

Redes Neurais

Multilayers Perceptron

MLP

Perceptrons de multcamadas

Esse algoritmo veio para sanar as limitações do perceptron, que somente conseguir classificar dados linearmente separáveis

Assim, este algoritmo consegue classificar dados que não são linearmente separáveis, assim cada neurônio vai incluir uma função de ativação, contendo um ou mais camadas de entradas.

Deficiências: Análise teórica é mais complexo

Dificuldade de visualização

Aprendizado mais difícil, se há um espaço maior para as funções, ou seja, um espaço maior de busca

MLP

Como treinar? BackPropagation

Fase DE IDA: Quando se tem os pesos fixos, e se propaga o sinal camada a camada até a saída

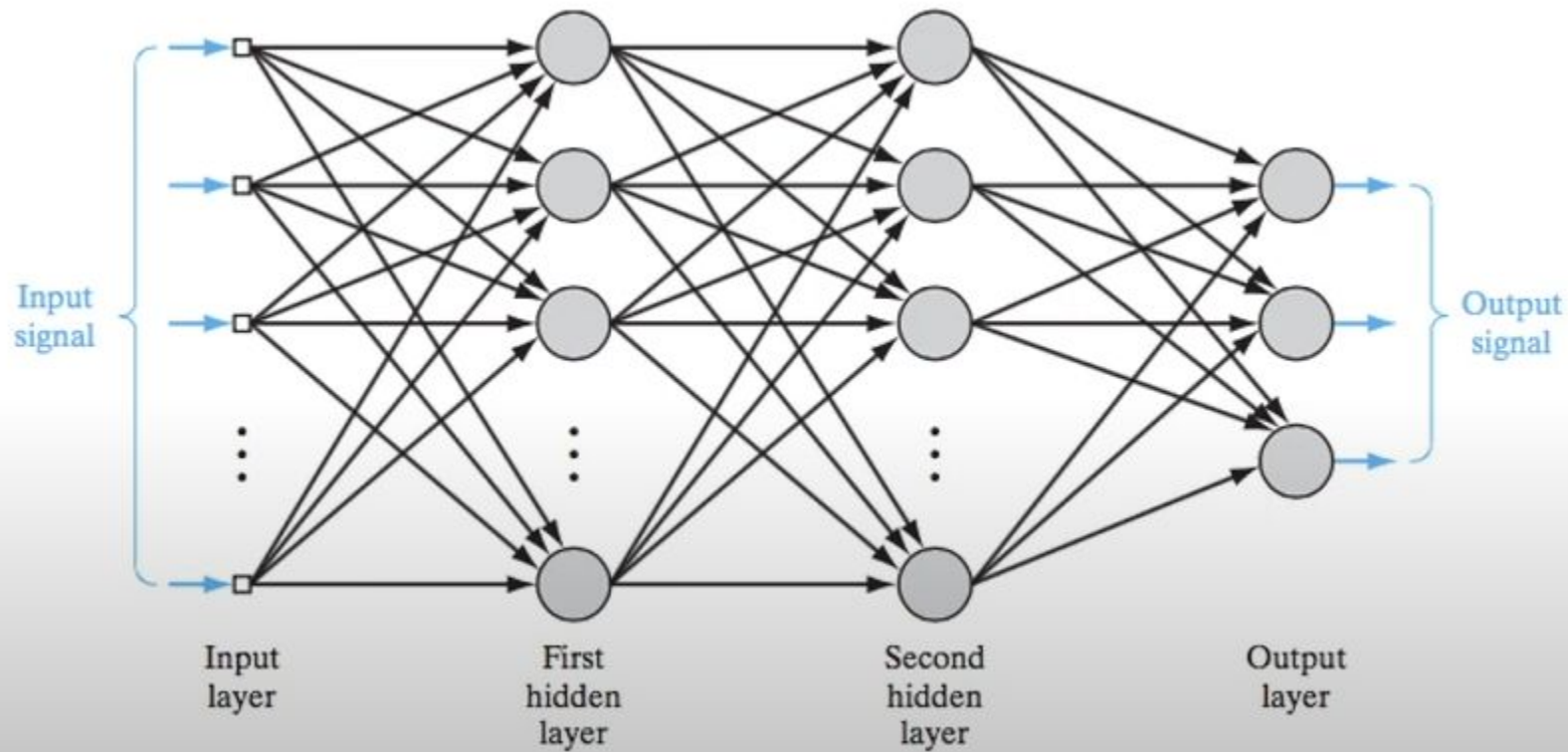
As mudanças ocorrem somente na ativação das potências, e nas saída dos neurônio, ou seja, não contém ajuste.

Fase de volta: O erro é produzido, comparando a saída desejada com a obtida, esse erro é retro propagado pela rede camada a camada, e assim esses pesos sinápticos são ajustados.

O termo back-propagation se tornou popular após a publicação do livro Parallel Distributed Processing Rumelhart e McClelland

Esse conceito foi um marco na área de redes neurais, visto que é computacionalmente bem eficiente.

MLP



MLP

Dois tipos de funções sinais em uma rede MLP:

Sinal: Estímulos que vem da entrada da rede, e são propagados neurônio a neurônio e saem no final da rede como o sinal de saída.

Em cada neurônio o sinal é calculado como função das entradas e dos pesos.

