pesos $\mathbf{w}(t)$
- Como dito, $u(t) \in \mathbb{R}$, ou seja, pode assumir infinitos valores.
- Quantização escalar é o processo de transformar a saída contínua $u(t)$ em discreta $y(t) \in \{+1, -1\}$
- Já utilizamos alguma função quantizadora?



Modelo ADALINE.

- Uma função quantizadora bastante utilizada em reconhecimento de padrões é construída com a função sinal (*sign function*)
- **Dica importante:** A codificação das saídas desejadas, deve ser compatível com a saída quantizadora.



Regra de Aprendizagem.

- Definições iniciais:
 - ① A precisão instantânea (ou seja, no instante t) do modelo ADALINE é medida com base no **Erro Quadrático(EQ)**:

$$\varepsilon(t) = \frac{1}{2}e^2(t) = \frac{1}{2}(d(t) - u(t))^2$$

- Em que $e(t) = d(t) - u(t)$ é o erro associado à apresentação do par entrada-saída $(\mathbf{x}(t), d(t))$.



Regra de Aprendizagem.

- A regra de aprendizagem, é baseada na minimização de uma medida global do desempenho.
- Esta medida é chamada de **Erro Quadrático Médio**(EQM), que é produzida para **todos** os pares entrada-saída $(\mathbf{x}(t), d(t))$:

$$J[\mathbf{w}] = \frac{1}{N} \sum_{t=1}^N \varepsilon(t) = \frac{1}{2N} \sum_{t=1}^N e^2(t) = \frac{1}{2N} \sum_{t=1}^N [d(t) - u(t)]^2$$

- Em que \mathbf{w} denota o conjunto de todos os parâmetros ajustáveis do modelo.
- Os parâmetros do modelo ADALINE devem ser especificados de modo que este produza uma saída bem próxima da esperada para um vetor de entrada $\mathbf{x}(t)$.
- Ou seja, identificar um \mathbf{w} ótimo (\mathbf{w}^*) que minimize o EQM.



Regra de Aprendizagem.

- Um procedimento iterativo de se chegar aos parâmetros ótimos envolve o uso da equação recursiva:

$$w_j(t+1) = w_j(t) - \eta \frac{\partial \varepsilon(t)}{\partial w_j(t)}$$

- Em que η é a taxa de aprendizagem $0 < \eta < 1$.
- Utilizando a regra da cadeia, na derivada exibida, pode-se fazer:



Regra de Aprendizagem.

- Um procedimento iterativo de se chegar aos parâmetros ótimos envolve o uso da equação recursiva:

$$w_j(t+1) = w_j(t) - \eta \frac{\partial \varepsilon(t)}{\partial w_j(t)}$$

- Em que η é a taxa de aprendizagem $0 < \eta < 1$.
- Utilizando a regra da cadeia, na derivada exibida, pode-se fazer:

$$\frac{\partial \varepsilon(t)}{\partial w_j(t)} = \frac{\partial \varepsilon(t)}{\partial e(t)} \cdot \frac{\partial e(t)}{\partial u(t)} \cdot \frac{\partial u(t)}{\partial w_j(t)}$$



Regra de Aprendizagem.

$$\frac{\partial \varepsilon(t)}{\partial e(t)} = e(t)$$

$$\frac{\partial e(t)}{\partial u(t)} = -1$$

$$\frac{\partial u(t)}{\partial w_j(t)} = x_j(t)$$

- Assim, a regra recursiva de ajuste dos pesos é dada por:

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

$$w_j(t+1) = w_j(t) + \eta e(t) x_j(t)$$



Regra de Aprendizagem.

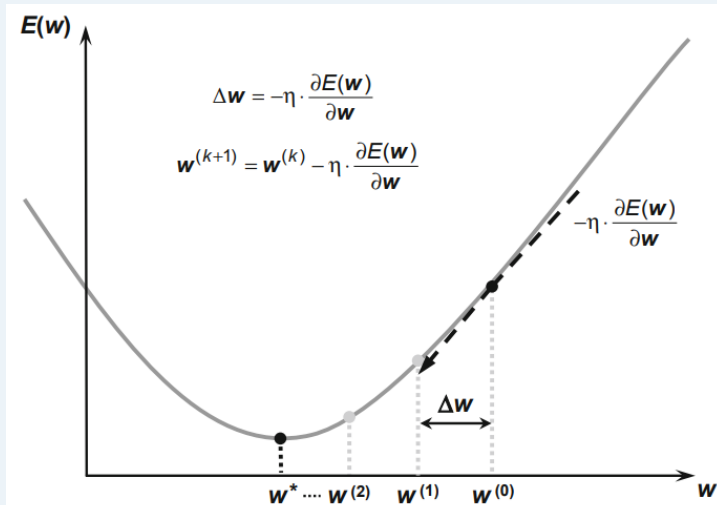
- De maneira vetorial, a regra de ajuste do pesos pode ser escrita como:

$$\begin{aligned}\mathbf{w}(t+1) &= \mathbf{w}(t) + \eta \Delta J[\mathbf{w}] \\ &= \mathbf{w}(t) + \eta e(t) \mathbf{x}(t)\end{aligned}$$

- É possível verificar a interpretação geométrica deste ajuste de pesos da seguinte maneira:



Regra de Aprendizagem.





Regra de Aprendizagem.

- Pontos importantes:

- 1 Em sua etapa de treinamento, o modelo ADALINE deve ser interrompido quando a convergência acontecer, ou seja: Quando a diferença dos EQM entre duas épocas sucessivas for suficientemente pequeno:

$$|EQM_{atual} - EQM_{anterior}| \leq \epsilon$$

- 2 Onde ϵ é a precisão a ser definida pelo projetista da rede ADALINE.
- Outro critério de parada, envolve simplesmente a definição de um número máximo de épocas.
 - É de interesse verificar a curva de aprendizagem do modelo, dado o problema proposto. Para tal feito, deve-se para cada época plotar valores dos EQM calculados.



Algoritmo ADALINE (Treinamento).

Algorithm 3: Pseudocódigo para ajuste (fase de treinamento), do ADALINE.

- 1: Definir o valor de η , número máximo de épocas e precisão (ϵ).
- 2: Inicializar o vetor de pesos $\mathbf{w}(t)$ com valores nulos **ou** aleatórios.
- 3: Iniciar o contador de épocas ($epoch \leftarrow 0$)
- 4: **repeat**
- 5: $EQM_{anterior} \leftarrow EQM(\mathbf{x}, d, \mathbf{w})$
- 6: **for** todas as N amostras de treinamento **do**
- 7: $u(t) \leftarrow \mathbf{w}^T(t)\mathbf{x}(t)$
- 8: $\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + \eta(d(t) - u(t))\mathbf{x}(t)$
- 9: **end for**
- 10: $epoch \leftarrow epoch + 1$
- 11: $EQM_{atual} \leftarrow EQM(\mathbf{x}, d, \mathbf{w})$
- 12: **until** $|EQM_{atual} - EQM_{anterior}| \leq \epsilon$ OU Número máximo de épocas atingido



Algoritmo ADALINE (Treinamento).

Algorithm 4: Algoritmo para cálculo do EQM .

```
1:  $EQM \leftarrow 0$   
2: for todas as amostras de treinamento do  
3:    $u(t) \leftarrow \mathbf{w}^T(t)\mathbf{x}(t)$   
4:    $EQM \leftarrow EQM + (d(t) - u(t))^2$   
5: end for  
6:  $EQM \leftarrow \frac{EQM}{2N}$ 
```



Algoritmo do ADALINE (Regra de Aprendizagem).

Algorithm 5: Pseudocódigo para operação (fase de teste), do ADALINE.

- 1: Obter uma amostra ($\mathbf{x}_{desconhecido}$) a ser classificada
- 2: Utilizar o vetor \mathbf{w} já estimado
- 3: Realizar as seguintes operações:
- 4: $u \leftarrow \mathbf{w}^T \mathbf{x}_{desconhecido}$
- 5: $y(t) \leftarrow \text{signal}(u(t))$
- 6: **if** $y == -1$ **then**
- 7: amostra pertence a classe A
- 8: **else**
- 9: amostra pertence a classe B
- 10: **end if**



Dicas importantes ao trabalhar com estes algoritmos.

- Normalizar os vetores de entrada se as variáveis apresentarem ordens de grandeza desigual.
- Logo, a normalização dos dados equaliza as ordens de grandeza dos atributos usados em um problema de classificação.



Métodos 1: Norma constante

- Consiste em manter constante e igual a 1, as normas dos vetores de atributos \mathbf{x}

$$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$$

- O que acontece com o vetor quando esta normalização é aplicada?



Métodos 1: Norma constante

- Consiste em manter constante e igual a 1, as normas dos vetores de atributos \mathbf{x}

$$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$$

- O que acontece com o vetor quando esta normalização é aplicada?
- Mudança apenas no comprimento, ou seja, o vetor resultante é múltiplo do vetor original.

$$\mathbf{x}_{normalizado} = \frac{1}{\|\mathbf{x}\|} \mathbf{x} = \alpha \mathbf{x}$$

- Esta é uma normalização do tipo **local**, ou seja,



Métodos 1: Norma constante

- Consiste em manter constante e igual a 1, as normas dos vetores de atributos \mathbf{x}

$$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$$

- O que acontece com o vetor quando esta normalização é aplicada?
- Mudança apenas no comprimento, ou seja, o vetor resultante é múltiplo do vetor original.

$$\mathbf{x}_{normalizado} = \frac{1}{\|\mathbf{x}\|} \mathbf{x} = \alpha \mathbf{x}$$

- Esta é uma normalização do tipo **local**, ou seja, depende apenas dos valores das componentes do vetor.



Métodos 2: Mudança de escala

- Procedimento é realizado variável a variável e requer a determinação do valor mínimo e valor máximo da variável
- Este, portanto, trata-se de um procedimento de normalização **global**.
- na faixa $[0, +1]$:

$$x_j^{norm} = \frac{x_j - \min(\mathbf{X})}{\max(\mathbf{X}) - \min(\mathbf{X})}$$

- na faixa $[-1, +1]$:

$$x_j^{norm} = 2 \cdot \left(\frac{x_j - \min(\mathbf{X})}{\max(\mathbf{X}) - \min(\mathbf{X})} \right) - 1$$



Métodos 3: Padronização da variável (média=0, variância =1)

- A normalização estatística é dada por

$$x_j^{norm} = \frac{x_j - \bar{x}}{\sigma_x}$$

$$\bar{x} = \frac{\sum_{i=1}^p x_i}{p}$$

$$\sigma_x = \sqrt{\left(\frac{\sum_{i=1}^p (x_i - \bar{x})^2}{p - 1} \right)}$$



Informações importantes

- Por se tratarem de transformações lineares, as normalizações descritas não alteram a natureza da distribuição da variável normalizada em relação à variável original.
- Em outras palavras, o tipo de PDF (*Probability Density Function*) da variável permanece o mesmo. Ex: se PDF for gaussiana, continua gaussiana após a transformação.



Avaliar o desempenho de classificadores ou RNA

- Qual medida discutida até agora para avaliar o desempenho?
- É um procedimento aplicado em qual etapa?



Avaliar o desempenho de classificadores ou RNA

- Qual medida discutida até agora para avaliar o desempenho?
- É um procedimento aplicado em qual etapa?

$$TA = \frac{\text{Quantidade de amostras de teste classificadas corretamente}}{\text{Número total de vetores de atributos}}$$

- Contudo, uma análise com mais medidas pode ser realizada ao construir uma matriz de confusão.
- Trata-se de uma matriz de ordem $c \times c$, porém a sua versão elementar é 2×2 .



Avaliar o desempenho de classificadores ou RNA

		Real	
		Condição Positiva	Condição Negativa
Predito	Condição Positiva	Verdadeiro Positivo [1,1]	Falso Positivo [1,-1]
	Condição Negativa	Falso Negativo [-1,1]	Verdadeiro Negativo [-1,-1]



Avaliar o desempenho de classificadores ou RNA

- **VP** e **VN** representam respectivamente a quantidade de predições corretas para a condição positiva e negativa .
- **FP** é a quantidade de predições erradas para a condição real negativa.
- **FN** é a quantidade de predições realizadas de maneira errada para a condição real positiva.

		Real	
		Condição Positiva	Condição Negativa
Predito	Condição Positiva	Verdadeiro Positivo [1,1]	Falso Positivo [1,-1]
	Condição Negativa	Falso Negativo [-1,1]	Verdadeiro Negativo [-1,-1]



Avaliar o desempenho de classificadores ou RNA

- Desta matriz, pode-se extrair diversas medidas, porém, destacam-se:

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN}$$

$$\text{Sensibilidade} = \frac{VP}{VP + FN}$$

$$\text{Especificidade} = \frac{VN}{VN + FP}$$



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).
- São caracterizadas pela presença de pelo menos uma camada intermediária (conhecida como *hidden layer*) de neurônios.



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).
- São caracterizadas pela presença de pelo menos uma camada intermediária (conhecida como *hidden layer*) de neurônios.
- Tal camada se situa entre as camadas de entrada e saída.



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).
- São caracterizadas pela presença de pelo menos uma camada intermediária (conhecida como *hidden layer*) de neurônios.
- Tal camada se situa entre as camadas de entrada e saída.
- Desta maneira, as redes MLP possuem no mínimo duas camadas de neurônios.



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).
- São caracterizadas pela presença de pelo menos uma camada intermediária (conhecida como *hidden layer*) de neurônios.
- Tal camada se situa entre as camadas de entrada e saída.
- Desta maneira, as redes MLP possuem no mínimo duas camadas de neurônios.
- É um poderoso algoritmo de aprendizado de máquina com diversas aplicações em diferentes problemas.
 - ① Aproximação universal de funções.



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).
- São caracterizadas pela presença de pelo menos uma camada intermediária (conhecida como *hidden layer*) de neurônios.
- Tal camada se situa entre as camadas de entrada e saída.
- Desta maneira, as redes MLP possuem no mínimo duas camadas de neurônios.
- É um poderoso algoritmo de aprendizado de máquina com diversas aplicações em diferentes problemas.
 - 1 Aproximação universal de funções.
 - 2 Reconhecimento de padrões.



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).
- São caracterizadas pela presença de pelo menos uma camada intermediária (conhecida como *hidden layer*) de neurônios.
- Tal camada se situa entre as camadas de entrada e saída.
- Desta maneira, as redes MLP possuem no mínimo duas camadas de neurônios.
- É um poderoso algoritmo de aprendizado de máquina com diversas aplicações em diferentes problemas.
 - 1 Aproximação universal de funções.
 - 2 Reconhecimento de padrões.
 - 3 Identificação e controle de processos.



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).
- São caracterizadas pela presença de pelo menos uma camada intermediária (conhecida como *hidden layer*) de neurônios.
- Tal camada se situa entre as camadas de entrada e saída.
- Desta maneira, as redes MLP possuem no mínimo duas camadas de neurônios.
- É um poderoso algoritmo de aprendizado de máquina com diversas aplicações em diferentes problemas.
 - 1 Aproximação universal de funções.
 - 2 Reconhecimento de padrões.
 - 3 Identificação e controle de processos.
 - 4 Previsão de séries temporais.
 - 5 Otimização de sistemas.



Redes Perceptron de Multicamadas

- Comumente conhecidas como Multilayer Perceptron (MLP).
- São caracterizadas pela presença de pelo menos uma camada intermediária (conhecida como *hidden layer*) de neurônios.
- Tal camada se situa entre as camadas de entrada e saída.
- Desta maneira, as redes MLP possuem no mínimo duas camadas de neurônios.
- É um poderoso algoritmo de aprendizado de máquina com diversas aplicações em diferentes problemas.
 - 1 Aproximação universal de funções.
 - 2 Reconhecimento de padrões.
 - 3 Identificação e controle de processos.
 - 4 Previsão de séries temporais.
 - 5 Otimização de sistemas.
- Além disto, apresenta-se como ferramenta no tratamento de problemas não-lineares, que exigem o mapeamento entrada-saída não-linear.



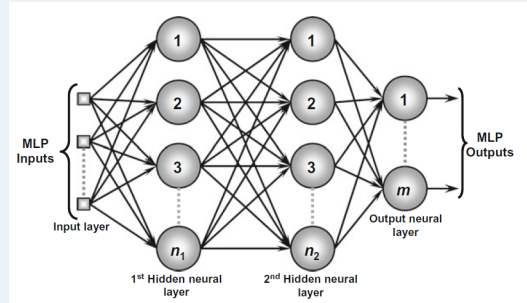
Redes Perceptron de Multicamadas

- A rede MLP é categorizada como uma pertencente a classe de redes *feedforward*, cujo treinamento é realizado de maneira **supervisionada**.



