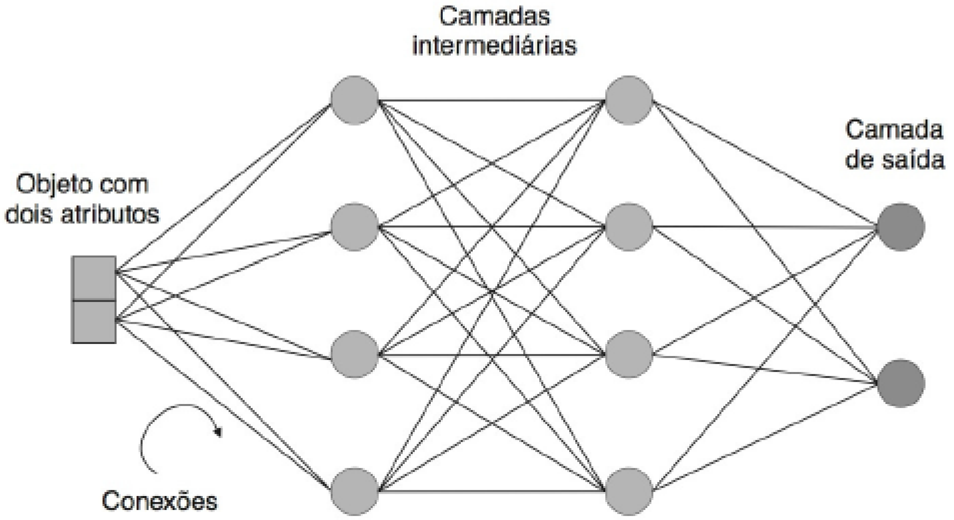
REDES NEURAIS BACKPROPAGATION

Cristiane Neri Nobre

MLP – *MultiLayer* Perceptron ou perceptron multicamadas

- Para resolver problemas não linearmente separáveis utilizando RNAs, a alternativa mais utilizada é adicionar uma ou mais camadas intermediárias.

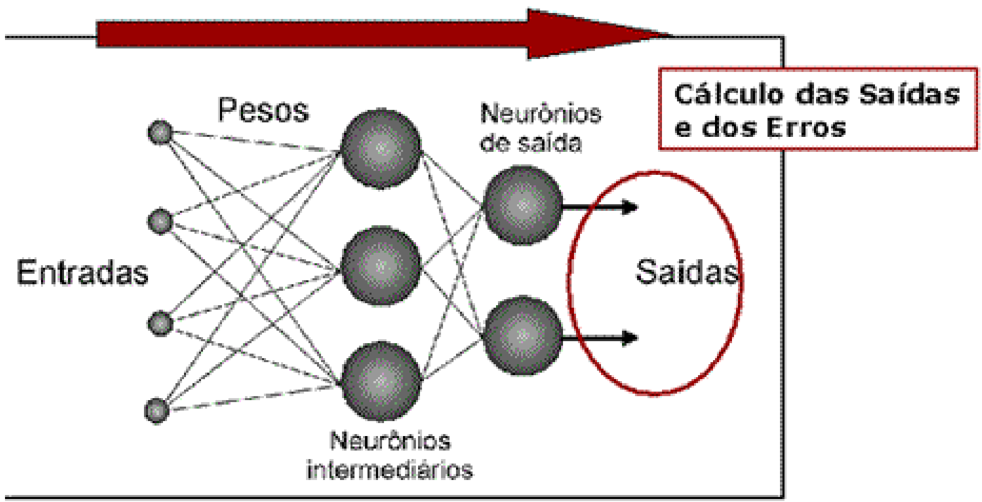
- As redes do tipo MLP apresentam uma ou mais camadas intermediárias de neurônios e uma camada de saída.