Funções de erro: Sinal que origina da saída da rede, e são retropropagação camada a camada na rede, e são utilizados para ajustar os pesos de um neurônio da rede. Ou seja, esse peso é calculado em uma camada posterior, e retropropagação

MLP

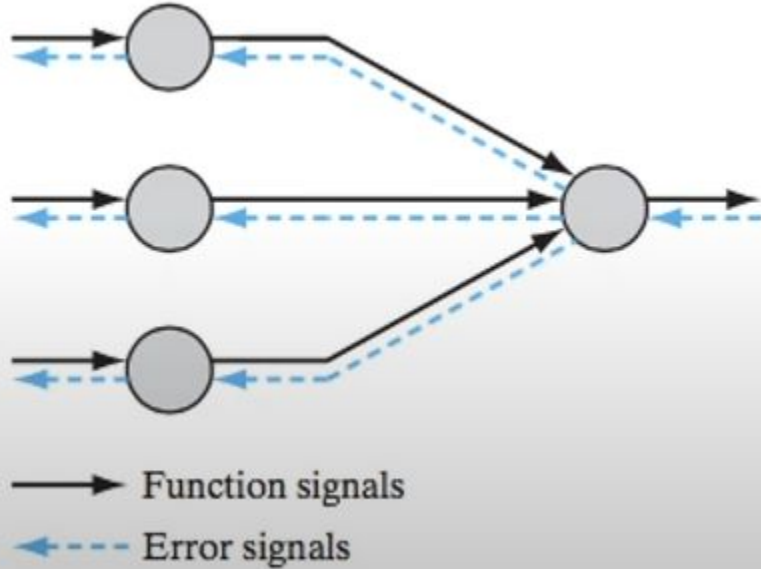


FIGURE 4.2 Illustration of the directions of two basic signal flows in a multilayer perceptron: forward propagation of function signals and back propagation of error signals.

MLP

Os neurônios de saída são a saída da rede

Os outros, são os escondidos, são ditos que não pertencem nem a entrada nem a saída.

Assim a primeira camada escondida é alimentada da entrada da camada

A saída dessa camada é aplicada como entrada para outras camadas escondidas

MLP

Cada neurônio da camada escondida é projetada para fazer dois cálculos.

- The function signal that appears at the output of each neuron, expressed as a continuous nonlinear function of the input signal and associated synaptic weights
- The estimate of the gradient vector (gradients of the error surface with respect to the weights connected to the neuron inputs). Required for the backpropagation phase

MLP

Os neurônios escondidos, agem como detectores de atributos. Conforme o aprendizado progride, os neurônios começam a descobrir os atributos que caracterizam o conjunto de treinamento

Isso é feito por uma transformação não linear da entrada em um novo espaço chamado feature space.

Nesse novo espaço, as classes podem ser facilmente separados de cada um.

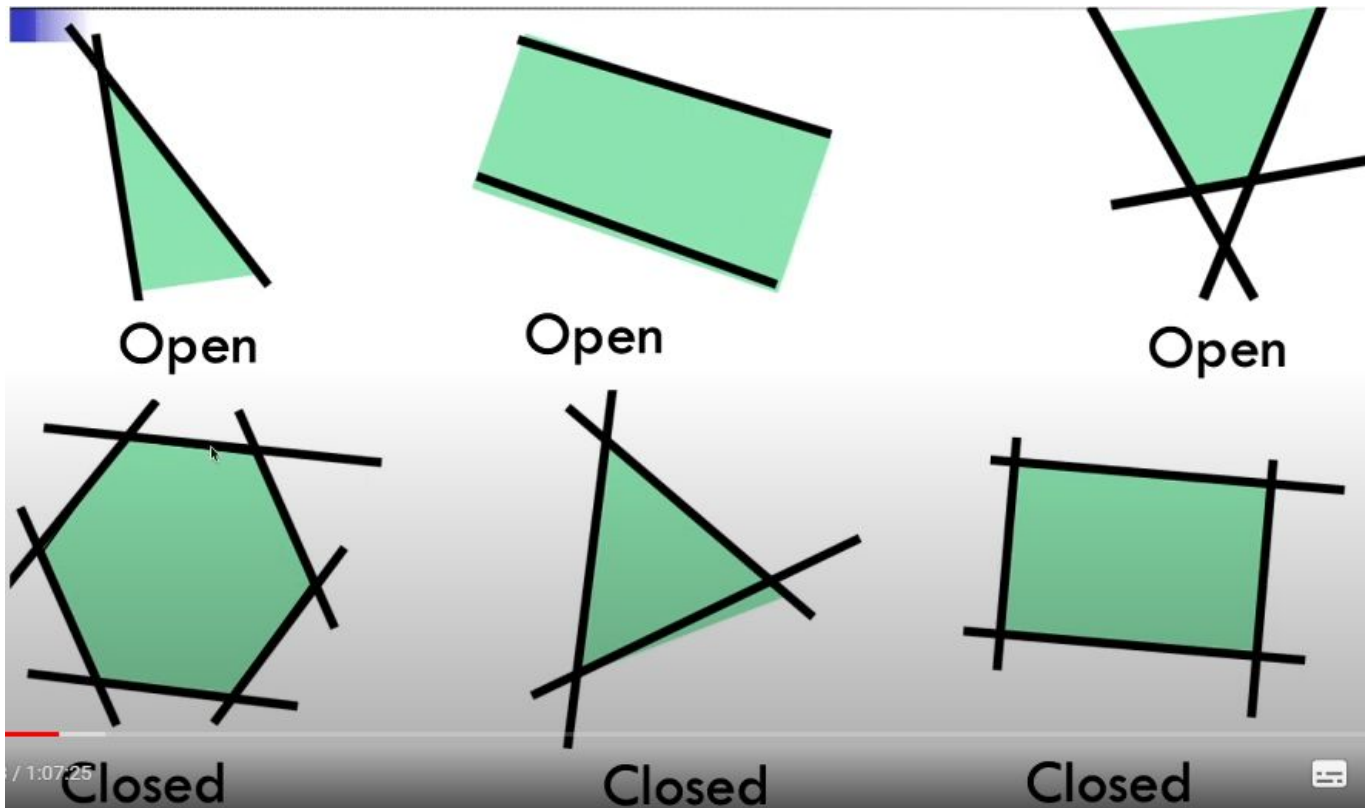
Camadas intermediária:

Primeira camada: Linha retas no espaço de decisão

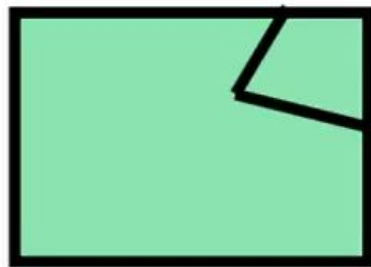
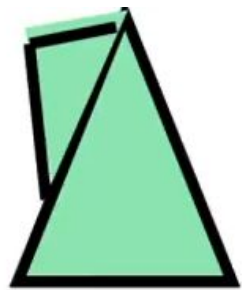
Segunda camada: Combina linhas da camada anterior para formar regiões convexas

Terceira camada: Combina figuras convexas, criando regiões mais complexas.

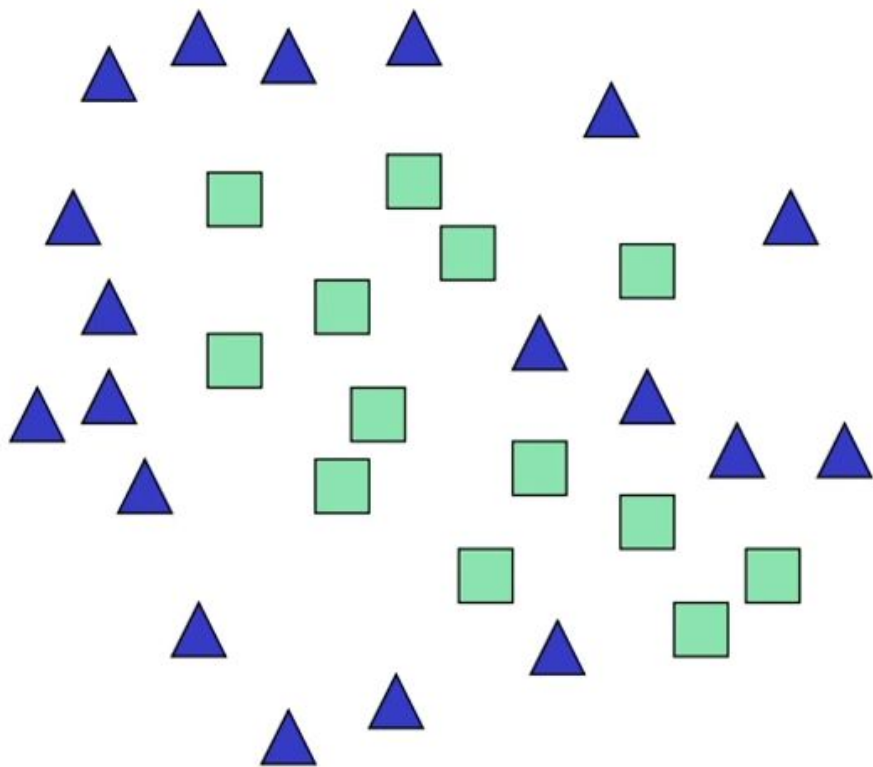
MLP



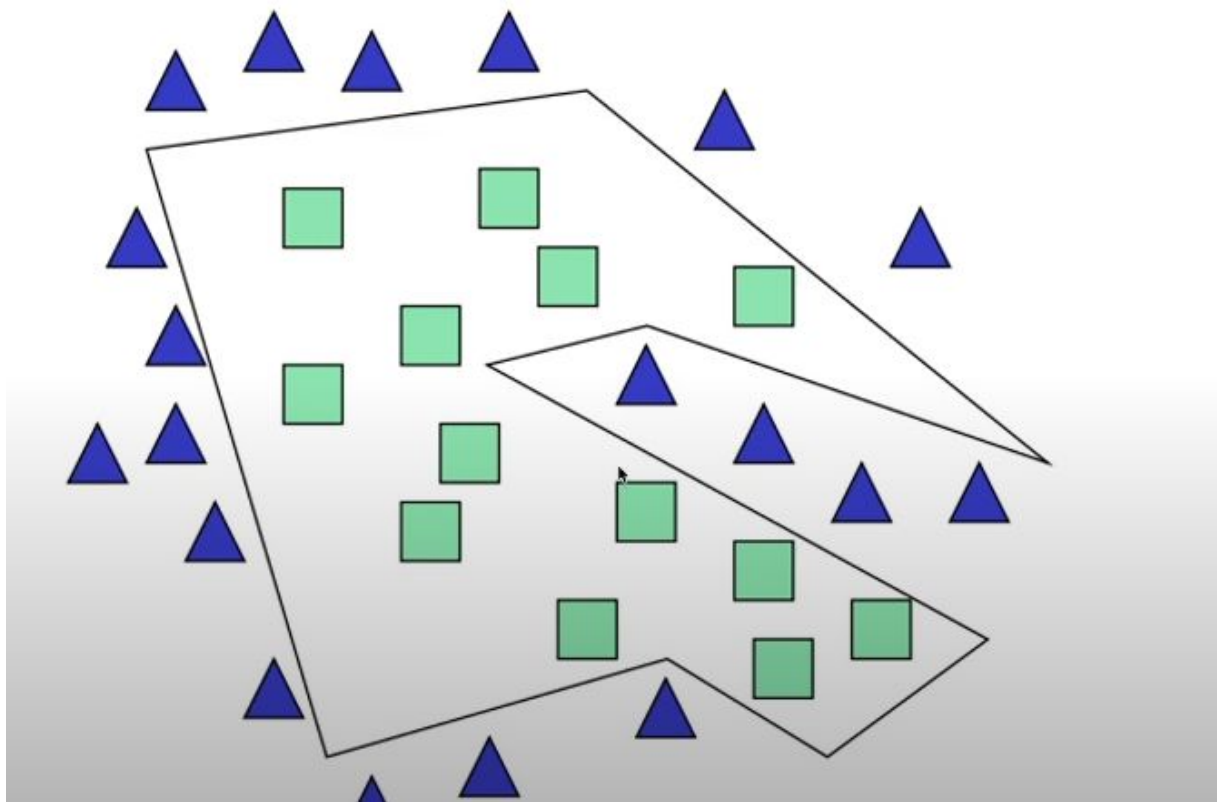
MLP



MLP



MLP



MLP

Consider a Multilayer Perceptron.

Consider $\tau = \{\mathbf{x}(n), \mathbf{d}(n)\}_{n=1}^N$ a training instance. Being $y_j(n)$ the signal produced in the output of neuron j in the output layer, stimulated by $\mathbf{x}(n)$ applied in the input layer

The error signal produced at the output of neuron j is given by $e_j(n) = d_j(n) - y_j(n)$

MLP

The error signal produced at the output of neuron j is given by $e_j(n) = d_j(n) - y_j(n)$, where $d_j(n)$ is the j -th element of the vector of desired responses $\mathbf{d}(n)$

The instantaneous error of neuron j is given by

$$\varepsilon_j(n) = \frac{1}{2} e_j^2(n)$$

MLP

Adding the errors of all neurons in the output layer, the total error of the entire network is given by

$$\varepsilon(n) = \sum_{j \in C} \varepsilon_j(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

C is the set of all output neurons. In a training set with N examples, the average error over all examples (empirical risk) is given by