Redes Perceptron de Multicamadas

- A rede MLP é categorizada como uma pertencente a classe de redes *feedforward*, cujo treinamento é realizado de maneira **supervisionada**.



- Conforme ilustrado, o fluxo de informações na estrutura da rede se inicia na camada de entrada, e finaliza na camada neural de saída.



Arquitetura das Redes Perceptron de Multicamadas

- É constituída de um conjunto de unidades entradas (que recebem os sinais)



Arquitetura das Redes Perceptron de Multicamadas

- É constituída de um conjunto de unidades entradas (que recebem os sinais)
- Uma ou mais camadas ocultas (ou escondidas) compostas por **neurônios não-lineares**.



Arquitetura das Redes Perceptron de Multicamadas

- É constituída de um conjunto de unidades entradas (que recebem os sinais)
- Uma ou mais camadas ocultas (ou escondidas) compostas por **neurônios não-lineares**.
- Uma saída composta por um ou mais neurônios que podem ser lineares ou não-lineares.



Arquitetura das Redes Perceptron de Multicamadas

- É constituída de um conjunto de unidades entradas (que recebem os sinais)
- Uma ou mais camadas ocultas (ou escondidas) compostas por **neurônios não-lineares**.
- Uma saída composta por um ou mais neurônios que podem ser lineares ou não-lineares.
- Em uma rede MLP padrão, não há a existência de qualquer tipo de realimentação dos valores produzidos pela camada neural de saídas, ou pelas intermediárias.



Arquitetura das Redes Perceptron de Multicamadas

- É constituída de um conjunto de unidades entradas (que recebem os sinais)
- Uma ou mais camadas ocultas (ou escondidas) compostas por **neurônios não-lineares**.
- Uma saída composta por um ou mais neurônios que podem ser lineares ou não-lineares.
- Em uma rede MLP padrão, não há a existência de qualquer tipo de realimentação dos valores produzidos pela camada neural de saídas, ou pelas intermediárias.
- A sua popularidade, se deu com o trabalho: **RUMELHART**, David E. et al. Parallel distributed processing. New York: IEEE, 1988.

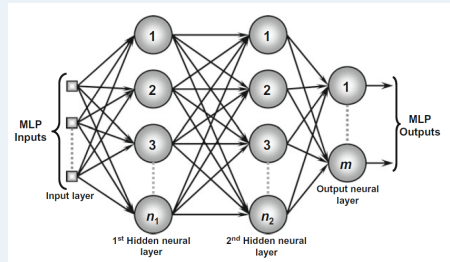


Arquitetura das Redes Perceptron de Multicamadas

- É constituída de um conjunto de unidades entradas (que recebem os sinais)
- Uma ou mais camadas ocultas (ou escondidas) compostas por **neurônios não-lineares**.
- Uma saída composta por um ou mais neurônios que podem ser lineares ou não-lineares.
- Em uma rede MLP padrão, não há a existência de qualquer tipo de realimentação dos valores produzidos pela camada neural de saídas, ou pelas intermediárias.
- A sua popularidade, se deu com o trabalho: **RUMELHART**, David E. et al. Parallel distributed processing. New York: IEEE, 1988.
- Neste trabalho, explicitou-se o algoritmo de aprendizagem denominado *backpropagation*.



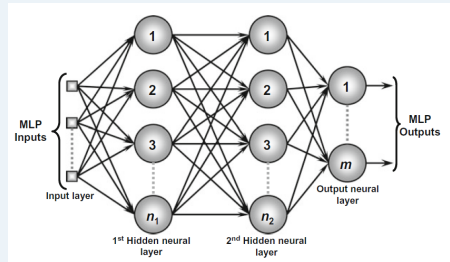
Redes Perceptron de Multicamadas



- A figura mostra a arquitetura de rede MLP *feedforward* com duas camadas ocultas e **totalmente** conectada.



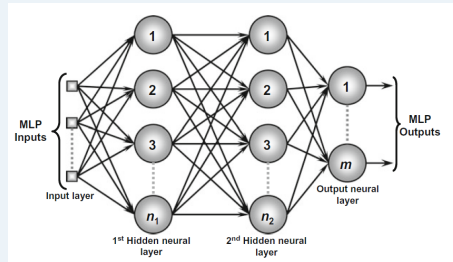
Redes Perceptron de Multicamadas



- A figura mostra a arquitetura de rede MLP *feedforward* com duas camadas ocultas e **totalmente** conectada.
- Isto significa que, um neurônio em qualquer camada da rede é conectado à todas as unidades/neurônios da camada anterior e o sinal progride da esquerda para direita (em sua fase de operação).



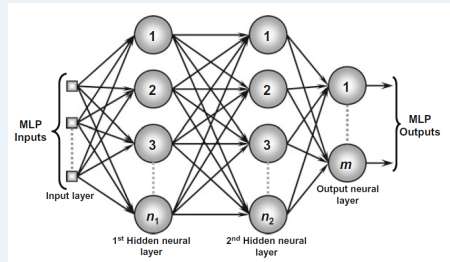
Arquitetura Redes Perceptron de Multicamadas



- Neste caso, as saídas dos neurônios da primeira camada oculta, serão as próprias entradas daqueles neurônios pertencentes à segunda camada neural escondida.



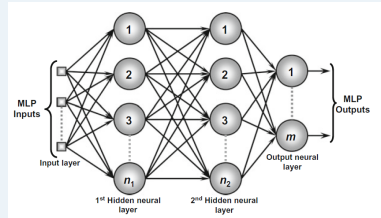
Arquitetura Redes Perceptron de Multicamadas



- Neste caso, as saídas dos neurônios da primeira camada oculta, serão as próprias entradas daqueles neurônios pertencentes à segunda camada neural escondida.
- Para a figura exibida, as saídas dos neurônios da segunda camada neural escondida, serão as respectivas entradas dos neurônios presentes na camada de saída.



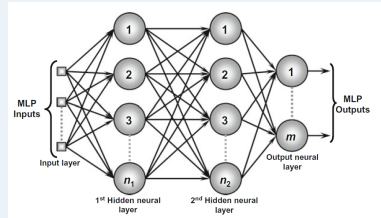
Arquitetura Redes Perceptron de Multicamadas



- Diferente das redes PS e ADALINE, a rede MLP



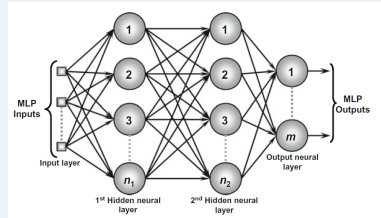
Arquitetura Redes Perceptron de Multicamadas



- Diferente das redes PS e ADALINE, a rede MLP pode conter diversos neurônios na camada de saída, sendo cada saída representada como a saída do processo a ser mapeado.



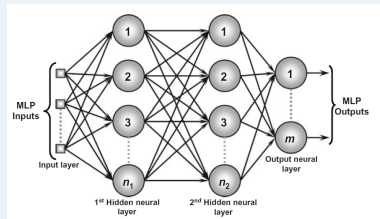
Arquitetura Redes Perceptron de Multicamadas



- Diferente das redes PS e ADALINE, a rede MLP pode conter diversos neurônios na camada de saída, sendo cada saída representada como a saída do processo a ser mapeado.
- Pode-se então definir m de duas maneiras amplamente utilizadas:
 - 1 $m = C$, sendo C a quantidade de classes (1-out-of- Q).



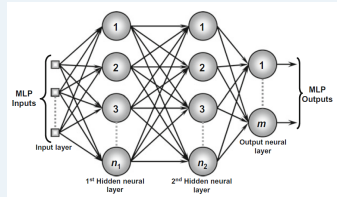
Arquitetura Redes Perceptron de Multicamadas



- Diferente das redes PS e ADALINE, a rede MLP pode conter diversos neurônios na camada de saída, sendo cada saída representada como a saída do processo a ser mapeado.
- Pode-se então definir m de duas maneiras amplamente utilizadas:
 - 1 $m = C$, sendo C a quantidade de classes (1-out-of- Q).
 - 2 m é o maior inteiro igual ou menor que \sqrt{C} ($m \geq \sqrt{C}$).



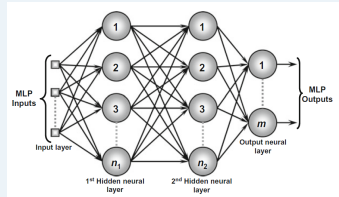
Arquitetura Redes Perceptron de Multicamadas



- Ainda falando no contraste em relação ao PS e ADALINE, em que um único neurônio era responsável pelo mapeamento, o conhecimento relacionado ao comportamento entrada/saída é **distribuído** por todos os neurônios na rede MLP.



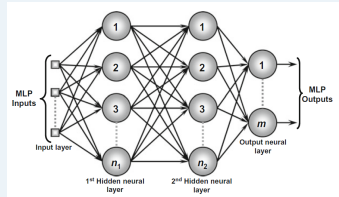
Arquitetura Redes Perceptron de Multicamadas



- Ainda falando no contraste em relação ao PS e ADALINE, em que um único neurônio era responsável pelo mapeamento, o conhecimento relacionado ao comportamento entrada/saída é **distribuído** por todos os neurônios na rede MLP.
- Conforme o processo de aprendizagem da rede avança, estes neurônios começam a "descobrir" as características salientes presentes nos dados.



Arquitetura Redes Perceptron de Multicamadas



- Ainda falando no contraste em relação ao PS e ADALINE, em que um único neurônio era responsável pelo mapeamento, o conhecimento relacionado ao comportamento entrada/saída é **distribuído** por todos os neurônios na rede MLP.
- Conforme o processo de aprendizagem da rede avança, estes neurônios começam a "descobrir" as características salientes presentes nos dados.
- Em seguida, realizam uma transformação não-linear nos dados de entrada para um novo espaço, onde as classes de interesse podem ser mais facilmente separadas.



Resumo inicial - Redes Perceptron de Multicamadas

- **Unidades de Entrada:**



Resumo inicial - Redes Perceptron de Multicamadas

- **Unidades de Entrada:** responsáveis pela simples passagem dos valores de entrada para os neurônios das camadas seguintes.
- **Camada(s) oculta(s):** contém neurônios responsáveis pelo processamento não-linear da informação de entrada, de modo a facilitar a resolução do problema.



Resumo inicial - Redes Perceptron de Multicamadas

- **Unidades de Entrada:** responsáveis pela simples passagem dos valores de entrada para os neurônios das camadas seguintes.
- **Camada(s) oculta(s):** contém neurônios responsáveis pelo processamento não-linear da informação de entrada, de modo a facilitar a resolução do problema.
- **Camada de saída:** contém os neurônios responsáveis pela geração da saída da rede neural, após as entradas terem sido devidamente processadas pelos neurônios ocultos.



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **uma** camada oculta é representada por:

$$\text{MLP}(p, q_1, m)$$



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **uma** camada oculta é representada por:

$$\mathbf{MLP}(p, q_1, m)$$

- 1 p é o número de variáveis de entrada.



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **uma** camada oculta é representada por:

$$\text{MLP}(p, q_1, m)$$

- 1 p é o número de variáveis de entrada.
- 2 q_1 é o número de neurônios ocultos.



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **uma** camada oculta é representada por:

$$\text{MLP}(p, q_1, m)$$

- 1 p é o número de variáveis de entrada.
- 2 q_1 é o número de neurônios ocultos.
- 3 m é o número de neurônios de saída.



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **uma** camada oculta é representada por:

$$\text{MLP}(p, q_1, m)$$

- 1 p é o número de variáveis de entrada.
 - 2 q_1 é o número de neurônios ocultos.
 - 3 m é o número de neurônios de saída.
- Desta maneira, a quantidade de parâmetros (Z) para ajuste é dado por:

$$Z = (p + 1)q_1 + (q_1 + 1)m$$



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **duas** camadas ocultas é representada por:

$$\text{MLP}(p, q_1, q_2, m)$$



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **duas** camadas ocultas é representada por:

$$\text{MLP}(p, q_1, q_2, m)$$

- 1 p é o número de variáveis de entrada.



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **duas** camadas ocultas é representada por:

$$\text{MLP}(p, q_1, q_2, m)$$

- 1 p é o número de variáveis de entrada.
- 2 q_1 é o número de neurônios ocultos na primeira camada.



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **duas** camadas ocultas é representada por:

$$\text{MLP}(p, q_1, q_2, m)$$

- 1 p é o número de variáveis de entrada.
- 2 q_1 é o número de neurônios ocultos na primeira camada.
- 3 q_2 é o número de neurônios ocultos na segunda camada.



Construção de Rede Perceptron de Multicamadas

- Uma rede MLP com **duas** camadas ocultas é representada por:

$$\text{MLP}(p, q_1, q_2, m)$$

- 1 p é o número de variáveis de entrada.
 - 2 q_1 é o número de neurônios ocultos na primeira camada.
 - 3 q_2 é o número de neurônios ocultos na segunda camada.
 - 4 m é o número de neurônios de saída.
- Desta maneira, a quantidade de parâmetros (Z) para ajuste é dado por:

$$Z = (p + 1)q_1 + (q_1 + 1)q_2 + (q_2 + 1)m$$



Construção de Rede Perceptron de Multicamadas

- Notas importantes:
 - 1 A especificação de p e m são ditadas pela forma como o problema é codificado para ser resolvido por uma RNA.



Construção de Rede Perceptron de Multicamadas

- Notas importantes:
 - 1 A especificação de p e m são ditadas pela forma como o problema é codificado para ser resolvido por uma RNA.
 - 2 A especificação dos valores de q_1, q_2, \dots, q_o dependem da complexidade do problema, ou seja, é preciso realizar vários testes até encontrar os valores mais adequados.



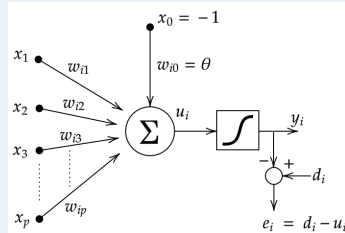
Construção de Rede Perceptron de Multicamadas

- Notas importantes:
 - 1 A especificação de p e m são ditadas pela forma como o problema é codificado para ser resolvido por uma RNA.
 - 2 A especificação dos valores de q_1, q_2, \dots, q_o dependem da complexidade do problema, ou seja, é preciso realizar vários testes até encontrar os valores mais adequados.
- Todavia para ambos os casos existem regras que definem as quantidades dos neurônios na(s) camada(s) ocultas e de saída.
- Veremos estas regras ao final deste conteúdo.



Construção de Rede Perceptron de Multicamadas

- Neurônio artificial da rede MLP:
- Um neurônio qualquer da rede MLP, seja oculto ou de saída é representado genericamente como na figura abaixo



- Qual diferença deste modelo para o de M-P?



Construção de Rede Perceptron de Multicamadas

- A função de ativação!



Construção de Rede Perceptron de Multicamadas

- A função de ativação!
- No modelo M-P a função é do tipo **degrau**(*hard*)



Construção de Rede Perceptron de Multicamadas

- A função de ativação!
- No modelo M-P a função é do tipo **degrau**(*hard*)
- No modelo MLP a função de ativação é do tipo **sigmoidal**(*soft*).



Construção de Rede Perceptron de Multicamadas

- A função de ativação!
- No modelo M-P a função é do tipo **degrau**(*hard*)
- No modelo MLP a função de ativação é do tipo **sigmoidal**(*soft*).
- Assim, a saída deixa de ser uma variável do tipo **ON-OFF** (binária[0,1] ou bipolar[-1,+1]).



Construção de Rede Perceptron de Multicamadas

- A função de ativação!
- No modelo M-P a função é do tipo **degrau**(*hard*)
- No modelo MLP a função de ativação é do tipo **sigmoidal**(*soft*).
- Assim, a saída deixa de ser uma variável do tipo **ON-OFF** (binária[0,1] ou bipolar[-1,+1]).
- Passando a ser uma variável **real** ou **analógica** (qualquer valor entre [0,1] ou [-1,+1]).

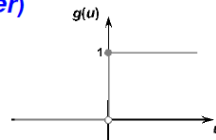


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Funções parcialmente diferenciáveis.