Algoritmo Backpropagation - MLP

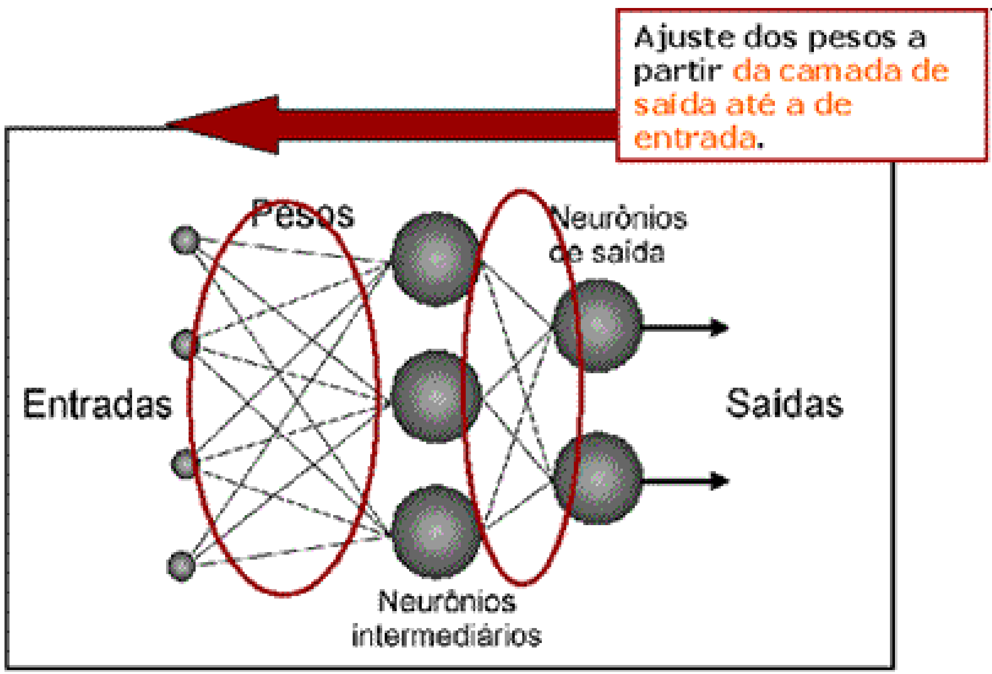
"**Backpropagation**" é um algoritmo para treinamento de Redes Multi-Camadas mais difundido. Baseia-se no **Aprendizado Supervisionado por Correção de Erros**, constituido de:

1° - Propagação (fase *forward*): Depois de apresentado o padrão de entrada, a resposta de uma unidade é propagada como entrada para as unidades na camada seguinte, até a camada de saída, onde é obtida a resposta da rede e o erro é calculado;



Algoritmo Backpropagation - MLP

2° - Retropropagação (fase *backward*) Desde a camada de saída até a camada de entrada, são feitas alterações nos pesos sinápticos.



Fase de Retropropagação

Algoritmo *Backpropagation* - MLP

Algoritmo 7.2 Algoritmo de treinamento *back-propagation*

```
Entrada: Um conjunto de $n$ objetos de treinamento
Saída: Rede MLP com valores dos pesos ajustados
Inicializar pesos da rede com valores aleatórios
Inicializar $erro_{total} = 0$
repita
    para cada objeto $\mathbf{x}_i$ do conjunto de treinamento faça
        para cada camada da rede, a partir da primeira camada intermediária faça
            para cada cada neurônio $n_{jl}$ da camada atual faça
                Calcular valor da saída produzida pelo neurônio, $\hat{f}$
            fim
        fim
        Calcular $erro_{parcial} = y - \hat{f}$
        para cada camada da rede, a partir da camada de saída faça
            para cada cada neurônio $n_{jl}$ da camada atual faça
                Ajustar pesos do neurônio utilizando Equação 7.3
            fim
        fim
        Calcular $erro_{total} = erro_{total} + erro_{parcial}$
    fim
até $erro_{total} < \xi$;
```

$$w_{jl}(t+1) = w_{jl}(t) + \eta x^j \delta_l \quad (7.3)$$

Extraído do livro texto adotado na disciplina

Backpropagation - Inicialização

1 - Inicialização: Inicialize os pesos sinápticos e os bias aleatoriamente

Backpropagation – Computação para frente

2 - Computação para frente (propagação):