$$\varepsilon_{av}(N) = \frac{1}{N} \sum_{n=1}^N \varepsilon(n) = \frac{1}{2N} \sum_{n=1}^N \sum_{j \in C} e_j^2(n)$$

MLP

Dois métodos de treinamento:

- Batch: Ajusta os pesos de treinamento, depois do apresentação do treinamento de todos os exemplos da rede uma época completa.

- A curva de aprendizado é plotado, com o erro e o número de épocas, e em cada época, os exemplos são misturados.

- Esse modo permite uma estimativa do vetor gradiente. que é a derivada da função de erro, com relação a cada peso, garantindo uma convergência de um mínimo local.

- Também é melhor para a paralelização

- Mas de forma prática, a busca não é mais estocástica por natureza, e pode ficar presa no mínimo local, e isso é ruim, pois o desejado é chegar no mínimo global

- É mais difícil detectar pequenas mudanças nos dados

- Quando se tem dados redundantes, não pode se tomar vantagens disso, visto que os pesos são calculados no final

- Outro método é o Online training: O ajuste dos pesos ocorre depois da apresentação de cada exemplo.

- E então é minimizado o erro instantâneo da rede inteira.

- Essa busca no espaço de pesos multidimensional se torna estocástico. Por isso, também é chamado de método estocástico.

- Esse método tem menos chance de ficar preso em mínimos locais

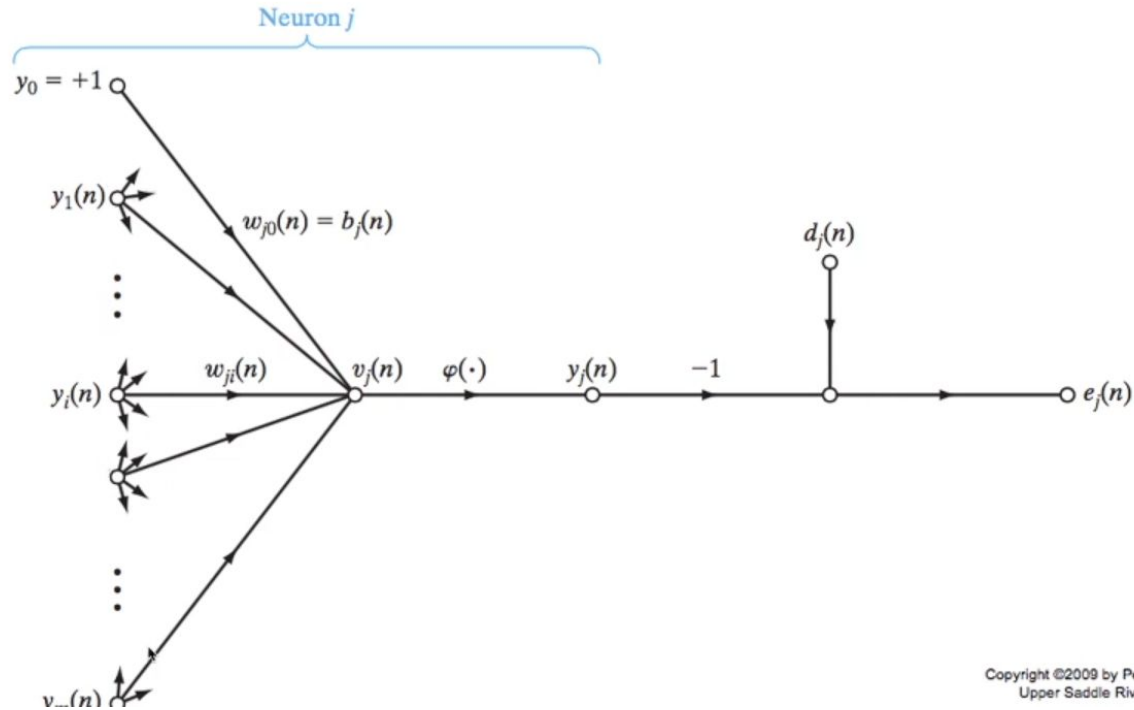
- Além disso, quando se tem redundância, pode ser tirar vantagem disso, já que os pesos são calculados a cada exemplo.