❑ Função degrau (*heavyside/hard limiter*)

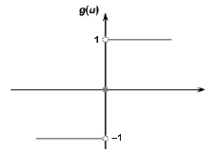
$$g(u) = \begin{cases} 1, & \text{se } u \geq 0 \\ 0, & \text{se } u < 0 \end{cases}$$



❑ Função degrau bipolar ou sinal (*symmetric hard limiter*)

$$g(u) = \begin{cases} 1, & \text{se } u > 0 \\ 0, & \text{se } u = 0 \\ -1, & \text{se } u < 0 \end{cases} \quad \text{ou} \quad g(u) = \begin{cases} 1, & \text{se } u \geq 0 \\ -1, & \text{se } u < 0 \end{cases}$$

P/ classificação



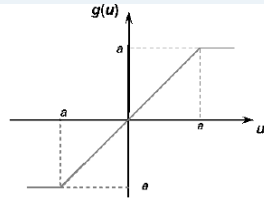


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Funções parcialmente diferenciáveis.

□ Função rampa simétrica

$$g(u) = \begin{cases} a, & \text{se } u > a \\ u, & \text{se } -a \leq u \leq a \\ -a, & \text{se } u < -a \end{cases}$$



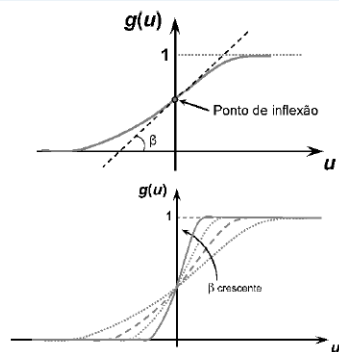


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Funções totalmente diferenciáveis.

□ Função logística

$$g(u) = \frac{1}{1 + e^{-\beta \cdot u}} \quad \beta > 0$$



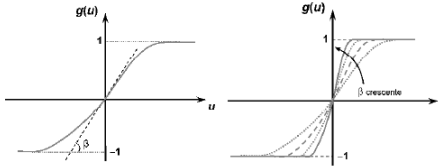


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Funções totalmente diferenciáveis.

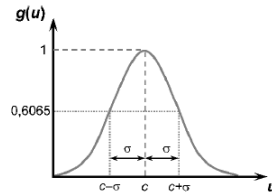
☐ Função tangente hiperbólica

$$g(u) = \frac{1 - e^{-\beta \cdot u}}{1 + e^{-\beta \cdot u}} \quad \beta > 0$$



☐ Função gaussiana

$$g(u) = e^{-\frac{(u-c)^2}{2\sigma^2}} \quad \sigma \neq 0$$



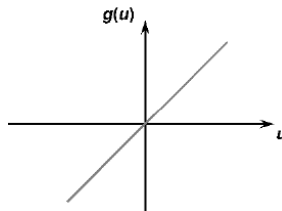


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Função Identidade.

❑ Função linear (identidade)

$$g(u) = u$$



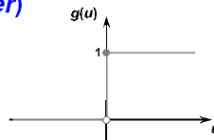


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Funções parcialmente diferenciáveis.

❑ Função degrau (*heavyside/hard limiter*)

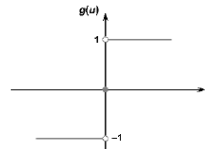
$$g(u) = \begin{cases} 1, & \text{se } u \geq 0 \\ 0, & \text{se } u < 0 \end{cases}$$



❑ Função degrau bipolar ou sinal (*symmetric hard limiter*)

$$g(u) = \begin{cases} 1, & \text{se } u > 0 \\ 0, & \text{se } u = 0 \\ -1, & \text{se } u < 0 \end{cases} \quad \text{ou} \quad g(u) = \begin{cases} 1, & \text{se } u \geq 0 \\ -1, & \text{se } u < 0 \end{cases}$$

P/ classificação



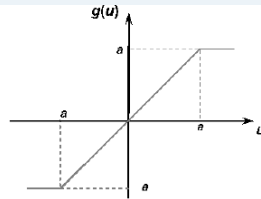


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Funções parcialmente diferenciáveis.

□ Função rampa simétrica

$$g(u) = \begin{cases} a, & \text{se } u > a \\ u, & \text{se } -a \leq u \leq a \\ -a, & \text{se } u < -a \end{cases}$$



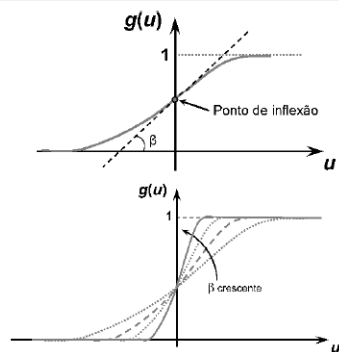


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Funções totalmente diferenciáveis.

□ Função logística

$$g(u) = \frac{1}{1 + e^{-\beta \cdot u}} \quad \beta > 0$$



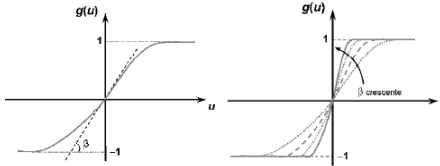


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Funções totalmente diferenciáveis.

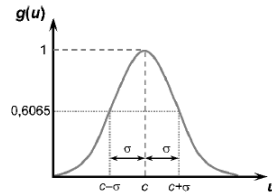
☐ Função tangente hiperbólica

$$g(u) = \frac{1 - e^{-\beta \cdot u}}{1 + e^{-\beta \cdot u}} \quad \beta > 0$$



☐ Função gaussiana

$$g(u) = e^{-\frac{(u-c)^2}{2\sigma^2}} \quad \sigma \neq 0$$



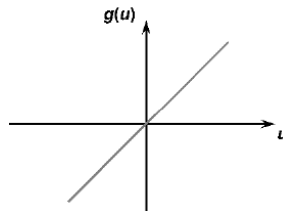


Neurônio de Mculloch - Pitts.

- A função de ativação $y(t)$ não se limita apenas ao degrau bipolar.
- Função Identidade.

□ Função linear (identidade)

$$g(u) = u$$





Construção de Rede Perceptron de Multicamadas

- Duas funções sigmoidais amplamente utilizadas são:



Construção de Rede Perceptron de Multicamadas

- Duas funções sigmoidais amplamente utilizadas são:
- **Sigmóide Logística:**

$$y_i = \frac{1}{1 + \exp(-u_i)} \quad y_i \in (0, 1)$$



Construção de Rede Perceptron de Multicamadas

- Duas funções sigmoidais amplamente utilizadas são:

- **Sigmóide Logística:**

$$y_i = \frac{1}{1 + \exp(-u_i)} \quad y_i \in (0, 1)$$

- Sua derivada vale:

$$y_i' = \frac{dy_i}{du_i} = y_i(1 - y_i)$$



Construção de Rede Perceptron de Multicamadas

- Duas funções sigmoidais amplamente utilizadas são:

- **Sigmóide Logística:**

$$y_i = \frac{1}{1 + \exp(-u_i)} \quad y_i \in (0, 1)$$

- **Tangente Hiperbólica:**

$$y_i = \frac{1 - \exp(-u_i)}{1 + \exp(-u_i)} \quad y_i \in (-1, 1)$$

- Sua derivada vale:

$$y_i' = \frac{dy_i}{du_i} = y_i(1 - y_i)$$



Construção de Rede Perceptron de Multicamadas

- Duas funções sigmoidais amplamente utilizadas são:

- **Sigmóide Logística:**

$$y_i = \frac{1}{1 + \exp(-u_i)} \quad y_i \in (0, 1)$$

- Sua derivada vale:

$$y'_i = \frac{dy_i}{du_i} = y_i(1 - y_i)$$

- **Tangente Hiperbólica:**

$$y_i = \frac{1 - \exp(-u_i)}{1 + \exp(-u_i)} \quad y_i \in (-1, 1)$$

- Sua derivada vale:

$$y'_i = \frac{dy_i}{du_i} = 0,5 * (1 - y_i^2)$$



Sobre o uso de funções sigmoidais:

- Vantagens:
 - 1 Derivadas fáceis de calcular.



Sobre o uso de funções sigmoidais:

- Vantagens:
 - 1 Derivadas fáceis de calcular.
 - 2 Interpretação da saída como taxa média de disparo, em vez de simplesmente indicar se o neurônio está ou não ativado.



Sobre o uso de funções sigmoidais:

- Vantagens:
 - 1 Derivadas fáceis de calcular.
 - 2 Interpretação da saída como taxa média de disparo, em vez de simplesmente indicar se o neurônio está ou não ativado.
- Desvantagens:



Sobre o uso de funções sigmoidais:

- Vantagens:
 - 1 Derivadas fáceis de calcular.
 - 2 Interpretação da saída como taxa média de disparo, em vez de simplesmente indicar se o neurônio está ou não ativado.
- Desvantagens:
 - 1 Elevado custo computacional para implementação em sistemas embarcados devido à presença



Sobre o uso de funções sigmoidais:

- Vantagens:
 - 1 Derivadas fáceis de calcular.
 - 2 Interpretação da saída como taxa média de disparo, em vez de simplesmente indicar se o neurônio está ou não ativado.
- Desvantagens:
 - 1 Elevado custo computacional para implementação em sistemas embarcados devido à presença da função exponencial.

$$\exp(x) = \sum_{i=0}^{i=\infty} \frac{x^i}{i!}$$



Processo de treinamento da rede MLP

- O processo de treinamento utilizando o algoritmo do *backpropagation*, é comumente conhecido como a regra Delta generalizada.



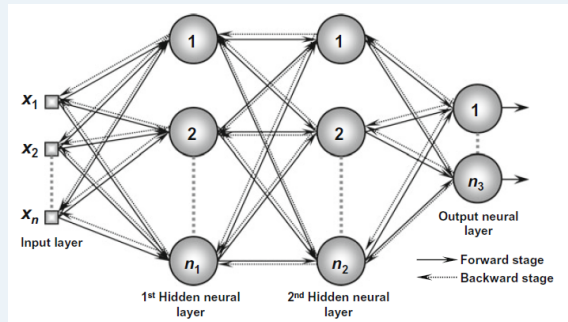
Processo de treinamento da rede MLP

- O processo de treinamento utilizando o algoritmo do *backpropagation*, é comumente conhecido como a regra Delta generalizada.
- Este processo é realizado mediante as aplicações de duas fases bem específicas.



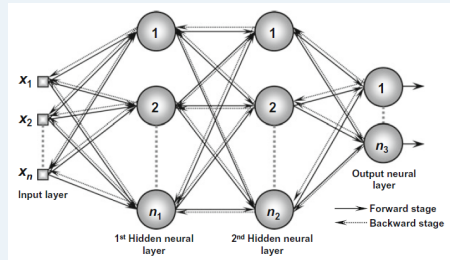
Processo de treinamento da rede MLP

- O processo de treinamento utilizando o algoritmo do *backpropagation*, é comumente conhecido como a regra Delta generalizada.
- Este processo é realizado mediante as aplicações de duas fases bem específicas.





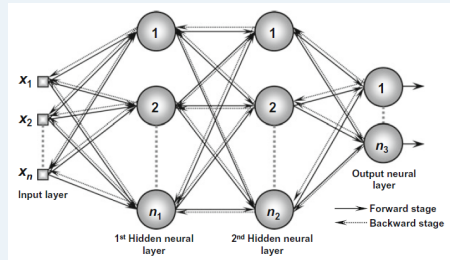
Processo de treinamento da rede MLP



- A primeira fase a ser aplicada é denominada propagação adiante (*forward*).



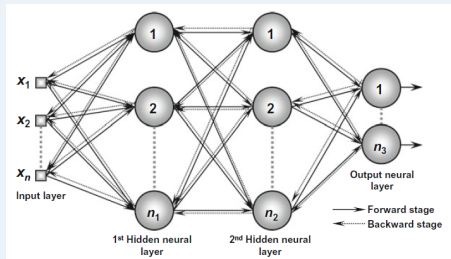
Processo de treinamento da rede MLP



- A primeira fase a ser aplicada é denominada propagação adiante (*forward*).
- Em que os sinais de entrada $\{x_1, x_2, \dots, x_p\}$ de uma amostra do conjunto de treinamento, são inseridos nas entradas da rede.



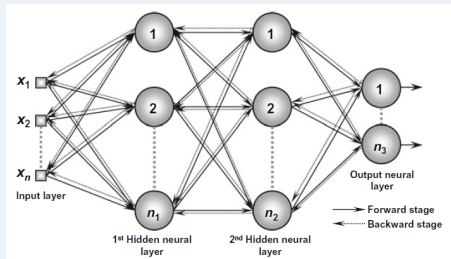
Processo de treinamento da rede MLP



- A primeira fase a ser aplicada é denominada propagação adiante (*forward*).
- Em que os sinais de entrada $\{x_1, x_2, \dots, x_p\}$ de uma amostra do conjunto de treinamento, são inseridos nas entradas da rede.
- Estes, são propagados camada a camada até a produção da saída.



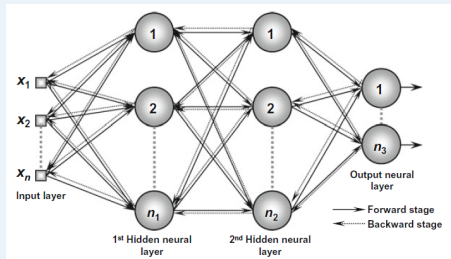
Processo de treinamento da rede MLP



- A primeira fase a ser aplicada é denominada propagação adiante (*forward*).
- Em que os sinais de entrada $\{x_1, x_2, \dots, x_p\}$ de uma amostra do conjunto de treinamento, são inseridos nas entradas da rede.
- Estes, são propagados camada a camada até a produção da saída.
- Neste processo, os pesos sinápticos e os limiares não são ajustados.



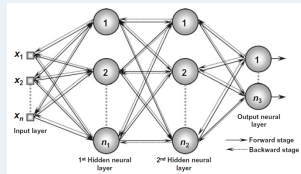
Processo de treinamento da rede MLP



- A primeira fase a ser aplicada é denominada propagação adiante (*forward*).
- Em que os sinais de entrada $\{x_1, x_2, \dots, x_p\}$ de uma amostra do conjunto de treinamento, são inseridos nas entradas da rede.
- Estes, são propagados camada a camada até a produção da saída.
- Neste processo, os pesos sinápticos e os limiares não são ajustados.



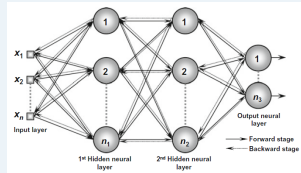
Processo de treinamento da rede MLP



- Em sequência, as respostas produzidas pela rede são comparadas com as respostas desejadas, produzindo assim um sinal de erro.



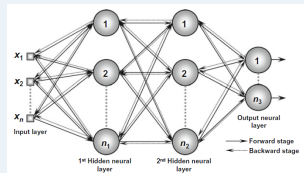
Processo de treinamento da rede MLP



- Em sequência, as respostas produzidas pela rede são comparadas com as respostas desejadas, produzindo assim um sinal de erro.
- Com isto, inicia-se a segunda fase do método *backpropagation*, denominada propagação reversa (*backward*).



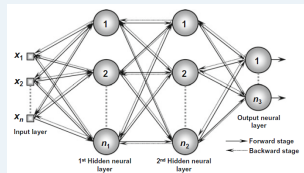
Processo de treinamento da rede MLP



- Em sequência, as respostas produzidas pela rede são comparadas com as respostas desejadas, produzindo assim um sinal de erro.
- Com isto, inicia-se a segunda fase do método *backpropagation*, denominada propagação reversa (*backward*).
- Nesta etapa, os pesos e limiares **são** ajustados.



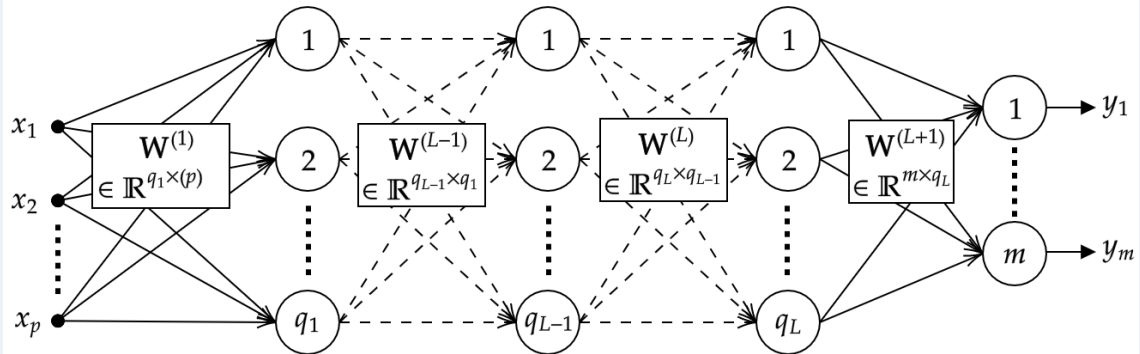
Processo de treinamento da rede MLP



- Em sequência, as respostas produzidas pela rede são comparadas com as respostas desejadas, produzindo assim um sinal de erro.
- Com isto, inicia-se a segunda fase do método *backpropagation*, denominada propagação reversa (*backward*).
- Nesta etapa, os pesos e limiares **são** ajustados.
- As aplicações sucessivas das fases *forward* e *backward* fazem com que os pesos sinápticos e limiares dos neurônios se ajustem automaticamente em cada iteração, de modo a diminuir a soma dos erros produzidos pela resposta da rede.