Depois de apresentado o exemplo do conjunto de treinamento T = {(x(n),d(n)}, sendo x(n) a entrada apresentada à rede e d(n) a saída desejada, calcule o valor da ativação v_j e a saída para cada unidade da rede, da seguinte forma:

$$v_j = \sum_{i=1}^{m} w_{ji} x_i + b x_0$$

para o cálculo do valor da ativação e

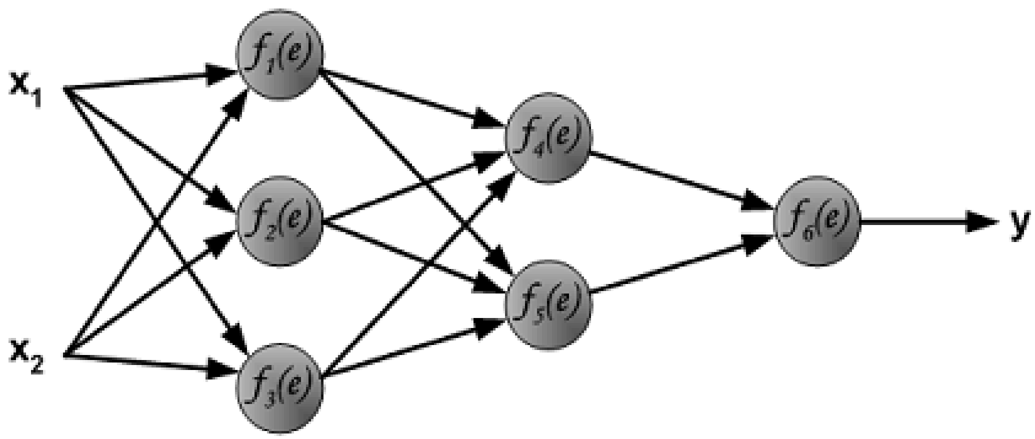
$$f(v) = \frac{1}{1 + e^{(-av)}}$$

para o cálculo da saída *y* da unidade *k*, utilizando a função sigmóide, como no exemplo, ou uma outra função, se necessário.

Utilize a saída das unidades de uma camada como entradas para a seguinte, até a última camada. A saída das unidades da última camada será a resposta da rede.

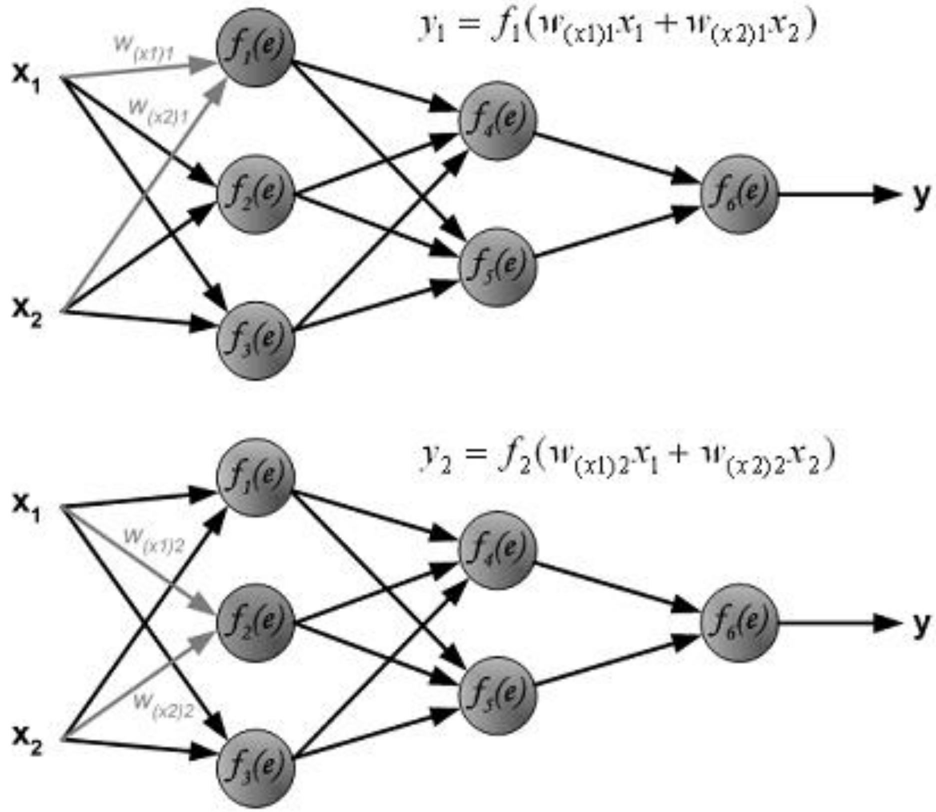
Backpropagation – Computação para frente