- Detectar melhor as pequenas mudanças nos dados

- É popular em problemas de classificações, visto que são simples de serem implementados e fornece boas soluções para problemas difíceis.

MLP

Neuron j being fed by a set of function signals



MLP

Potencial de ativação é a entrada x o peso

The induced local field (activation potential) $v_j(n)$ produced at the input of the associated activation function is given by

$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n)$$

- ▣ m : number of inputs (excluding the bias)
- ▣ w_{j0} : weight applied to the fixed input $y_0 = +1$ (bias)

MLP

The signal $y_j(n)$ in the output of neuron j at iteration n is :

$$y_j(n) = \varphi_j(v_j(n))$$

The algorithm applies a correction $\Delta w_{ji}(n)$ in the synaptic weight $w_{ji}(n)$, proportional to the partial derivative:

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \varepsilon(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

MLP

The partial derivative $\partial \varepsilon(n) / \partial w_{ji}(n)$ determines the search direction for w_{ji} in the weight space

Differentiating the equation below on both sides with respect to $e_j(n)$:

$$\varepsilon(n) = \sum_{j \in C} e_j^2(n) \rightarrow \frac{\partial \varepsilon(n)}{\partial e_j(n)} = e_j(n)$$

MLP

The partial derivative $\partial \varepsilon(n) / \partial w_{ji}(n)$ determines the search direction for w_{ji} in the weight space

Differentiating the equation below on both sides with respect to $v_j(n)$:

$$y_j(n) = \varphi_j(v_j(n)) \rightarrow \frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'_j(v_j(n))$$

MLP

The partial derivative $\partial \varepsilon(n) / \partial w_{ji}(n)$ determines the search direction for w_{ji} in the weight space

Differentiating the equation below on both sides with respect to $w_{ji}(n)$:

$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n) \rightarrow \frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

MLP

The partial derivative $\partial \varepsilon(n) / \partial w_{ji}(n)$ determines the search direction for w_{ji} in the weight space

Substituting the equations obtained in the chain rule equation, we obtain

$$\frac{\partial \varepsilon(n)}{\partial w_{ji}(n)} = -e_j(n) \varphi'_j(v_j(n)) y_i(n)$$


MLP

The correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is defined by the delta rule:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial w_{ji}(n)}$$

- ▣ η : learning rate of the algorithm
- ▣ The negative sign refers to the gradient descent in the weight space

Using the equation on slide 34 in the above equation:


$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$

MLP

O sinal de erro de um neurônio de saída é um fator chave para calcular o ajuste dos pesos

Mas, há dois casos em que se calcula o erro e se usa de maneira diferente.

- Neurônio está na ultima camada(saída)
- Neurônio está em uma camada escondida, ou seja, não estão diretamente acessíveis

MLP

Neuron j in an output layer

- In this case, the neuron is directly associated with the desired output. Thus, the error is calculated directly :

$$e_j(n) = d_j(n) - y_j(n)$$

- After calculating the error, the local gradient is calculated directly:

$$\delta_j(n) = e_j(n) \varphi'_j(v_j(n))$$

MLP

Neuron j is a hidden neuron

- In this case, there is no specific desired output associated with the neuron.
- The error signal must be recursively calculated, in terms of the error signals of all neurons which to neuron j is directly connected
- At this point the development of back-propagation becomes more complicated