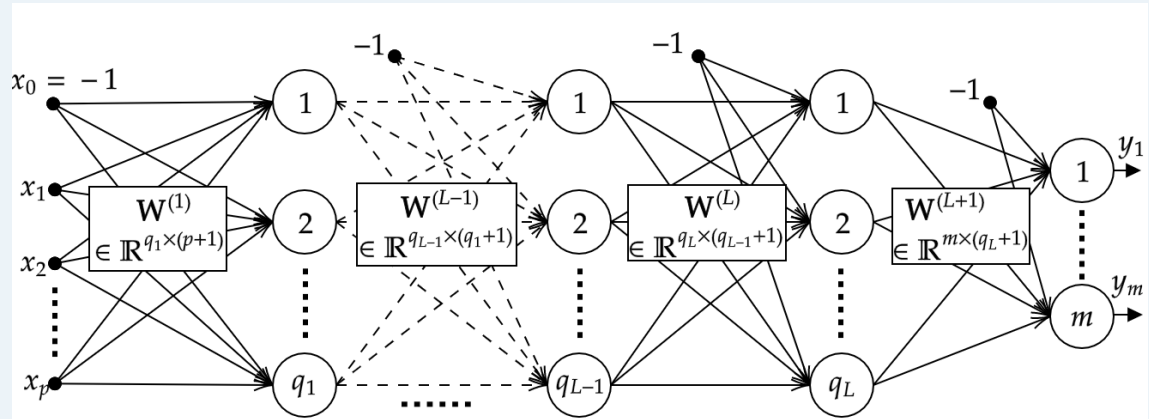


Derivação do algoritmo *backpropagation* - Sentido Direto



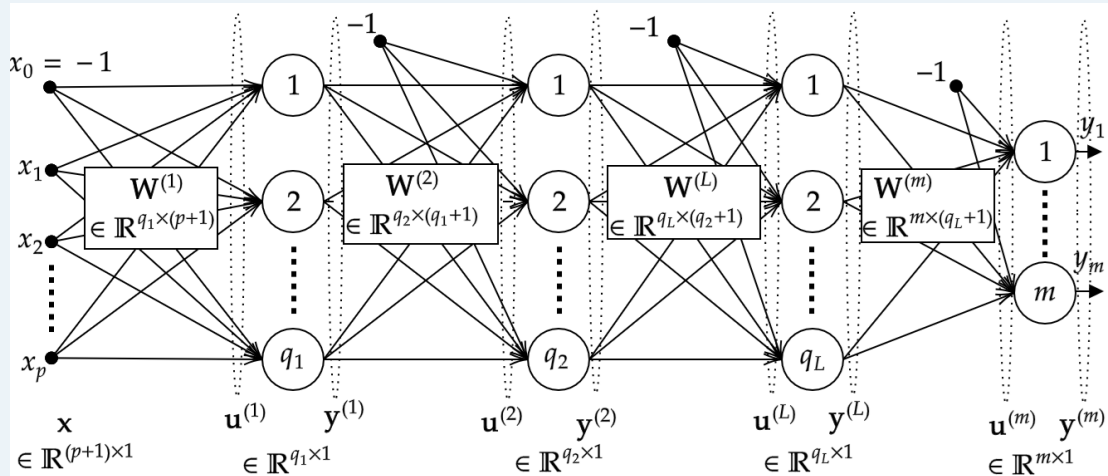


Derivação do algoritmo *backpropagation*- Sentido Direto



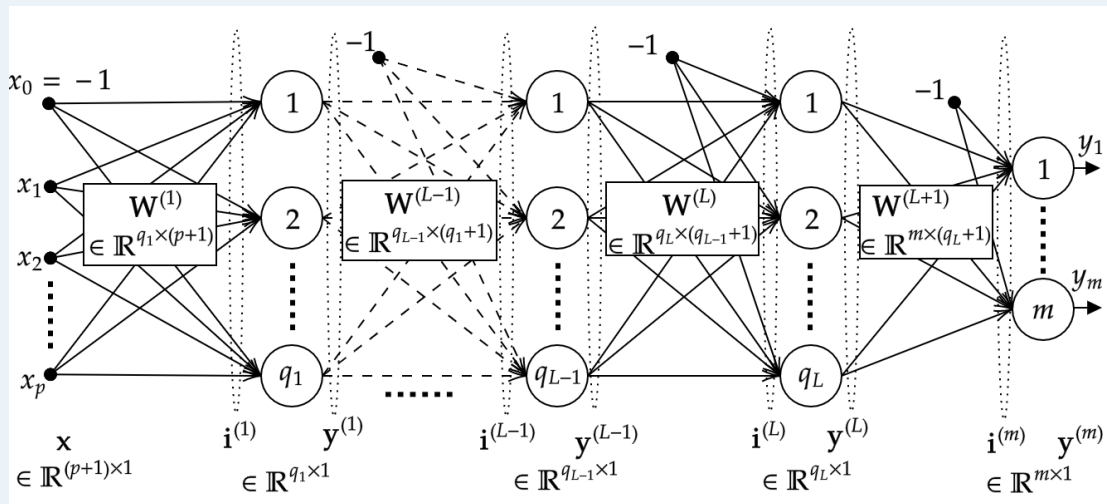


Derivação do algoritmo *backpropagation*- Sentido Direto





Derivação do algoritmo *backpropagation*- Sentido Direto





Derivação do algoritmo *backpropagation*- Sentido Direto

- Nesta topologia, pode-se definir as seguintes terminologias:
 - 1 $\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$ incluindo o limiar de ativação, é a N -ésima amostra apresentada na iteração.



Derivação do algoritmo *backpropagation*- Sentido Direto

- Nesta topologia, pode-se definir as seguintes terminologias:
 - 1 $\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$ incluindo o limiar de ativação, é a N -ésima amostra apresentada na iteração.
 - 2 $\mathbf{W}^{(L)}$ é a L -ésima matriz de pesos que deve ser ajustada no processo de treinamento.



Derivação do algoritmo *backpropagation*- Sentido Direto

- Nesta topologia, pode-se definir as seguintes terminologias:
 - 1 $\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$ incluindo o limiar de ativação, é a N -ésima amostra apresentada na iteração.
 - 2 $\mathbf{W}^{(L)}$ é a L -ésima matriz de pesos que deve ser ajustada no processo de treinamento.
 - 3 \mathbf{i}^L é o L -ésimo vetor de entrada ponderada para a L -ésima camada.



Derivação do algoritmo *backpropagation*- Sentido Direto

- Nesta topologia, pode-se definir as seguintes terminologias:
 - 1 $\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$ incluindo o limiar de ativação, é a N -ésima amostra apresentada na iteração.
 - 2 $\mathbf{W}^{(L)}$ é a L -ésima matriz de pesos que deve ser ajustada no processo de treinamento.
 - 3 \mathbf{i}^L é o L -ésimo vetor de entrada ponderada para a L -ésima camada.
 - 4 \mathbf{y}^L é o L -ésimo vetor de saída após a aplicação da função de ativação em cada neurônio na camada L .



Derivação do algoritmo *backpropagation*- Sentido Direto

- Nesta topologia, pode-se definir as seguintes terminologias:
 - 1 $\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$ incluindo o limiar de ativação, é a N -ésima amostra apresentada na iteração.
 - 2 $\mathbf{W}^{(L)}$ é a L -ésima matriz de pesos que deve ser ajustada no processo de treinamento.
 - 3 \mathbf{i}^L é o L -ésimo vetor de entrada ponderada para a L -ésima camada.
 - 4 \mathbf{y}^L é o L -ésimo vetor de saída após a aplicação da função de ativação em cada neurônio na camada L .
$$\mathbf{y}^{(1)} = g(\mathbf{i}^{(1)}) \quad \mathbf{y}^{(L-1)} = g(\mathbf{i}^{(L-1)}) \quad \mathbf{y}^{(L)} = g(\mathbf{i}^{(L)}) \quad \mathbf{y}^{(m)} = g(\mathbf{i}^{(m)})$$



Derivação do algoritmo *backpropagation*- Sentido Direto

- Nesta topologia, pode-se definir as seguintes terminologias:
 - 1 $\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$ incluindo o limiar de ativação, é a N -ésima amostra apresentada na iteração.
 - 2 $\mathbf{W}^{(L)}$ é a L -ésima matriz de pesos que deve ser ajustada no processo de treinamento.
 - 3 \mathbf{i}^L é o L -ésimo vetor de entrada ponderada para a L -ésima camada.
 - 4 \mathbf{y}^L é o L -ésimo vetor de saída após a aplicação da função de ativação em cada neurônio na camada L .

$$\mathbf{y}^{(1)} = g(\mathbf{i}^{(1)}) \quad \mathbf{y}^{(L-1)} = g(\mathbf{i}^{(L-1)}) \quad \mathbf{y}^{(L)} = g(\mathbf{i}^{(L)}) \quad \mathbf{y}^{(m)} = g(\mathbf{i}^{(m)})$$

$$\mathbf{i}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} \quad \mathbf{i}^{(L-1)} = \mathbf{W}^{(L-1)}\mathbf{y}^{(1)} \quad \mathbf{i}^{(L)} = \mathbf{W}^{(L)}\mathbf{y}^{(L-1)} \quad \mathbf{i}^{(m)} = \mathbf{W}^{(L+1)}\mathbf{y}^{(L)}$$



Derivação do algoritmo *backpropagation*- Sentido Direto

- Nesta topologia, pode-se definir as seguintes terminologias:
 - 1 $\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$ incluindo o limiar de ativação, é a N -ésima amostra apresentada na iteração.
 - 2 $\mathbf{W}^{(L)}$ é a L -ésima matriz de pesos que deve ser ajustada no processo de treinamento.
 - 3 \mathbf{i}^L é o L -ésimo vetor de entrada ponderada para a L -ésima camada.
 - 4 \mathbf{y}^L é o L -ésimo vetor de saída após a aplicação da função de ativação em cada neurônio na camada L .

$$\mathbf{y}^{(1)} = g(\mathbf{i}^{(1)}) \quad \mathbf{y}^{(L-1)} = g(\mathbf{i}^{(L-1)}) \quad \mathbf{y}^{(L)} = g(\mathbf{i}^{(L)}) \quad \mathbf{y}^{(m)} = g(\mathbf{i}^{(m)})$$

$$\mathbf{i}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} \quad \mathbf{i}^{(L-1)} = \mathbf{W}^{(L-1)}\mathbf{y}^{(1)} \quad \mathbf{i}^{(L)} = \mathbf{W}^{(L)}\mathbf{y}^{(L-1)} \quad \mathbf{i}^{(m)} = \mathbf{W}^{(L+1)}\mathbf{y}^{(L)}$$

- $g(\cdot)$ neste exemplo, representa a função de ativação escolhida no projeto da rede.



Derivação do algoritmo *backpropagation*- Sentido Direto

- Nesta topologia, pode-se definir as seguintes terminologias:

- 1 $\mathbf{x} \in \mathbb{R}^{(p+1) \times 1}$ incluindo o limiar de ativação, é a N -ésima amostra apresentada na iteração.
- 2 $\mathbf{W}^{(L)}$ é a L -ésima matriz de pesos que deve ser ajustada no processo de treinamento.
- 3 \mathbf{i}^L é o L -ésimo vetor de entrada ponderada para a L -ésima camada.
- 4 \mathbf{y}^L é o L -ésimo vetor de saída após a aplicação da função de ativação em cada neurônio na camada L .

$$\mathbf{y}^{(1)} = g(\mathbf{i}^{(1)}) \quad \mathbf{y}^{(L-1)} = g(\mathbf{i}^{(L-1)}) \quad \mathbf{y}^{(L)} = g(\mathbf{i}^{(L)}) \quad \mathbf{y}^{(m)} = g(\mathbf{i}^{(m)})$$

$$\mathbf{i}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} \quad \mathbf{i}^{(L-1)} = \mathbf{W}^{(L-1)}\mathbf{y}^{(1)} \quad \mathbf{i}^{(L)} = \mathbf{W}^{(L)}\mathbf{y}^{(L-1)} \quad \mathbf{i}^{(m)} = \mathbf{W}^{(L+1)}\mathbf{y}^{(L)}$$

- $g(\cdot)$ neste exemplo, representa a função de ativação escolhida no projeto da rede.
- Pode-se verificar na Figura que a saída m , não possui a adição do termo $y_0^{(m)} = -1$. Alguém arrisca dizer o motivo disto?



Derivação do algoritmo *backpropagation*- Sentido Reverso (função custo)

- Após a computação do sinal real no final da rede MLP, pode-se inicializar a derivação do algoritmo *backpropagation* ao definir a função representativa do erro de aproximação.



Derivação do algoritmo *backpropagation*- Sentido Reverso (função custo)

- Após a computação do sinal real no final da rede MLP, pode-se inicializar a derivação do algoritmo *backpropagation* ao definir a função representativa do erro de aproximação.
- Esta tem o papel de medir o desvio entre as respostas produzidas pelos neurônios na camada de **saída** em relação aos valores desejados.



Derivação do algoritmo *backpropagation*- Sentido Reverso (função custo)

- Após a computação do sinal real no final da rede MLP, pode-se inicializar a derivação do algoritmo *backpropagation* ao definir a função representativa do erro de aproximação.
- Esta tem o papel de medir o desvio entre as respostas produzidas pelos neurônios na camada de **saída** em relação aos valores desejados.
- Portanto, o sentido reverso da rede começa na camada de saída, com a determinação do erro produzido por cada um dos neurônios de saída.



Derivação do algoritmo *backpropagation*- Sentido Reverso (função custo)

- Após a computação do sinal real no final da rede MLP, pode-se inicializar a derivação do algoritmo *backpropagation* ao definir a função representativa do erro de aproximação.
- Esta tem o papel de medir o desvio entre as respostas produzidas pelos neurônios na camada de **saída** em relação aos valores desejados.
- Portanto, o sentido reverso da rede começa na camada de saída, com a determinação do erro produzido por cada um dos neurônios de saída.

$$e_k = d_k - y_k, \quad k = 1, \dots, m$$

- Em que d_k é a saída desejada para o k -ésimo neurônio da camada de saída.



Derivação do algoritmo *backpropagation*- Sentido Reverso (função custo)

- Após a computação do sinal real no final da rede MLP, pode-se inicializar a derivação do algoritmo *backpropagation* ao definir a função representativa do erro de aproximação.
- Esta tem o papel de medir o desvio entre as respostas produzidas pelos neurônios na camada de **saída** em relação aos valores desejados.
- Portanto, o sentido reverso da rede começa na camada de saída, com a determinação do erro produzido por cada um dos neurônios de saída.

$$e_k = d_k - y_k, \quad k = 1, \dots, m$$

- Em que d_k é a saída desejada para o k -ésimo neurônio da camada de saída.
- O valor instantâneo da energia do erro para o k -ésimo neurônio de saída é dado por $\frac{1}{2}e_k^2$.



Derivação do algoritmo *backpropagation*- Sentido Reverso (função custo)

- Após a computação do sinal real no final da rede MLP, pode-se inicializar a derivação do algoritmo *backpropagation* ao definir a função representativa do erro de aproximação.
- Esta tem o papel de medir o desvio entre as respostas produzidas pelos neurônios na camada de **saída** em relação aos valores desejados.
- Portanto, o sentido reverso da rede começa na camada de saída, com a determinação do erro produzido por cada um dos neurônios de saída.

$$e_k = d_k - y_k, \quad k = 1, \dots, m$$

- Em que d_k é a saída desejada para o k -ésimo neurônio da camada de saída.
- O valor instantâneo da energia do erro para o k -ésimo neurônio de saída é dado por $\frac{1}{2}e_k^2$.