Exemplificação do passo 2:



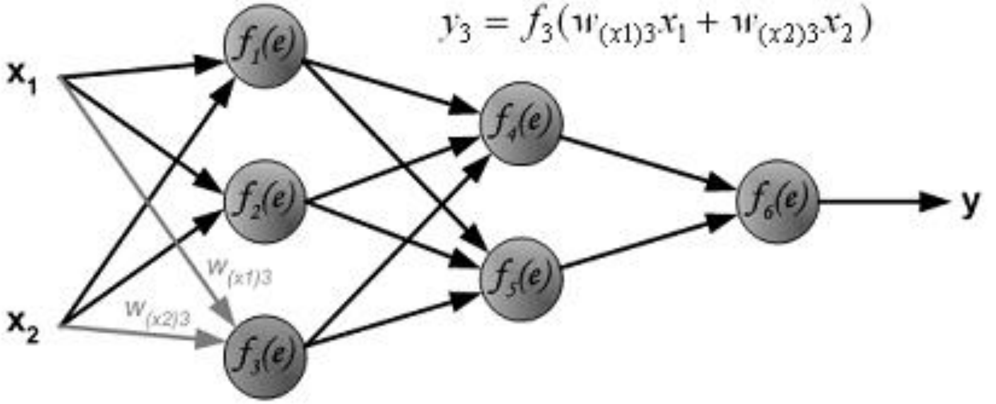
Backpropagation – Computação para frente

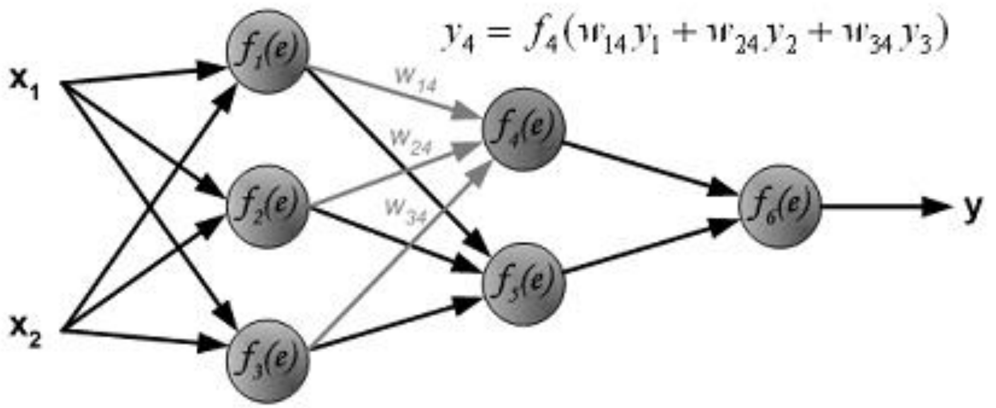
Exemplificação do passo 2:



Backpropagation – Computação para frente

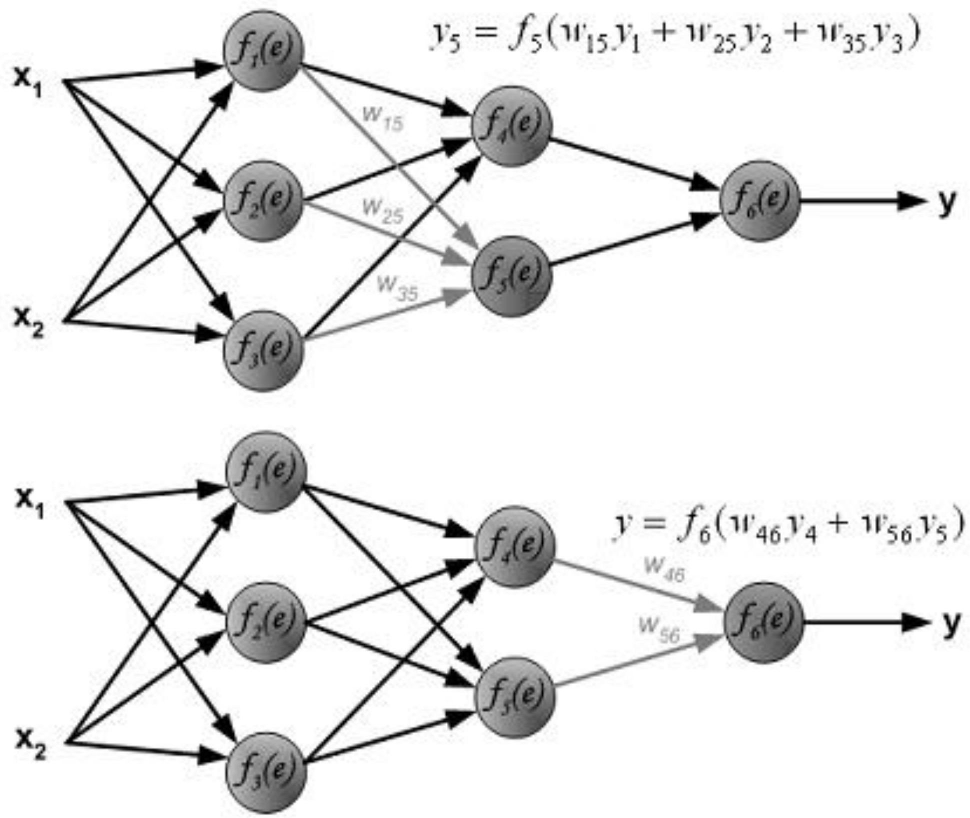
Exemplificação do passo 2:





Backpropagation – Computação para frente

Exemplificação do passo 2:



Backpropagation – propagação do erro (propagação do erro)

3 – Propagação do erro para trás

Como calcular o erro da rede?

Como propagar o erro para trás?

Backpropagation – propagação do erro (propagação do erro)

Como calcular o erro?

Depende em qual camada o neurônio se encontra:

✓ **Se pertencer à camada de saída**

$$\delta_j = f'(net_j)*(d_j - y_i)$$

Ao invés de usar a diferença absoluta, pode-se utilizar também o **erro médio quadrático (Mean Square Error – MSE) ou Root Mean Square Error – RMSE)**

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(S_i - O_i\right)^2}$$

Backpropagation – propagação do erro (propagação do erro)

Vamos ver um exemplo o cálculo do MSE

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

X	Y	Saída desejada (XOR)	Saída real da rede	Erro
0	0	0	0,34	$(0 - 0{,}34)^2 = 01156$
0	1	1	0,57	$(1 - 0{,}57)^2 = 0{,}1849$
1	0	1	0,35	$(1 - 0{,}35)^2 = 0{,}4225$
1	1	0	0,46	$(0 - 0{,}46)^2 = 0{,}2126$
			Soma	**0,9356**
			MSE (Soma/4)	**0,2339**

Backpropagation - propagação do erro (propagação do erro)

Se pertencer à camada intermediária:

$\delta_j = \mathrm{f'(net_j)} \times \sum_l \delta_l w_{lj}$

onde,
w_{lj} = Pesosdasconexões
δ_l = ErroDaCamadaposterior

Backpropagation - propagação do erro (propagação do erro)

Resumindo

A forma para se calcular o erro depende da camada onde se encontra o neurônio

$$\delta_l = \begin{cases} f'_a e_l, & \text{se } n_l \in c_{sai} \\ f'_a \sum w_{lk}\delta_k, & \text{se } n_l \in c_{int} \end{cases}$$

Backpropagation - propagação do erro (propagação do erro)

Ou seja, como os valores dos erros são conhecidos apenas para os neurônios da camada de saída, o erro para os neurônios da camada intermediária precisam ser **estimados**