MLP

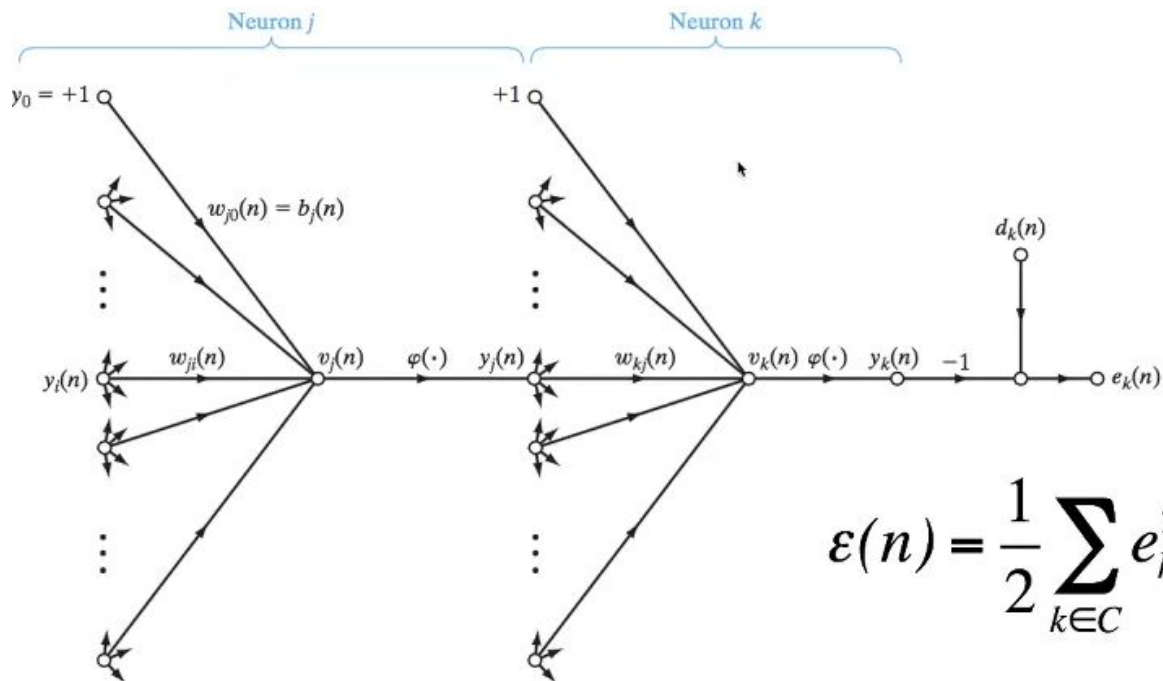
Neuron j is a hidden neuron

- We can redefine the gradient calculation (Slide 36), with the neuron j being a hidden neuron

$$\begin{aligned}\delta_j(n) &= -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \varphi'_j(v_j(n))\end{aligned}$$

MLP

Neuron j is a hidden neuron



$$\varepsilon(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

MLP

Neuron j is a hidden neuron

▣ Differentiating $\varepsilon(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$ with respect to $y_j(n)$:

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_{k \in C} e_k \frac{\partial e_k(n)}{\partial y_j(n)}$$

MLP

Neuron j is a hidden neuron

- Using the chain rule in $\partial e_k(n)/\partial y_j(n)$:

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_{k \in C} e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

- From the figure on slide 41, we have that (neuron k is output):

$$\begin{aligned} e_k(n) &= d_k(n) - y_k(n) \\ &= d_k(n) - \varphi_k(v_k(n)) \end{aligned}$$

MLP

Neuron j is a hidden neuron

- ▣ Thus, we can write the derivative of the activation function for the neuron k :

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k(v_k(n))$$

MLP

Neuron j is a hidden neuron

- From the figure on slide 41, we have that the induced local field of neuron k is given by:

$$v_k(n) = \sum_{j=0}^m w_{kj}(n) y_j(n)$$

- m : total number of inputs (excluding bias)
- $w_{k0}(n)$ is equal to bias $b_k(n)$ applied to neuron k , with a fixed input +1

MLP

Neuron j is a hidden neuron

▣ We have
$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = \sum_{k \in C} e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -\varphi'_k(v_k(n)) \quad \frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

▣ Resulting in

$$\frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k e_k(n) \varphi'_k(v_k(n)) w_{kj}(n) = -\sum_k \delta_k(n) w_{kj}(n)$$

MLP

Neuron j is a hidden neuron

- ▣ Making the substitutions, we have the local gradient of neuron j

$$\delta_j(n) = -\frac{\partial \varepsilon(n)}{\partial y_j(n)} \varphi'_j(v_j(n)) \quad \frac{\partial \varepsilon(n)}{\partial y_j(n)} = -\sum_k \delta_k(n) w_{kj}(n)$$

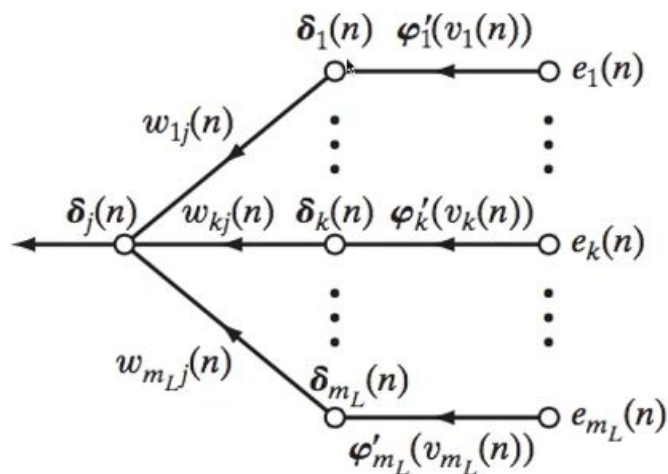
$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

MLP

Neuron j is a hidden neuron

▣ Graphical representation of

$$\delta_j(n) = \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$



MLP

Summarizing: the correction applied to the weights connecting a neuron i to a neuron j is given by the delta rule

$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{Learning} \\ \text{rate} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{Local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \text{Input signal} \\ \text{Neuron } i \\ y_i(n) \end{pmatrix}$$

Output: $\delta_j(n)$ is the product of the derivative $\varphi'_j(v_j(n))$ and the error signal $e_j(n)$, both associated to neuron j

Hidden: $\delta_j(n)$ is the product of the derivative associated to $\varphi'_j(v_j(n))$ and the weighted sum of the δ s calculated for the neurons of the next layer, or the output layer, connected to neuron j

MLP

Funções de ativação:

Para calcular a gradiente de cada neurônio, precisamos saber a derivada da função de ativação associada a cada neurônio

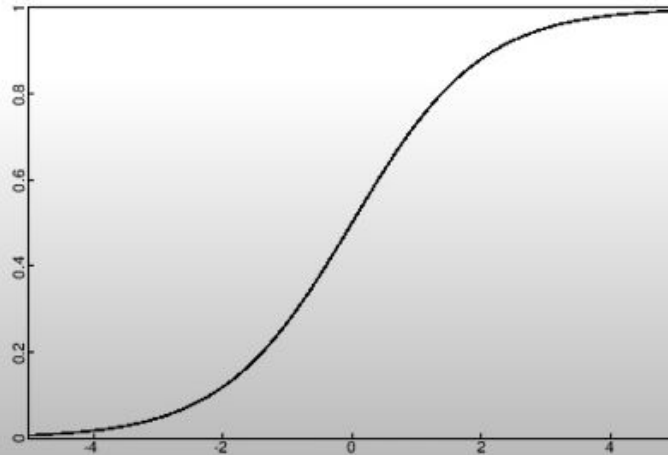
Para essa derivada existir, a função deve ser contínua
A diferenciabilidade é o único requisito que a função precisa satisfazer