Derivação do algoritmo *backpropagation*- Sentido Reverso (função custo)

- O valor instantâneo **total** do erro (Erro Quadrático Instantâneo), pode ser obtido somando-se os termos $\frac{1}{2}e_k^2$ para todos os m neurônios de saída:

$$J(t) = \frac{1}{2} \sum_{k=1}^m e_k^2 = \frac{1}{2} \sum_{k=1}^m (d_k - y_k)^2 \quad (1)$$

- para a obtenção da regra de aprendizagem para a rede MLP, a função custo de interesse é o **Erro Quadrático Médio** (EQM) calculado para os N vetores de treinamento, e é dado por

$$EQM = \frac{1}{N} \sum_{t=1}^N J(t) = \frac{1}{2N} \sum_{t=1}^N \sum_{k=1}^m (d_k(t) - y_k(t))^2$$

- Pretende-se então a minimização desta função ao realizar a minimização de $J(t)$.



Derivação do algoritmo *backpropagation*- Sentido Reverso (minimização da função custo)

- O algoritmo de retropropagação do erro aplica uma correção Δw_{ki}^m ao peso sináptico w_{ki}^m , que é proporcional ao gradiente da função $J(t)$, expresso por $\partial J(t)/\partial w_{ki}^m$.



Derivação do algoritmo *backpropagation*- Sentido Reverso (minimização da função custo)

- O algoritmo de retropropagação do erro aplica uma correção Δw_{ki}^m ao peso sináptico w_{ki}^m , que é proporcional ao gradiente da função $J(t)$, expresso por $\partial J(t)/\partial w_{ki}^m$.
- Para calcular esta derivada faz-se o uso da regra da cadeia para fatorá-la em vários termos.

$$\frac{\partial J(t)}{\partial w_{ji}^m} = \frac{\partial J(t)}{\partial e_k(t)} \frac{\partial e_k(t)}{\partial y_k(t)} \frac{\partial y_k(t)}{\partial u_k(t)} \frac{\partial u_k(t)}{\partial w_{ki}^m(t)}$$



Derivação do algoritmo *backpropagation*- Sentido Reverso (minimização da função custo)

- Cada derivada parcial pode ser calculada como

$$\frac{\partial J(t)}{\partial e_k(t)} = e_k(t)$$

$$\frac{\partial e_k(t)}{\partial y_k(t)} = -1$$

$$\frac{\partial y_k(t)}{\partial u_k(t)} = g'_k(u_k(t)) = y'_k(t)$$

$$\frac{\partial u_k(t)}{\partial w_{ki}^m(t)} = y_i^{(L)(t)}$$



Derivação do algoritmo *backpropagation*- Sentido Reverso (minimização da função custo)

- Portanto o gradiente de $\partial J(t)/\partial w_{ki}^m$ é dado por

$$\frac{\partial J(t)}{\partial w_{ki}^m} = -e_k \cdot g'_k(u_k(t))y_i^L(t)$$

- A regra de aprendizagem para ajuste dos pesos na camada de saída é dada por:

$$w_{ki}^m(t+1) = w_{ki}^m(t) + \Delta w_{ki}^m(t)$$

$$w_{ki}^m(t+1) = w_{ki}^m(t) - \eta \frac{\partial J(t)}{\partial w_{ki}^m(t)}$$

$$w_{ki}^m(t+1) = w_{ki}^m(t) + \eta e_k(t)g'_k(u_k(t))y_i^L(t)$$

- Em que a constante η é a taxa de aprendizagem, ou passo de adaptação ($0 > \eta > 1$).



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na camada de saída)

- A Equação exibida anteriormente, é conhecida como Regra Delta Generalizada, a qual é derivada a partir da Regra Delta (chamada de LMS ou Regra de Widrow-Hoff).



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na camada de saída)

- A Equação exibida anteriormente, é conhecida como Regra Delta Generalizada, a qual é derivada a partir da Regra Delta (chamada de LMS ou Regra de Widrow-Hoff).
- Esta pode ser reescrita em versão matricial da seguinte forma:



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na camada de saída)

- A Equação exibida anteriormente, é conhecida como Regra Delta Generalizada, a qual é derivada a partir da Regra Delta (chamada de LMS ou Regra de Widrow-Hoff).
- Esta pode ser reescrita em versão matricial da seguinte forma:

$$\mathbf{W}^{(L+1)}(t+1) = \mathbf{W}^{(L+1)}(t) + \eta \boldsymbol{\delta}^{(L+1)}(t) \mathbf{y}^{(L)}(t)$$



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na camada de saída)

- A Equação exibida anteriormente, é conhecida como Regra Delta Generalizada, a qual é derivada a partir da Regra Delta (chamada de LMS ou Regra de Widrow-Hoff).
- Esta pode ser reescrita em versão matricial da seguinte forma:

$$\mathbf{W}^{(L+1)}(t+1) = \mathbf{W}^{(L+1)}(t) + \eta \boldsymbol{\delta}^{(L+1)}(t) \mathbf{y}^{(L)}(t)$$

- Em que δ vale:

$$\boldsymbol{\delta}^{(L+1)} = g'(\mathbf{i}^{(m)}) \circ (\mathbf{d} - \mathbf{y}^m)$$

- Nesta, \circ representa o produto de **Hadamard** (*element-wise product*).
- Em notação algorítmica o peso pode ser ajustado:

$$\mathbf{W}^{(L+1)} \Leftarrow \mathbf{W}^{(L+1)} + \eta \boldsymbol{\delta}^{(L+1)} \otimes \mathbf{y}^{(L)}$$



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na camada de saída)

- A Equação exibida anteriormente, é conhecida como Regra Delta Generalizada, a qual é derivada a partir da Regra Delta (chamada de LMS ou Regra de Widrow-Hoff).
- Esta pode ser reescrita em versão matricial da seguinte forma:

$$\mathbf{W}^{(L+1)}(t+1) = \mathbf{W}^{(L+1)}(t) + \eta \boldsymbol{\delta}^{(L+1)}(t) \mathbf{y}^{(L)}(t)$$

- Em que δ vale:

$$\boldsymbol{\delta}^{(L+1)} = g'(\mathbf{i}^{(m)}) \circ (\mathbf{d} - \mathbf{y}^m)$$

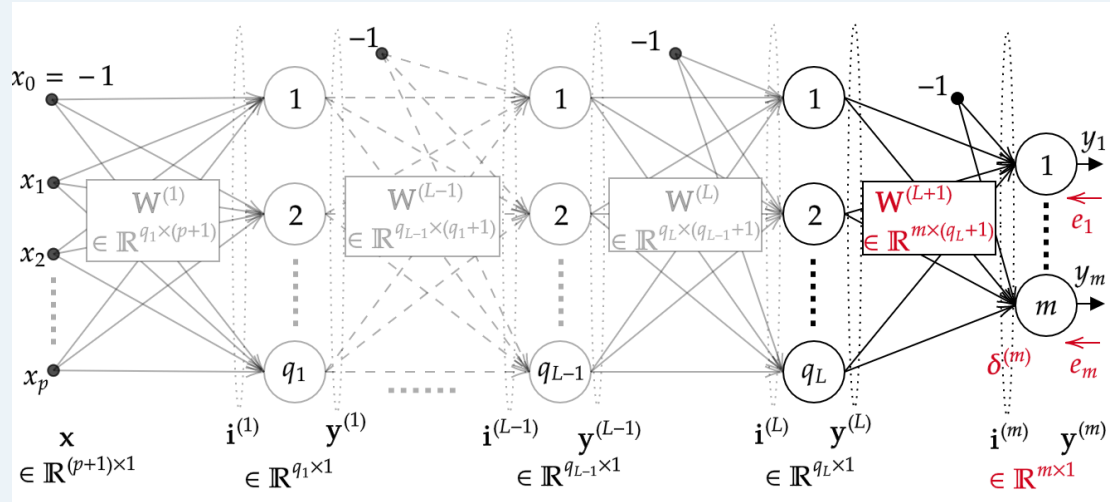
- Nesta, \circ representa o produto de **Hadamard** (*element-wise product*).
- Em notação algorítmica o peso pode ser ajustado:

$$\mathbf{W}^{(L+1)} \leftarrow \mathbf{W}^{(L+1)} + \eta \boldsymbol{\delta}^{(L+1)} \otimes \mathbf{y}^{(L)}$$

- É preciso verificar a equivalência de ordem, ao multiplicar os vetores $\boldsymbol{\delta}^{(L+1)}$ e $\mathbf{y}^{(L)}$ (obs: \otimes trata-se do **produto externo**).
- Além disto, deve-se **lembrar** que em $\mathbf{y}^{(L)}$ nesta etapa, há a presença do **bias**, portanto, $\mathbf{y}^{(L)} \in \mathbb{R}^{q_L+1}$.



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na camada de saída)





Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos camadas ocultas)

- Para o ajuste das matrizes de pesos nas camadas subsequentes, faz-se um procedimento similar ao que foi realizado na saída.



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos camadas ocultas)

- Para o ajuste das matrizes de pesos nas camadas subsequentes, faz-se um procedimento similar ao que foi realizado na saída.
- Para este caso os neurônios nas camadas ocultas não tem acesso a uma saída desejada equivalente a **d**.



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos camadas ocultas)

- Para o ajuste das matrizes de pesos nas camadas subsequentes, faz-se um procedimento similar ao que foi realizado na saída.
- Para este caso os neurônios nas camadas ocultas não tem acesso a uma saída desejada equivalente a d .
- O artifício utilizado pelos pesquisadores foi projetar uma espécie de medida de erro para os neurônios ocultos, sem que houvesse uma saída desejada.



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos camadas ocultas)

- Para o ajuste das matrizes de pesos nas camadas subsequentes, faz-se um procedimento similar ao que foi realizado na saída.
- Para este caso os neurônios nas camadas ocultas não tem acesso a uma saída desejada equivalente a d .
- O artifício utilizado pelos pesquisadores foi projetar uma espécie de medida de erro para os neurônios ocultos, sem que houvesse uma saída desejada.
- O erro neste caso, é obtido a partir dos erros dos neurônios de saída por meio de uma projeção no sentido reverso ao fluxo da rede.



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos camadas ocultas)

- Para o ajuste das matrizes de pesos nas camadas subsequentes, faz-se um procedimento similar ao que foi realizado na saída.
- Para este caso os neurônios nas camadas ocultas não tem acesso a uma saída desejada equivalente a \mathbf{d} .
- O artifício utilizado pelos pesquisadores foi projetar uma espécie de medida de erro para os neurônios ocultos, sem que houvesse uma saída desejada.
- O erro neste caso, é obtido a partir dos erros dos neurônios de saída por meio de uma projeção no sentido reverso ao fluxo da rede.
- Novamente o gradiente da função custo deve ser calculado, agora na direção de \mathbf{W}^L .



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos camadas ocultas)

- Com o auxílio da regra da cadeia, pode-se fazer:

$$\frac{\partial J(t)}{\partial w_{ij}^{(L)}} = \frac{\partial J(t)}{\partial y_i^{(L)}} \frac{\partial y_i^{(L)}}{\partial u_i^{(L)}} \frac{\partial u_i^{(L)}}{\partial w_{ij}^{(L)}}$$

$$\frac{\partial J(t)}{\partial y_i^{(L)}} = - \sum_{k=1}^m w_{ki}^m \delta_k^{(L+1)}$$

$$\frac{\partial y_i^{(L)}}{\partial u_i^{(L)}} = g'(u^{(L)})$$

$$\frac{\partial u_i^{(L)}}{\partial w_{ij}^{(L)}} = y_i^{(L-1)}$$



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos nas camadas ocultas)

- Novamente, reescrevendo a regra LMS de forma matricial, tem-se:

$$\mathbf{W}^{(L)} = \mathbf{W}^{(L)} + \eta \boldsymbol{\delta}^{(L)} \otimes \mathbf{y}^{(L-1)}$$

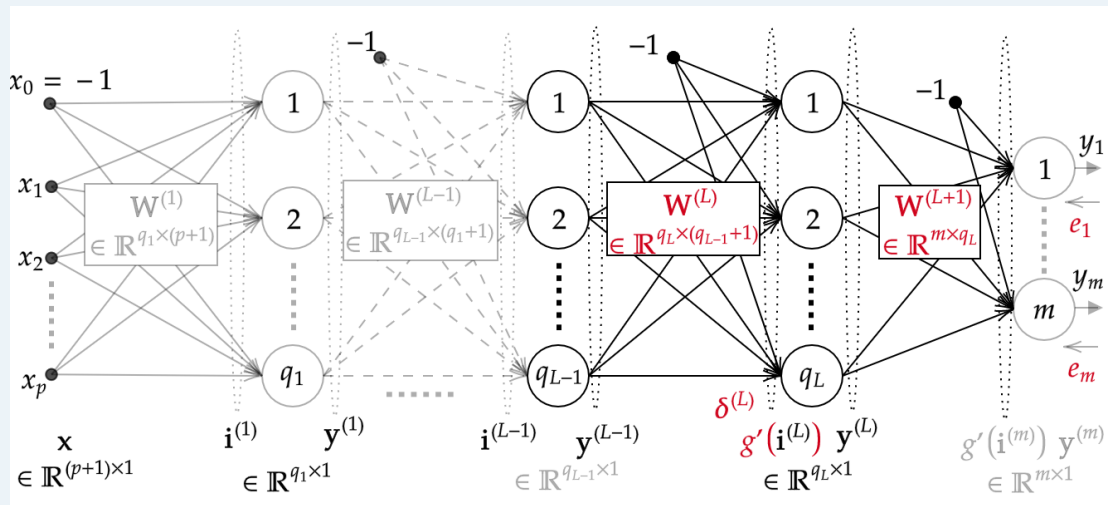
- em que $\boldsymbol{\delta}^{(L)}$ vale:

$$\boldsymbol{\delta}^{(L)} = g'(\mathbf{i}^{(L)}) \circ (\mathbf{W}_{\mathbf{b}}^{(L+1)} \boldsymbol{\delta}^{(L+1)})$$

- Em que $\boldsymbol{\delta}^{(L)}$ é definido como o gradiente local em relação ao j -ésimo neurônio da camada escondida.
- Desta maneira a matriz $\mathbf{W}_{\mathbf{b}}^{(L+1)}$ representa a matriz $\mathbf{W}^{(L+1)}$ que é **transposta** sem a influência do bias.
- Uma dica importante nesta etapa, é omitir toda a coluna referente aos pesos do *bias*.
- Da mesma maneira como foi realizado anteriormente, em $\mathbf{y}^{(L-1)}$ nesta etapa, há a presença do **bias**, portanto, $\mathbf{y}^{(L)} \in \mathbb{R}^{q_L+1}$.



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na camada de saída)





Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na primeira camada oculta)

- O processo é muito similar ao ajuste nas demais camadas escondidas, portanto se faz a aplicação novamente da regra da cadeia. Assim:

$$\frac{\partial J(t)}{\partial w_{ij}^{(1)}} = \frac{\partial J(t)}{\partial y_i^{(1)}} \frac{\partial y_i^{(1)}}{\partial u_i^{(1)}} \frac{\partial u_i^{(1)}}{\partial w_{ij}^{(1)}}$$



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na primeira camada oculta)

$$\frac{\partial u_i^1}{\partial w_{ij}^1} = \mathbf{x}$$

$$\frac{\partial y_i^{(1)}}{\partial u_i^{(1)}} = g'(u^{(1)})$$

$$\frac{\partial J(t)}{\partial y_i^{(1)}} = - \sum_{q=2}^{k=1} \sigma^2 \mathbf{w}^2$$



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos camadas ocultas)

- Novamente, reescrevendo a regra LMS uma última vez de forma matricial, tem-se:

$$\mathbf{W}^{(1)} = \mathbf{W}^{(1)} + \eta \boldsymbol{\delta}^{(1)} \otimes \mathbf{x}$$

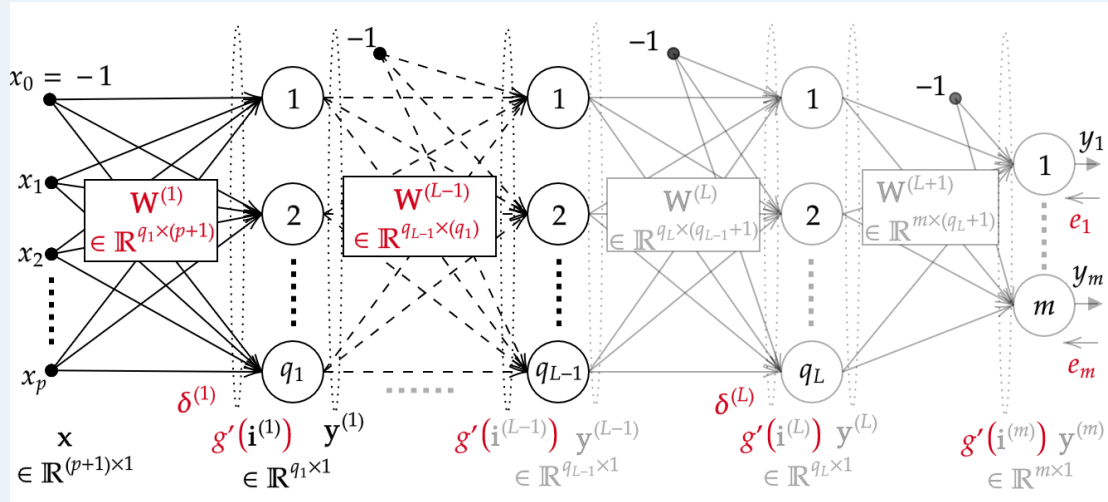
- em que $\boldsymbol{\delta}^{(1)}$ vale:

$$\boldsymbol{\delta}^{(1)} = g'(\mathbf{i}^{(1)}) \circ (\mathbf{W}_b^{(2)} \boldsymbol{\delta}^{(2)})$$

- Qual principal diferença desta etapa para as demais camadas ocultas?