Backpropagation – Importância do gradiente

Importância do gradiente

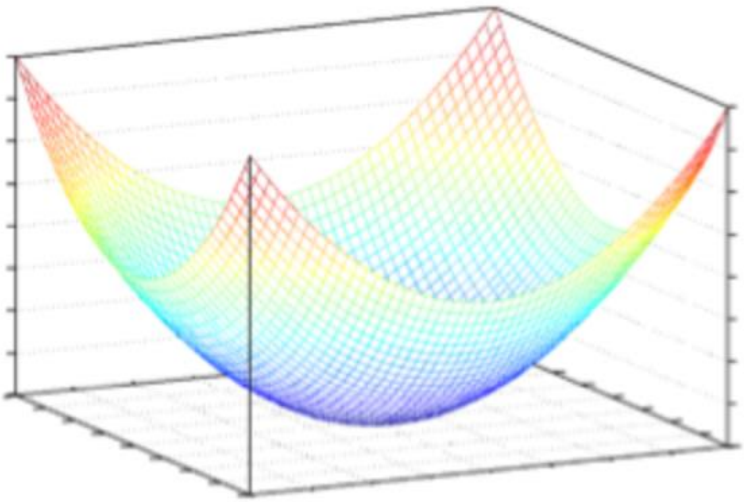
$$\min C(w_1, w_2, \ldots w_n)$$

O objetivo é calcular a derivada parcial para mover para a direção do gradiente

Ou seja, o objetivo é encontrar a combinação de pesos que **o erro é o menor possível**

Gradiente é calculado para saber quanto **ajustar os pesos**

Backpropagation – Importância do gradiente



Backpropagation –Importância da derivada

A derivada parcial define o ajuste dos pesos, utilizando o gradiente descendente da função de ativação.

Essa derivada mede a contribuição de cada peso no erro da rede para a classificação de um dado objeto x.

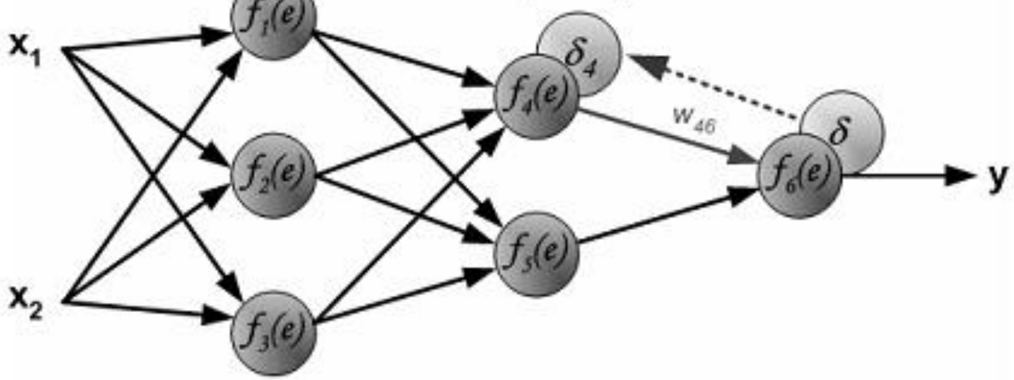
Se essa derivada para um dado peso for **positiva**, o peso está provocando um **aumento da diferença** entre a saída da rede e a saída desejada. Assim, sua magnitude deve ser **reduzida** para **baixar** o erro.

Se a derivada for **negativa**, o peso está contribuindo para que a saída produzida pela rede seja mais próxima da desejada. Dessa forma, seu valor deve ser **aumentado**.

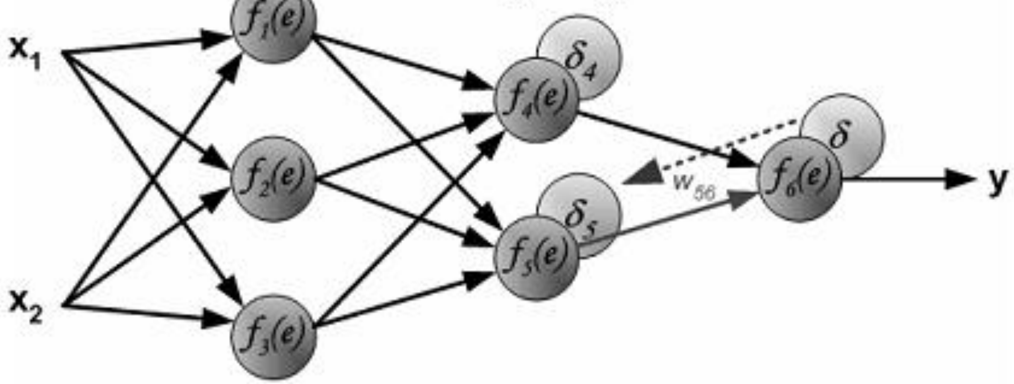
Backpropagation – propagação do erro (propagação do erro)