Uma função utilizada é a sigmoideal:

MLP

Logistic Function: $\varphi_j(v_j(n)) = \frac{1}{1 + \exp(-av_j(n))}, \quad a > 0$

▣ Output signal amplitude : $0 \leq y_j \leq 1$



MLP

Logistic Function: $\varphi_j(v_j(n)) = \frac{1}{1 + \exp(-av_j(n))}, \quad a > 0$

▣ Output signal amplitude : $0 \leq y_j \leq 1$

▣ The derivative of the function with respect to $v_j(n)$ gives:

$$\varphi'_j(v_j(n)) = \frac{a \exp(-av_j(n))}{[1 + \exp(-av_j(n))]^2}$$

▣ With $y_j(n) = \varphi_j(v_j(n))$, we can rewrite the derivative:

$$\varphi'_j(v_j(n)) = ay_j(n)[1 - y_j(n)]$$

MLP

$$\varphi'_j(v_j(n)) = ay_j(n)[1 - y_j(n)]$$

For a neuron j of the output layer, $y_j(n) = o_j(n)$.
The local gradient of neuron j is given by:

$$\begin{aligned}\delta_j(n) &= e_j(n)\varphi'_j(v_j(n)) \\ &= a[d_j(n) - o_j(n)]o_j(n)[1 - o_j(n)]\end{aligned}$$

MLP

$$\varphi'_j(v_j(n)) = ay_j(n)[1 - y_j(n)]$$

For a neuron j of a hidden layer, the local gradient of neuron j is given by:

$$\begin{aligned}\delta_j(n) &= \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \\ &= ay_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n)\end{aligned}$$

MLP

Hyperbolic tangent: $\varphi_j(v_j(n)) = a \tanh(bv_j(n))$

- ▣ a and b are positive constants
- ▣ Output signal amplitude: $-1 \leq y_j \leq 1$
- ▣ The derivative of the function with respect to $v_j(n)$ gives:

$$\begin{aligned}\varphi'_j(v_j(n)) &= ab \operatorname{sech}^2(bv_j(n)) \\ &= ab(1 - \tanh^2(bv_j(n))) \\ &= \frac{b}{a}[a - y_j(n)][a + y_j(n)]\end{aligned}$$

MLP

$$\varphi'_j(v_j(n)) = \frac{b}{a} [a - y_j(n)] [a + y_j(n)]$$

For a neuron j of the output layer, the local gradient of neuron j is given by:

$$\begin{aligned}\delta_j(n) &= e_j(n) \varphi'_j(v_j(n)) \\ &= \frac{b}{a} [d_j(n) - o_j(n)] [a - o_j(n)] [a + o_j(n)]\end{aligned}$$

MLP

$$\varphi'_j(v_j(n)) = \frac{b}{a} [a - y_j(n)] [a + y_j(n)]$$

For a neuron j of a hidden layer, the local gradient of neuron j is given by:

$$\begin{aligned}\delta_j(n) &= \varphi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \\ &= \frac{b}{a} [a - y_j(n)] [a + y_j(n)] \sum_k \delta_k(n) w_{kj}(n)\end{aligned}$$

MLP

Quanto menor a taxa de aprendizagem, menor é a mudança nos pesos sinápticos da rede, e mais suave é a trajetória do espaço de busca
Aprendizado será mais devagar

Aumentando a taxa de aprendizado, pode-se ter um aprendizado mais rápido, com mudanças mais significativas nos pesos sinápticos

A rede pode se tornar instável, visto que os pesos podem não ser ajustados corretamente.

MLP

When the partial derivative $\partial \varepsilon(n) / \partial w_{ji}(n)$ has the same sign in consecutive iterations, $\Delta w_{ji}(n)$ grows, and $w_{ji}(n)$ is adjusted by a large amount. Thus the momentum constant accelerates the algorithm in regions of constant descent on the error surface.

If the partial derivative $\partial \varepsilon(n) / \partial w_{ji}(n)$ has opposite signs in consecutive iterations, $\Delta w_{ji}(n)$ shrinks, and $w_{ji}(n)$ is adjusted in small quantities. Thus, the momentum constant has a stabilizing effect in directions in which the signal oscillates.

MLP

Critérios de parada:

Em geral, não pode-se afirmar que o algoritmo de backpropagation convergiu.

Uma condição pode ser verificada, que o gradiente do vetor w deve ser zero quando $w = w^*$, em que w^* é o vetor de pesos ajustado.

A convergência é considerada, quando a norma euclidiana do gradiente é suficientemente pequena

Outra condição é utilizando uma função de custo, em que se verifica o erro, e quanto que ele diminui,

se a mudança nesse erro, for suficientemente pequeno, o treinamento para

A outra condição é a cada iteração testar a generalização da rede neural

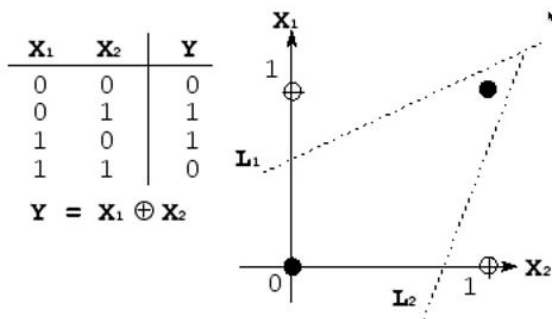
MLP

O problema XOR:

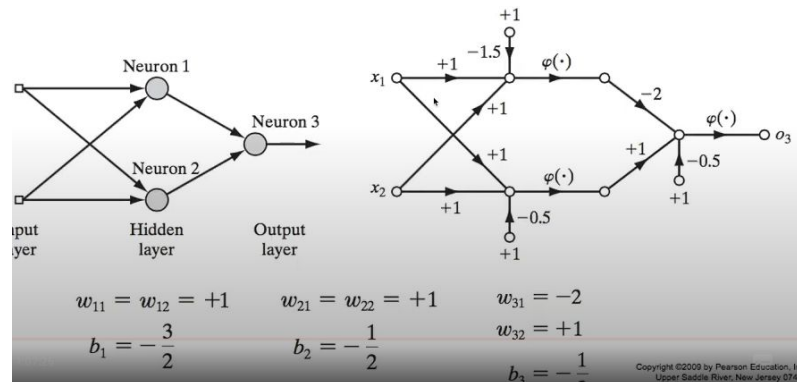
O perceptron não pode classificar padrões linearmente separáveis, visto que um perceptron pode traçar apenas um hiperplano. visto que é necessário combinar hiperplanos.

Mas podemos solucionar esse problema utilizando uma camada escondida com dois neurônios:

Rosenblatt's Perceptron cannot classify nonlinearly separable patterns

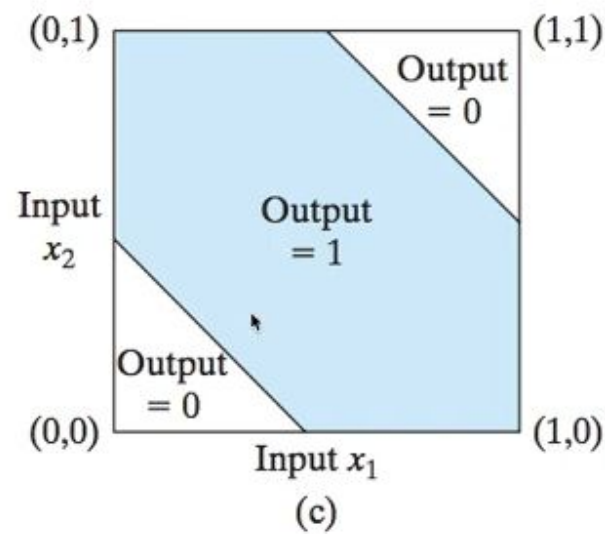
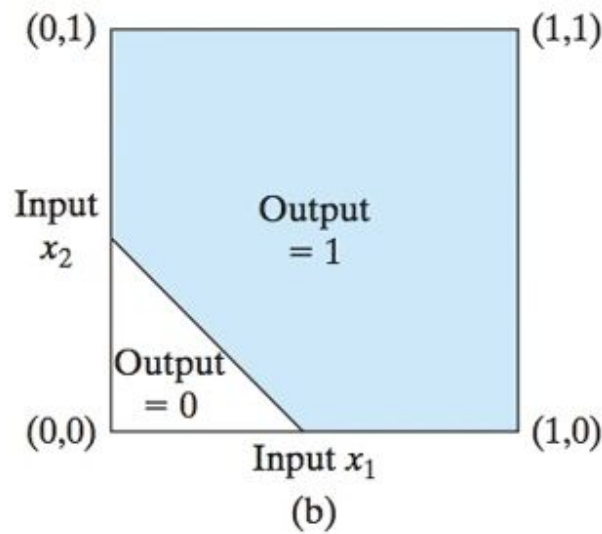
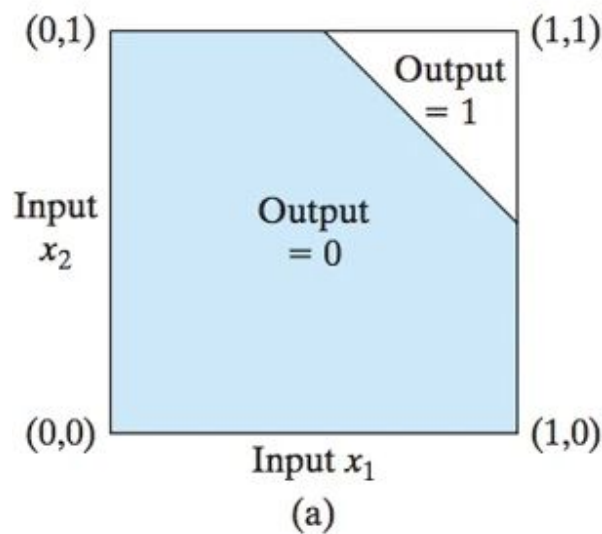


We can solve the problem using a hidden layer with two neurons



MLP

- We can solve the problem using a hidden layer with two neurons



MLP

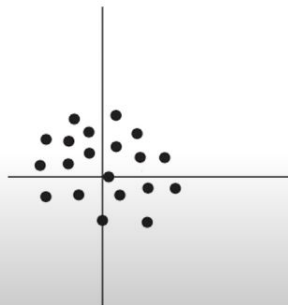
Heurísticas que podem ser utilizados para melhorar a performance

Treinamento online pode ser computacionalmente mais eficiente do que o treinamento batch, quando se tem muita redundância nos dados.

Os valores desejados devem ser escolhidos para ficarem entre os limiares de ativação.

Normalization: attributes can be pre-processed to have an average close to 0

- ▣ Improves convergence
- ▣ Important considering the parameters (bias and weights)
- ▣ Bias: distance from the hyperplane to the origin
- ▣ Weights: hyperplane orientation



MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP

MLP