Derivação do algoritmo *backpropagation*- Sentido Reverso (Atualização dos pesos na camada de saída)





MLP Algoritmos para Configuração - Treinamento - Teste.

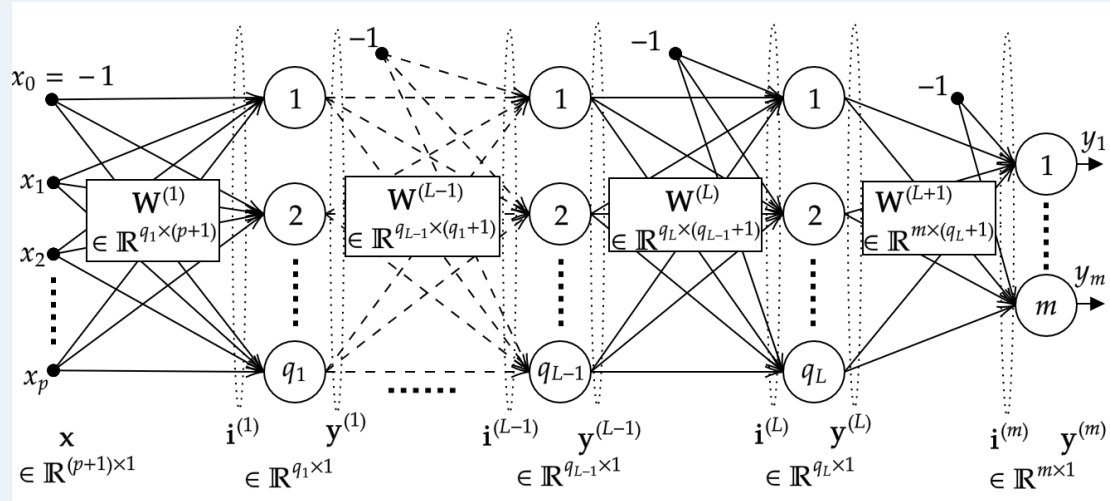
Algorithm 6: Pseudocódigo configuração da topologia da rede MLP.

- 1: Definir a quantidade L de camadas escondidas.
- 2: Definir a quantidade de neurônio em cada uma das L camadas escondidas: $[q_1, q_2, q_3, \dots, q_L]$.
- 3: Definir a quantidade de neurônios m na camada de saída.
- 4: Definir o valor da taxa de aprendizagem η .
- 5: Definir a quantidade máxima de épocas $maxEpoch$.
- 6: Definir o critério de parada em função do erro (EQM).
- 7: Criar uma lista (*list*) dos elementos: $\mathbf{W}, \mathbf{i}, \mathbf{y}, \delta$ cada uma com $L + 1$ posições.
- 8: Inicializar as $L + 1$ matrizes \mathbf{W} com valores aleatórios pequenos $(-0.5, 0.5)$.
- 9: Receber os dados de treinamento com a ordem $\mathbf{X}_{treino} \in \mathbb{R}^{p \times N}$ e os rótulos de treinamento com ordem $\mathbf{Y}_{treino} \in \mathbb{R}^{c \times N}$.
- 10: Adicionar o vetor linha de -1 na primeira linha da matriz de dados \mathbf{X}_{treino} , resultando em $\mathbf{X}_{treino} \in \mathbb{R}^{(p+1) \times N}$.

- **Obs1:** Em Python pode-se gerar números aleatórios entre 0 e 1 com a função `np.random.random_sample()`
- **Obs2:** Ao inicializar as matrizes de peso, lembrar da presença do *bias*.



Figura base para construção do algoritmo





MLP Algoritmos Para Configuração Treinamento.

Algorithm 7: Pseudocódigo para treinamento da rede MLP.

```
1: EQM  $\leftarrow$  1.  
2: Epoch  $\leftarrow$  0.  
3: while EQM > CritérioParada && Epoch < MaxEpoch do  
4:   for Cada amostra em  $\mathbf{X}_{treino}$  do  
5:      $\mathbf{x}_{amostra} \leftarrow$   $N$ -ésima amostra de  $\mathbf{X}_{treino}$ .  
6:     Forward( $\mathbf{x}_{amostra}$ )  
7:      $\mathbf{d} \leftarrow$   $N$ -ésimo rótulo de  $\mathbf{X}_{treino}$ .  
8:     BackWard( $\mathbf{x}_{amostra}$ ,  $\mathbf{d}$ ).  
9:   end for  
10:  EQM  $\leftarrow$  EQM().  
11:  Epoch  $\leftarrow$  Epoch + 1.  
12: end while
```



MLP Algoritmos Para Configuração Treinamento.

Algorithm 8: Pseudocódigo para o Forward da rede MLP.

```
1: Receber a amostra  $\mathbf{x}_{amostra} \in \mathbb{R}^{(p+1) \times 1}$ .
2:  $j \leftarrow 0$ 
3: for cada matriz de peso  $\mathbf{W}$  em cada uma das  $L + 1$  camadas. do
4:   if  $j == 0$  then
5:      $\mathbf{i}[j] \leftarrow \mathbf{W}[j] \cdot \mathbf{x}_{amostra}$ 
6:      $\mathbf{y}[j] \leftarrow g(\mathbf{i}[j])$ 
7:   else
8:      $\mathbf{y}_{bias} \leftarrow \mathbf{y}[j - 1]$  com adição de  $-1$  na primeira posição do vetor.
9:      $\mathbf{i}[j] \leftarrow \mathbf{W}[j] \cdot \mathbf{y}_{bias}$ 
10:     $\mathbf{y}[j] \leftarrow g(\mathbf{i}[j])$ 
11:   end if
12:    $j \leftarrow j + 1$ 
13: end for
```



MLP Algoritmos Para Configuração Treinamento.

Algorithm 9: Pseudocódigo para o BackWard da rede MLP.

```
1: Receber a amostra  $\mathbf{x}_{amostra} \in \mathbb{R}^{(p+1) \times 1}$  e seu rótulo  $\mathbf{d} \in \mathbb{R}^{c \times 1}$ .
2:  $j \leftarrow$  Quantidade de matrizes  $\mathbf{W} - 1$ .
3: while  $j \geq 0$  do
4:   if  $j + 1 ==$  Quantidade de matrizes  $\mathbf{W}$ , then
5:      $\delta[j] \leftarrow g'(\mathbf{i}[j]) \circ (\mathbf{d} - \mathbf{y}[j])$ .
6:      $\mathbf{y}_{bias} \leftarrow \mathbf{y}[j - 1]$  com adição de  $-1$  na primeira posição do vetor.
7:      $\mathbf{W}[j] \leftarrow \mathbf{W}[j] + \eta(\delta[j] \otimes \mathbf{y}_{bias})$ 
8:   else if  $j == 0$  then
9:      $\mathbf{W}_b[j + 1]$  Recebe a matriz  $\mathbf{W}[j + 1]$  transposta sem a coluna que multiplica pelos limiares de ativação.
10:     $\delta[j] \leftarrow g'(\mathbf{i}[j]) \circ (\mathbf{W}_b[j + 1] \cdot \delta[j + 1])$ .
11:     $\mathbf{W}[j] \leftarrow \mathbf{W}[j] + \eta(\delta[j] \otimes \mathbf{x}_{amostra})$ 
12:   else
13:     $\mathbf{W}_b[j + 1]$  Recebe a matriz  $\mathbf{W}[j + 1]$  transposta sem a coluna que multiplica pelos limiares de ativação.
14:     $\delta[j] \leftarrow g'(\mathbf{i}[j]) \circ (\mathbf{W}_b[j + 1] \cdot \delta[j + 1])$ .
15:     $\mathbf{y}_{bias} \leftarrow \mathbf{y}[j - 1]$  com adição de  $-1$  na primeira posição do vetor.
16:     $\mathbf{W}[j] \leftarrow \mathbf{W}[j] + \eta(\delta[j] \otimes \mathbf{y}_{bias})$ 
17:   end if
18:    $j \leftarrow j - 1$ 
19: end while
```




MLP Algoritmos Para Configuração Treinamento.

Algorithm 10: Pseudocódigo para o Erro Quadrático Médio.

```
1:  $EQM \leftarrow 0$ 
2: for Cada amostra em  $X_{treino}$  do
3:    $\mathbf{x}_{amostra} \leftarrow N\text{--ésima amostra de } X_{treino}.$ 
4:    $Forward(\mathbf{x}_{amostra})$ 
5:    $\mathbf{d} \leftarrow N\text{--ésimo rótulo de } X_{treino}.$ 
6:    $EQI \leftarrow 0$ 
7:    $j \leftarrow 0$ 
8:   for Cada neurônio na camada de saída do
9:      $EQI \leftarrow EQI + (d[j] - \mathbf{y}[QTD\_L - 1][j])^2$ 
10:     $j \leftarrow j + 1$ 
11:   end for
12:    $EQM \leftarrow EQM + EQI$ 
13: end for
14:  $EQM \leftarrow EQM / (2 * QtdAmostrasTreino)$ 
```



MLP etapa de teste.

Algorithm 11: Pseudocódigo para fase de operação (teste) da rede MLP.

- 1: **for** Cada amostra em \mathbf{X}_{teste} **do**
 - 2: $\mathbf{x}_{amostra} \leftarrow N$ -ésima amostra de \mathbf{X}_{teste} .
 - 3: $Forward(\mathbf{x}_{amostra})$
 - 4: Realizar a atribuição desta amostra para a classe cujo índice do vetor de saída da rede MLP possuir maior valor.
 - 5: **end for**
-



Construção de Rede Perceptron de Multicamadas

- Duas funções sigmoidais amplamente utilizadas são:



Construção de Rede Perceptron de Multicamadas

- Duas funções sigmoidais amplamente utilizadas são:

- **Sigmóide Logística:**

$$y_i = \frac{1}{1 + \exp(-u_i)} \quad y_i \in (0, 1)$$

- Sua derivada vale:

$$y'_i = \frac{dy_i}{du_i} = y_i(1 - y_i)$$

- **Tangente Hiperbólica:**

$$y_i = \frac{1 - \exp(-u_i)}{1 + \exp(-u_i)} \quad y_i \in (-1, 1)$$

- Sua derivada vale:

$$y'_i = \frac{dy_i}{du_i} = 0,5 * (1 - y_i^2)$$



Informações importantes para a construção de redes neurais.

- O projeto de uma RNA envolve a especificação de diversos itens, cujos valores influenciam consideravelmente a operação do algoritmo.
- Desta maneira, é interessante conhecer cada parâmetro do modelo, e quais valores estes podem assumir.



Dimensão do vetor de entrada (p).

- Em teoria, este item pode assumir valores entre 1 e ∞ .
- Porém, há um limite superior que depende da aplicação de interesse e do custo de se medir as variáveis presentes em \mathbf{x} .
- Deve-se ter em mente que um valor alto para p , não indica necessariamente um melhor desempenho para rede neural, pois, pode haver redundância no processo de medição.
- Algumas vezes, a quantidade p é tão elevada, que o processo se torna inviável ou extremamente custoso. Neste caso, pode-se realizar a escolha das variáveis mais relevantes para o problema.
- O caso ideal seria que cada variável $1, \dots, p$ tivesse informação exclusiva apenas para ela.
- Do ponto de vista estatístico, isto equivale dizer que as variáveis são independentes ou não-correlacionadas entre si.



Dimensão do vetor de saída (m).

- Também depende da aplicação.
- Se o interesse está em aproximação de funções, $y = F(x)$, a quantidade de neurônios na camada de saída deve refletir diretamente a quantidade de funções de saída desejadas.
- Se o interesse está em problemas de classificação de padrões, a quantidade de neurônios na camada de saída deve codificar o número de classes desejadas.
- É importante destacar que classes, referem-se aos rótulos associados ao vetor de dados. Comumente cada rótulo pode ser um valor não numérico (e.g. classe dos professores, classe dos secretários, classe dos coordenadores, etc).
- Estes devem ser convertidos para forma numérica para se treinar a rede MLP. Tal procedimento é comumente conhecido como codificação da saída da rede.
- A codificação mais comum é utilizar um único neurônio na camada de saída, e atribuir 1 para a classe 1 e atribuir 0 (ou -1) para a classe 2.



Dimensão do vetor de saída (m).

- A dimensão do vetor de saída desejado, corresponde ao número de classes do problema em questão.
- Desta maneira, pode-se definir um neurônio de saída para cada classe.
- Um exemplo prático disto, é quando se tem um problema com três classes, logo, existirão três neurônios de saída.
- Como um vetor de entrada, não pertencerá a mais de uma classe ao mesmo tempo, o vetor de saída desejada valor 1 na componente correspondente à classe deste vetor, e 0 (ou -1) para outras componentes.
- Por exemplo, seja o vetor de entrada $\mathbf{x} \in \mathbb{R}^{p \times 1}$ pertence à segunda classe, então seu vetor de saída $\mathbf{d} = [0 \ 1 \ 0]^T$



Número de neurônios na(s) camada(s) escondida(s) (q_1, q_2, \dots, q_L)

- Encontrar o número ideal de neurônios da camada escondida **não** é uma tarefa fácil, pois, depende de diversos fatores, dos quais não temos total controle. Pode-se destacar fatores importantes como:
 - 1 Quantidade de dados disponíveis para treinar a rede.
 - 2 Qualidade dos dados disponíveis.
 - 3 Número de parâmetros ajustáveis (pesos e limiares da rede).
 - 4 Complexidade do problema.
 - 5 Além disto, pode-se imaginar que este é um problema apenas para uma camada.
- O valor de q é geralmente encontrado por tentativa e erro, em função da capacidade de generalização da topologia definida.
- Um valor ideal, é aquele permite com que o modelo tenha desempenho adequado tanto para os dados de treinamento como para os "desconhecidos".
- Existem algumas fórmulas que foram criadas para tentar resolver este problema em particular da quantidade de neurônios na(s) camada(s) escondida(s).



Número de neurônios na(s) camada(s) escondida(s) (q_1, q_2, \dots, q_L)

- Deve-se ter em mente que estas regras devem ser usadas apenas para dar um valor inicial para q .
- Desta maneira, deve-se treinar e testar várias vezes a rede MLP com diversas topologias, de modo a certificar que a rede generaliza bem para novos dados.
- Pode-se destacar três regras amplamente utilizadas:
 - 1 **Regra do valor médio:** O número de neurônios na camada escondida é igual ao valor médio do número de entradas e o número de saída da rede:

$$q = \frac{p + m}{2}$$



Número de neurônios na(s) camada(s) escondida(s) (q_1, q_2, \dots, q_L)

- Pode-se destacar três regras amplamente utilizadas:

① **Regra da raiz quadrada:** O número de neurônios na camada escondida é igual a raiz quadrada do produto do número de entradas pelo número de saídas da rede:

$$q = \sqrt{p * m}$$

② **Regra Kolmogorov:** O número de neurônios na camada escondida é igual a duas vezes o número de entrada da rede adicionado de 1:

$$q = 2p + 1$$



Número de neurônios na(s) camada(s) escondida(s) (q_1, q_2, \dots, q_L)

- As equações exibidas anteriormente apenas consideram características da rede, e desprezam informações úteis, tais como números de dados disponíveis para treinar e testar a rede.
- Uma regra que define um valor inferior para q levando em consideração o número de dados de treinamento é dada por:

$$q \geq \frac{N - 1}{p + 2}$$

- Entretanto, deve-se ter em mente a seguinte regra geral: É necessário se ter muito mais dados do que parâmetros ajustáveis. Logo, $N \gg Z$



Número de neurônios na(s) camada(s) escondida(s) (q_1, q_2, \dots, q_L)

- Uma maneira refinada de expressar a última equação, pode ser realizada de acordo com a equação

$$q \approx \left(\frac{\varepsilon N - m}{p + m + 1} \right),$$

em que ε é o erro percentual máximo aceitável durante a fase de teste (operação) da rede.