Exemplificação do passo 3 – propagação do erro:

$$\delta_4 = w_{46}\delta$$

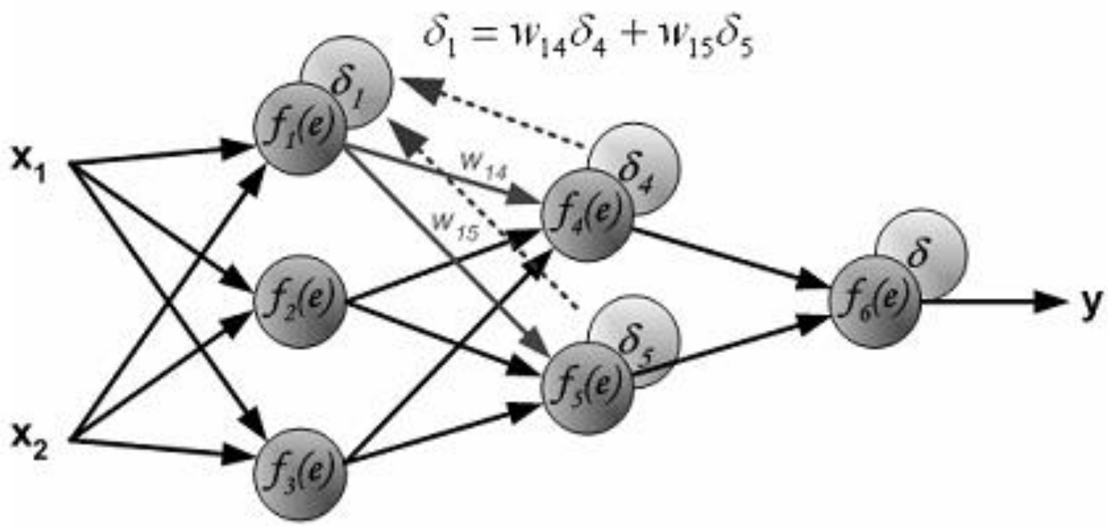


$$\delta_5 = w_{56}\delta$$



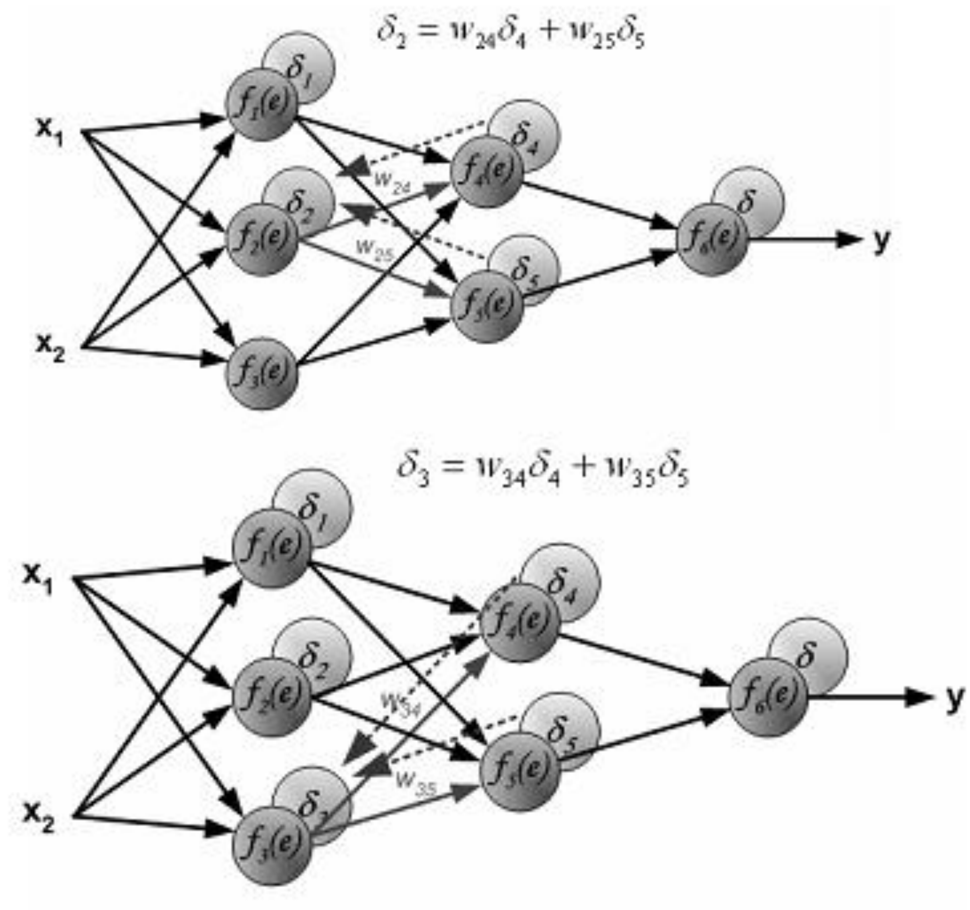
Backpropagation – propagação do erro (propagação do erro)