- Esta equação é bastante completa, visto que considera não só apenas a estrutura da rede, como também o erro máximo tolerado para teste e o número de dados disponíveis.



Funções de ativação

- Em teoria, cada neurônio pode ter uma função de ativação distinta dos demais.
- Entretanto, para simplificar o projeto da rede, é comum adotar a mesma função para todos os neurônios.
- Em geral, utiliza-se como escolha a função logística ou a tangente hiperbólica.
- Aquela que for escolhida para os neurônios nas camadas escondidas, será adotada para os neurônios na camada de saída também.
- Em algumas aplicações, é comum escolher uma função de ativação linear para os neurônios na camada de saída, ou seja, $g(\mathbf{u}) = C \cdot \mathbf{u}$, em que C é uma constante positiva.
- Neste caso, $g'(\mathbf{u}) = C$.
- É reforçado ainda que esta função ser linear não altera o poder computacional da rede, porém, deve-se lembrar que os neurônios na camada escondida devem ter uma função de ativação não-linear.



Critério de parada e Convergência.

- A convergência da rede MLP é, em geral, avaliada com base nos valores do EQM por época de treinamento.
- Em geral, o treinamento da rede é interrompido quando EQM_{epoca} atinge um limite inferior adequado para o problema, por exemplo, $EQM \leq 0,001$ ou quando o número máximo de épocas for atingido.
- Porém, uma outra análise também pode ser realizada quando o problema é de classificação: pela acurácia na classificação por época

$$Acurácia_{epoca} = \frac{\text{Número de vetores classificados corretamente}}{N_{\text{mero total de vetores}}}.$$

- O gráfico que pode ser construído a partir destas medidas, $EQM_{epoca} \times \text{número de épocas}$ ou $Acurácia_{epoca} \times \text{número de épocas}$ é chamado de curva de aprendizagem da rede neural.



Avaliação da rede treinada

- Para validar a rede treinada, é importante testar sua resposta para dados de entrada diferentes daqueles utilizados na etapa de treinamento.
- Entretanto, nem sempre é possível realizar novas medições, e portanto, faz-se necessário treinar a rede com apenas uma parte dos dados selecionados aleatoriamente.
- Desta maneira, os dados são divididos em treinamento, com tamanho $N_1 < N$ e de teste, com tamanho $N_2 = N - N_1$.
- Em geral, escolhe-se N_1 para que a razão N_1/N esteja na faixa 0,75 a 0,90.
- O valor EQM calculado com os dados de teste é chamado de **erro de generalização** da rede, pois testa a capacidade da mesma em "extrapolar" o conhecimento aprendido durante o treinamento.
- Destaca-se ainda que geralmente o erro de generalização é superior ao erro de treinamento, pois, trata-se de um novo conjunto de dados.



Dicas para um bom projeto da rede MLP

- Pré-processamento dos dados:

- 1 Antes de apresentar os dados para a rede é comum mudar a escala original pelos métodos já discutidos anteriormente.
- 2 Entretanto, essa normalização dos dados pode ser escolhida a partir da definição das funções de ativação nos neurônios da rede.
- 3 Para a escolha da função logística, aplica-se a seguinte transformação em cada uma das p componentes do vetor:

$$x_j^* = \frac{x_j - \min(x_j)}{\max(\mathbf{X}) - \min(\mathbf{X})}$$

- 4 E para o caso do uso da função tangente hiperbólica, aplica-se:

$$x_j^* = 2 \cdot \left(\frac{x_j - \min(\mathbf{X})}{\max(\mathbf{X}) - \min(\mathbf{X})} \right) - 1$$



Dicas para um bom projeto da rede MLP

- Taxa de aprendizagem variável:

- 1 É interessante utilizar uma taxa de aprendizagem variável ao longo do tempo, $\eta(t)$, decaindo até um valor baixo com o passar das iterações.

$$\eta(t) = \eta_0 \left(1 - \frac{1}{t_{max}} \right), \text{Decaimento linear}$$

$$\eta(t) = \frac{\eta_0}{1 + t}, \text{Decaimento exponencial.}$$

Nestas equações, η_0 é o valor inicial da taxa de aprendizagem, e t_{max} é o número máximo de **apresentações** dado por:

$$t_{max} = N_1 \cdot \text{Número máximo de épocas}$$

- 2 É interessante inicializar η com um valor alto ($\eta_0 < 0, 1$), e terminar com um valor bem baixo, na ordem de $\eta \approx 0, 0001$, de modo a estabilizar o processo de aprendizado.



Dicas para um bom projeto da rede MLP

- **Termo de momento:**

- 1 Na equação de ajuste dos pesos da rede, é interessante adicionar um termo de momento, que possui objetivo em tornar o processo de modificação dos pesos mais estável.

$$\mathbf{W}^{(L+1)}(t+1) = \mathbf{W}^{(L+1)}(t) + \eta \delta^{(L+1)} \otimes \mathbf{y}^{(L)} + \alpha [\mathbf{W}^{(L+1)}(t) - \mathbf{W}^{(L+1)}(t-1)]$$

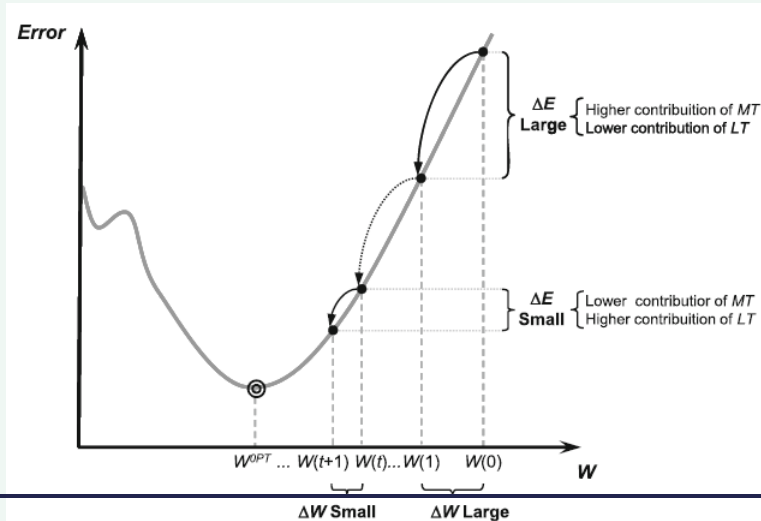
$$\mathbf{W}^{(L)}(t+1) = \mathbf{W}^{(L)}(t) + \eta \delta^{(L)} \otimes \mathbf{y}^{(L-1)} + \alpha [\mathbf{W}^{(L)}(t) - \mathbf{W}^{(L)}(t-1)]$$

Em que α é a constante chamada fator de momento e é mantido na faixa 0 a 0,9.



Dicas para um bom projeto da rede MLP

- Termo de momento:





Dicas para um bom projeto da rede MLP

- **Generalização:**

- ① A rede MLP é um dos algoritmos de aproximação mais poderosos que existem, porém, todo este poder computacional, se não for utilizado corretamente, não necessariamente implica em uma rede que seja capaz de generalizar corretamente.
- ② A definição desta generalização adequada é a habilidade da rede em utilizar o conhecimento armazenado nos seus pesos e limiares para gerar saídas coerentes para novos vetores de entrada.
- ③ Uma generalização é considerada boa quando a rede foi capaz de aprender a relação entrada-saída do mapeamento de interesse. O bom treinamento da rede, depende de vários fatores: parâmetros ajustáveis e todos os hiperparâmetros a serem definidos.



Dicas para um bom projeto da rede MLP

- **Generalização:**

- 1 Com relação aos parâmetros ajustáveis, um dos principais motivos de um treinamento inadequado, é pelo fato de um possível subdimensionamento ou sobredimensionamento da rede MLP.
- 2 Neste caso, pode haver a ocorrência de um **underfitting** ou **overfitting** da rede. Em ambos casos, a capacidade de generalização é ruim.
- 3 O *underfitting* ocorre quando a rede não tem poder computacional suficiente para aprender o mapeamento de interesse.
- 4 O *overfitting* ocorre quando a rede possui neurônios ocultos demais (incluindo camadas ocultas), e memoriza os dados de treinamento.
- 5 O ajuste ideal é obtido para um número de camadas ocultas e neurônios nestas camadas que confere à rede um bom desempenho durante a fase de teste, ou seja, uma boa generalização.



Dicas para um bom projeto da rede MLP

- **Generalização:**

- 1 Uma das técnicas utilizadas para treinar a rede MLP, de modo a garantir uma generalização é conhecida como parada prematura (*early stopping*).
- 2 Neste método, o conjunto de treinamento também é dividido em duas partes, uma para estimação dos parâmetros da rede e outra para validação durante o treinamento.
- 3 Este conjunto de validação, deve ser usado de "tempos em tempos", por exemplo, a cada 5 épocas de treinamento. Quando isto ocorrer, deve-se medir o EQM de validação.
- 4 A ideia do *early stopping*, é interromper o treinamento quando o EQM de validação assumir uma tendência de crescimento.
- 5 Essa tendência é um indicativo que a rede está começando a se especializar demais nos dados de estimação.



Dicas para um bom projeto da rede MLP

- **Generalização:**





Rede de Funções de Base Radial - RBF

- *Radial Basis Function* (RBF).
- RBF são funções em que seus valores de mapeamento são dependentes da distância:
 - 1 Com relação à distância.
 - 2 Com relação a um ponto **c**, normalmente chamada de **centro**.
- A arquitetura da rede RBF é utilizada para as mesmas aplicações da rede MLP padrão:
 - 1 Aproximação universal de função.
 - 2 Classificação de padrões.
- Entretanto, diferentemente da rede MLP, a rede RBF apresenta apenas uma única camada oculta, além da camada de saída.
- Os neurônios na camada oculta (funções de base radial), possuem funções de ativação não lineares diferentes das estudadas na rede MLP.
- Já os neurônios na camada de saída possuem, em geral, saída linear.



Rede de Funções de Base Radial - RBF

- Para uso da rede RBF, é necessário ter um número finito de N exemplos de treinamento na forma (\mathbf{x}, \mathbf{d})
- Assume-se que estes vetores são relacionado segundo uma lei matemática $\mathbf{F}(\cdot)$, tal que:
 $\mathbf{d}(t) = \mathbf{F}[\mathbf{x}(t)]$. Em que $t = 1, 2, \dots, N$.
- Uma maneira de se adquirir conhecimento sobre $\mathbf{F}(\cdot)$ é através de dados disponíveis.
- Pode-se então utilizar a rede RBF para gerar uma aproximação de $\mathbf{F}(\cdot)$, denotada por $\hat{\mathbf{F}}(\cdot)$, tal que

$$\hat{\mathbf{y}}(t) = \hat{\mathbf{F}}[\mathbf{x}(t)]$$

- Em que $\hat{\mathbf{y}}$ é a saída gerada pela rede.
- Espera-se que esta saída seja próxima da saída real $\mathbf{d}(t)$



Rede de Funções de Base Radial - RBF

- Cada vetor de entrada é representado como:

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_p(t) \end{bmatrix}$$

- Enquanto o vetor de saída associado ao vetor de entrada é representado por:

$$\mathbf{d}(t) = \begin{bmatrix} d_1(t) \\ \vdots \\ d_m(t) \end{bmatrix}$$

- Define-se ainda x_j como uma componente qualquer do vetor de entrada \mathbf{x} , e d_k uma componente qualquer do vetor de saída desejada \mathbf{d} .



Rede de Funções de Base Radial - RBF

- O vetor de pesos de cada função de base radial, também chamado de **centro** da função de base, é representado como:

$$\mathbf{c}_i = \begin{bmatrix} c_{i1} \\ \vdots \\ c_{ij} \\ \vdots \\ c_{ip} \end{bmatrix}$$

- Em que c_{ij} é o peso que conecta a j -ésima entrada à i -ésima função de base.
- Assim como na rede MLP, as funções de base não têm acesso direto à saída da rede RBF.



Rede de Funções de Base Radial - RBF

- De modo semelhante, o vetor de pesos associado ao k -ésimo neurônio da camada de saída é representado como:

$$\mathbf{w}_k = \begin{bmatrix} w_{k0} \\ \vdots \\ w_{kq} \end{bmatrix} = \begin{bmatrix} \theta_k \\ \vdots \\ w_{kq} \end{bmatrix}$$

- Em que θ_k é o limiar associado ao neurônio de saída k .
- q é o número de funções de base radial (valor igual à quantidade de neurônios na camada oculta).

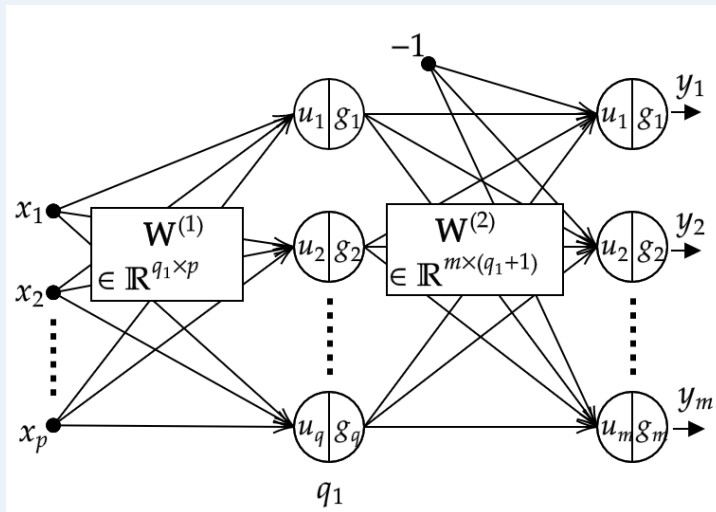


RBF - Projeto da camada oculta

- Esta etapa envolve a especificação:
 - 1 Do número de funções de base.
 - 2 Determinação dos seus parâmetros
 - 3 Determinação dos pesos dos neurônios de saída.
- Este material, baseia-se na construção a rede RBF com base em: J. E. Moody and C. Darken. Fast learning in networks of locally-tuned processing units. Neural Computation, 1(1):281–294, 1989.
- Estes autores separam o treinamento da rede RBF em três etapas executadas em sequência.
- Durante a primeira etapa, usa-se um algoritmo de formação de agrupamentos para encontrar os chamados **centros** das funções de base.
- A segunda etapa trata do uso de métodos heurísticos para determinar o raio ou abertura de cada função de base.
- Por último, uma vez determinados os centros e os raios, pode-se computar os pesos dos neurônios da camada de saída através de um algoritmo de aprendizagem supervisionado.
- O processo de treinamento, inicia-se pela camada intermediária e é encerrado na saída.



RBF - Projeto da camada oculta





RBF - Projeto da camada oculta

- Assim, após a apresentação de um vetor de entrada \mathbf{x} na iteração t , calcula-se a ativação da i -ésima função de ativação por meio de:

$$u_i(t) = \|\mathbf{x}(t) - \mathbf{c}_i(t)\|, \quad i = 1, \dots, q$$

- Em que q é o número de funções de base desta camada, e o vetor \mathbf{c}_i , mantido constante para o neurônio i , define o centro da i -ésima função de base.
- A saída da i -ésima função de base é calculada por:

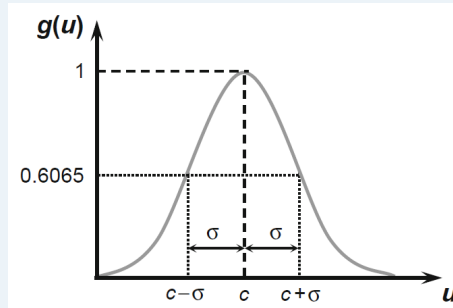
$$y_i(t) = g(u_i(t)) = \exp \left\{ -\frac{u_i^2(t)}{2\sigma_i^2} \right\}, \quad i = 1, \dots, q$$

- Em que σ_i é o chamado raio da i -ésima função de base, pois define a largura (abertura) da função de ativação gaussiana deste neurônio.



RBF - Projeto da camada oculta

- Discussões sobre a gaussiana

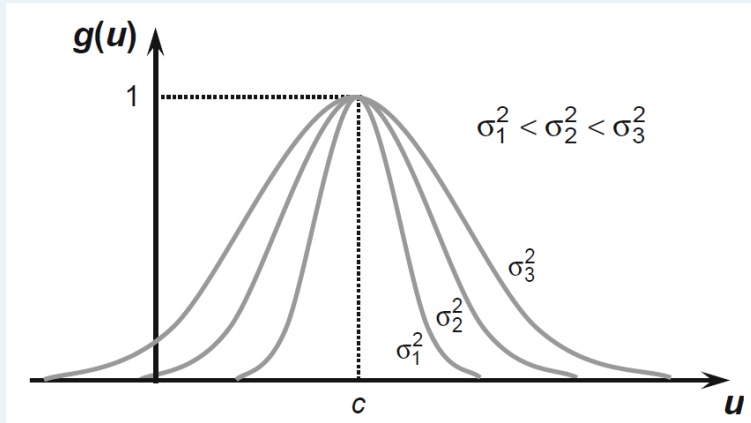


- Como já discutido, c é o centro da função de base e σ^2 denota a variância, a qual indica o quão disperso está o potencial de ativação u



RBF - Projeto da camada oculta

- Discussões sobre a gaussiana





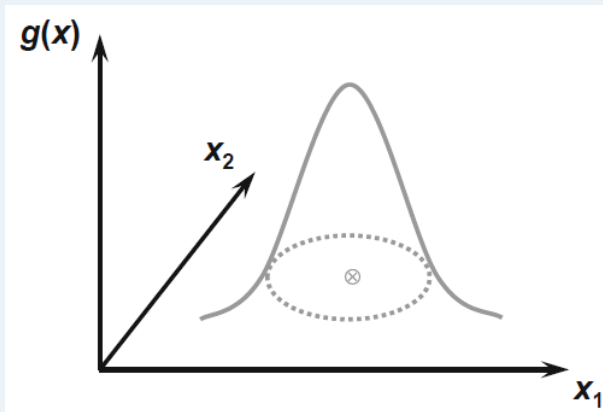
RBF - Projeto da camada oculta

- O que acontece quando x está próximo ao vetor c ?



RBF - Projeto da camada oculta

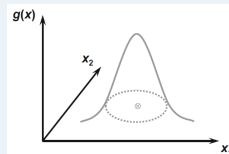
- O que acontece quando x está próximo ao vetor c ?





RBF - Projeto da camada oculta

- Quando o vetor de entrada está próximo de seu **centro** \mathbf{c}_i , o neurônio i fornece resposta máxima ($y_i \approx 1$).
- Desta maneira, diz-se que cada neurônio da camada escondida tem seu próprio **campo receptivo** no espaço de entrada, que é uma região centrada em \mathbf{c}_i com tamanho proporcional a σ_i .
- Em outras palavras, o neurônio produzirá respostas similares para todos aqueles padrões que estejam a uma mesma distância radial do centro da gaussiana.





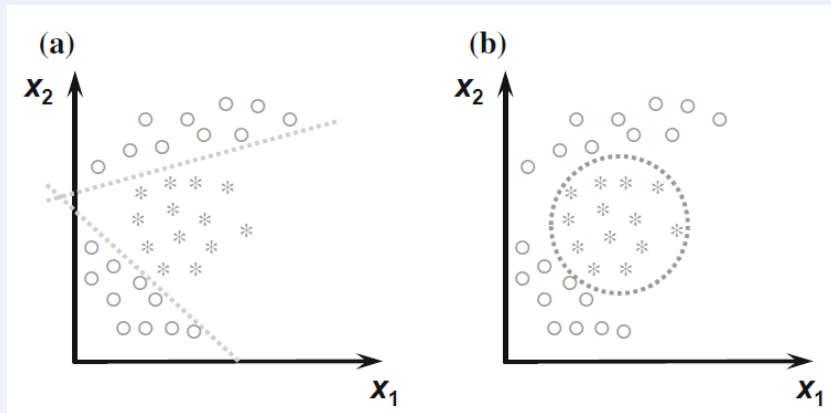
RBF - Projeto da camada oculta

- Um comparativo com a rede MLP pode ser realizado, pois, este computa fronteiras de decisão das classes por intermédio de uma combinação de hiperplanos.
- Na rede RBF, com funções de ativação gaussiana, as fronteiras delimitadoras são definidas por **campos receptivos** hiperesféricos.
- Com isto, a classificação dos padrões considerará a distância radial em relação ao centro das hiperesféras.



RBF - Projeto da camada oculta

- Diferença MLP e RBF.





RBF - Projeto da camada oculta

- Assim, o principal objetivo do treinamento dos neurônios da camada intermediária, consiste em posicionar os centros de suas gaussianas de forma apropriada.
- Existem duas maneiras amplamente utilizadas de se determinar os centros das funções de base.
- A primeira de forma aleatória, e a segunda através de um método não supervisionado. Tais métodos são dependentes somente das características dos dados.



RBF - Seleção aleatória de **centros**

- Este método consiste em selecionar aleatoriamente um certo número q de entradas para atuarem como centros da rede RBF. Matematicamente pode-se escrever:

$$\mathbf{c}_i = \mathbf{x}_k, \quad i = 1, \dots, q$$

- Em que k é um número inteiro escolhido aleatoriamente, **sem reposição** entre 1 e N .
- Para cada i deve-se escolher um k diferente para não haver repetição de centros.



RBF - Seleção por quantização vetorial

- O segundo método envolve a determinação dos centros através de um algoritmo de **quantização vetorial**, sejam eles de origem não-neural como K -médias, ou de origem neural, tais como rede SOM, neural-GAS ou WTA.
- Desta maneira, descreve-se um dos algoritmos mais simples de formação de agrupamento, popularmente conhecido por WTA (*Winner-Take-All*).
- Sua formulação foi proposta por Kohonen em: T. K. Kohonen. Self-Organizing Maps. Springer-Verlag, Berlin, Heidelberg, 2nd extended edition, 1997.



RBF - Seleção por quantização vetorial

Algorithm 12: Pseudocódigo para o Erro Quadrático Médio.

- 1: Definir o número de funções de base radial (q)
- 2: Definir a taxa de aprendizagem η ($0 < \eta < 1$)
- 3: Iniciar os **centros** \mathbf{c}_i , $i = 1, \dots, q$ com valores aleatórios
- 4: Fazer $t=0$
- 5: **while** $t < N$ **do**
- 6: Selecionar aleatoriamente (sem reposição) o vetor de entrada $\mathbf{x}(t)$.
- 7: Determinar o índice do centro mais próximo de $\mathbf{x}(t)$, ou seja,

$$i^* = \arg \min_{\forall i} \|\mathbf{x}(t) - \mathbf{c}_i(t)\|$$

- 8: Aplicar a seguinte regra não supervisionada:

$$\mathbf{c}_i(t+1) = \begin{cases} \mathbf{c}_i(t) + \eta[\mathbf{x}(t) - \mathbf{c}_i(t)] & \text{se } i = i^*(t) \\ \mathbf{c}_i(t) & \text{caso contrário} \end{cases}$$

- 9: Fazer $t = t + 1$
- 10: **end while**
- 11: Verificar se critério de parada foi atingido, caso contrário retorne ao passo 4.



RBF - Seleção por quantização vetorial

- Alguns comentários importantes sobre a regra de aprendizagem da rede WTA.
 - 1 A cada iteração só atualiza o centro que está mais próximo do vetor de entrada atual.
 - 2 Interrompe-se o treinamento da rede quando:
 - A variação do erro de quantização em função da época de treinamento for pequena.
 - Quando o número máximo de épocas for atingido.
- Este erro é calculado por:

$$E_q = \frac{\sum_{t=0}^{N-1} \|\mathbf{x}(t) - \mathbf{c}_{i^*}(t)\|^2}{N}$$

em que $\mathbf{c}_{i^*}(t)$ é o centroide mais próximo de $\mathbf{x}(t)$



RBF - Determinação do raio das funções de base

- Uma vez que os centros das funções de base tenham sido determinados, o passo seguinte é definir os raios (σ_i) das várias funções de base.
- Este parâmetro é de fundamental importância para o projeto da rede RBF.
- Se ele for muito alto, existe um elevado grau de superposição entre campos receptivos das funções de base. Isto pode levar a uma baixa precisão, ou seja, a rede generaliza demais.
- Se σ for muito pequeno a superposição deixa de existir, porém a precisão é elevada apenas para os casos em que $\mathbf{x}(t) \approx \mathbf{c}_i$. Neste caso a rede não generaliza bem.



RBF - Determinação do raio das funções de base

- Existem diversas técnicas para determinar σ_i , contudo, algumas amplamente utilizadas são:
 - Um único raio utilizado por todos os neurônios, ou seja, $\sigma_i = \sigma$. Isso significa que todas as funções de base terão a mesma abertura. Neste caso, uma estratégia comum consiste em fazer σ igual a uma fração da maior distância entre os centros de todos os neurônios:

$$\sigma = \frac{d_{\max}(\mathbf{c}_i, \mathbf{c}_j)}{\sqrt{2q}}, \quad \forall i \neq j$$

Em que $d_{\max}(\mathbf{c}_i, \mathbf{c}_j) = \max_{\forall i \neq j} \{\|\mathbf{c}_i - \mathbf{c}_j\|\}$.



RBF - Determinação do raio das funções de base

- Existem diversas técnicas para determinar σ_i , contudo, algumas amplamente utilizadas são:
 - Um único raio utilizado por todos os neurônios, ou seja, $\sigma_i = \sigma$. Isso significa que todas as funções de base terão a mesma abertura. Neste caso, uma estratégia comum consiste em fazer σ igual a uma fração da maior distância entre os centros de todos os neurônios:

$$\sigma = \frac{d_{\max}(\mathbf{c}_i, \mathbf{c}_j)}{\sqrt{2q}}, \quad \forall i \neq j$$

Em que $d_{\max}(\mathbf{c}_i, \mathbf{c}_j) = \max_{\forall i \neq j} \{\|\mathbf{c}_i - \mathbf{c}_j\|\}$.

- Cada neurônio usa seu próprio valor de raio, que tem seu valor definido como metade da menor distância do neurônio i e o centro mais próximo.

$$\sigma_i = \frac{d_{\min}(\mathbf{c}_i, \mathbf{c}_j)}{2}, \quad \forall i \neq j$$

Em que $d_{\min}(\mathbf{c}_i, \mathbf{c}_j) = \min_{\forall i \neq j} \{\|\mathbf{c}_i - \mathbf{c}_j\|\}$.



RBF - Determinação do raio das funções de base

- Existem diversas técnicas para determinar σ_i , contudo, algumas amplamente utilizadas são:
 - Uma variante do caso 2. O raio da i -ésima função de base é a distância média do centro desta base para os K ($1 \leq K \ll q$) centros mais próximos:

$$\sigma_i = \frac{\sum_{k=1}^K d(\mathbf{c}_i, \mathbf{c}_{i_k})}{K},$$

em que i_k é o índice do k -ésimo centro mais próximo de \mathbf{c}_i .



RBF - Projeto da camada de saída

- Após a determinação dos centros e raios das funções de base da camada intermediária, a última etapa consiste em estimar a matriz de pesos.
- Nesta, pode-se utilizar uma regra de aprendizagem como por exemplo as utilizadas nos algoritmos do perceptron simples, LMS ou MLP.
- Entretanto, será exibido a estimação desta matriz através do método dos mínimos quadrados (MQO/OLS/LMQ).
- Durante esta terceira etapa, os centros e raios calculados nas duas etapas anteriores não tem valores alterados.



RBF - Projeto da camada de saída (MQ)

- Para aplicar tal método, que estima a matriz de pesos na camada de saída, deve-se acumular todas as saídas dos neurônios **ocultos** em uma matriz \mathbf{Z} . A ideia é que cada vetor de entrada $\mathbf{x}(t) \in \mathbb{R}^p$ seja transformado em um vetor de saída dos neurônios **ocultos** $\mathbf{z}(t) \in \mathbb{R}^q$.
- Portanto, após a aplicação do vetor de entrada na camada oculta, tem-se q saídas que podem ser representadas por um único vetor $\mathbf{z}(t) \in \mathbb{R}^{q+1}$.

$$\mathbf{z}(t) = \begin{bmatrix} z_0(t) \\ z_1(t) \\ \vdots \\ z_q(t) \end{bmatrix} = \begin{bmatrix} +1 \\ z_1(t) \\ \vdots \\ z_q(t) \end{bmatrix}$$

em que foi adicionado um valor constante igual a $z_0(t) = +1$.



RBF - Projeto da camada de saída (MQ)

- Para cada vetor de entrada $\mathbf{x}(t) \in \mathbb{R}^p, t = 1, \dots, N$, tem-se um vetor $\mathbf{z}(t) \in \mathbb{R}^{q+1}$ correspondente, que deve ser organizado como uma coluna de uma matriz \mathbf{Z} . Esta matriz organizada é $\mathbf{Z} \in \mathbb{R}^{[q+1] \times N}$:

$$\mathbf{Z} = [\mathbf{z}(1) | \mathbf{z}(2) | \dots | \mathbf{z}(N)]$$

- Sabe-se que para cada vetor de entradas, tem-se um vetor de saídas desejadas $\mathbf{d}(t)$ correspondente. Pode-se organizar todos os N vetores desejados ao longo das colunas da matriz $\mathbf{D} \in \mathbb{R}^{m \times N}$:

$$\mathbf{D} = [\mathbf{d}(1) | \mathbf{d}(2) | \dots | \mathbf{d}(N)]$$



RBF - Projeto da camada de saída (MQ)

- Pode-se entender o cálculo dos pesos na camada de saída como o cálculo dos parâmetros de um mapeamento linear entre a camada oculta e a de saída.
- O papel do vetor de "entrada" para a camada de saída na iteração t é desempenhado pelo vetor $\mathbf{z}(t)$, enquanto o vetor de saída é representado por $\mathbf{d}(t)$.
- Assim, busca-se estimar uma matriz \mathbf{W} que melhor represente o mapeamento:

$$\mathbf{d}(t) = \mathbf{W}\mathbf{z}(t)$$

Ou em sua versão matricial:

$$\mathbf{D} = \mathbf{W}\mathbf{Z}$$



RBF - Projeto da camada de saída (MQ)

- Desta maneira, podemos utilizar o método dos mínimos quadrados já discutido nas notas de aula sobre os fundamentos da regressão linear e projeto do classificador linear de mínimos quadrados.
- Assim, usando as matrizes \mathbf{Z} e \mathbf{D} , a matriz \mathbf{W} pode ser estimada através de:



RBF - Projeto da camada de saída (MQ)

- Desta maneira, podemos utilizar o método dos mínimos quadrados já discutido nas notas de aula sobre os fundamentos da regressão linear e projeto do classificador linear de mínimos quadrados.
- Assim, usando as matrizes \mathbf{Z} e \mathbf{D} , a matriz \mathbf{W} pode ser estimada através de:

$$\mathbf{W} = \mathbf{D}\mathbf{Z}^T(\mathbf{Z}\mathbf{Z}^T)^{-1}$$

- É importante realizar alguns comentários sobre a matriz \mathbf{W} :
- É importante ressaltar que para satisfazer a equação $\mathbf{d}(\mathbf{t}) = \mathbf{W}\mathbf{z}(t)$ é necessário que $\mathbf{W} \in \mathbb{R}^{m \times [q+1]}$
- A k -ésima linha da matriz \mathbf{W} , corresponde ao vetor de pesos do k -ésimo neurônio de saída.



RBF - Considerações finais

- Por se tratar de uma rede com aprendizado supervisionado, a rede RBF é aplicada aos mesmos problemas da rede MLP.
- A rede RBF pode ser entendida como um modelo matemático que realiza aproximação de funções através da combinação linear de funções de base gaussiana.
- Para ser usada como classificador de padrões, a codificação dos vetores de saída desejadas segue a mesma lógica daquela utilizada para a rede MLP. Ou seja, o número de neurônios na camada de saída é igual ao número de classes, exceto para o caso em que se tem apenas duas classes. Para este caso, necessita-se apenas de um neurônio de saída.
- Durante o treinamento, a saída desejada do neurônio associado à classe do vetor de entrada atual deve ser igual a 1, enquanto as saídas desejadas para outros neurônios deverão ser iguais a 0 (ou -1).
- Durante a fase de teste, o vetor de entrada (cuja classe é desconhecida), será associado à classe representada pelo neurônio que gerar maior valor para $y_k(t)$.