Exemplificação do passo 3 – propagação do erro:



Backpropagation – propagação do erro

Exemplificação do passo 3 – propagação do erro:



Backpropagation – Ajuste dos pesos

4 – Ajuste dos pesos

Após o cálculo dos erros é necessário corrigir os pesos das ligações entre os neurónios. O cálculo do novo peso é dado pela fórmula:

$$wij(t+1) = wij(t) + Ta * xi * erro$$

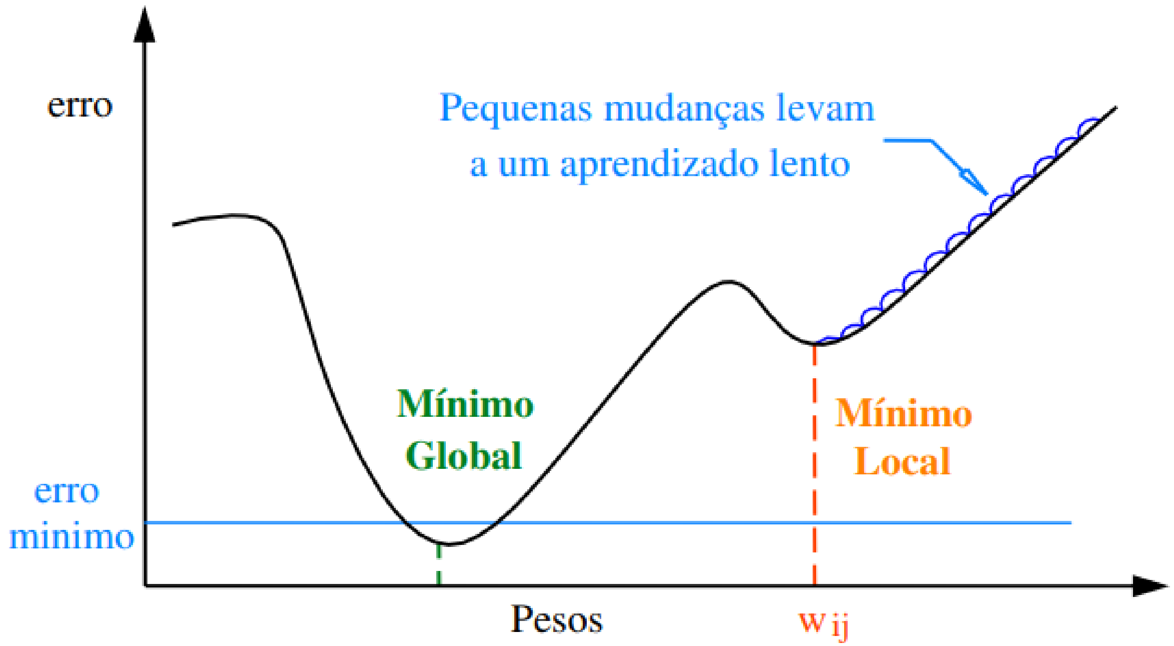
Onde,
w_{ji} representa o peso entre um neurônio i e o j-ésimo atributo de entrada ou a saída do j-ésimo neurônio da camada anterior

$erro_j$ indica o erro associado ao l-ésimo neurônio

x_i indica a entrada recebida por esse neurônio (o i-ésimo atributo de entrada ou a saída do i-ésimo neurônio da camada anterior)

Backpropagation – Ajuste dos pesos

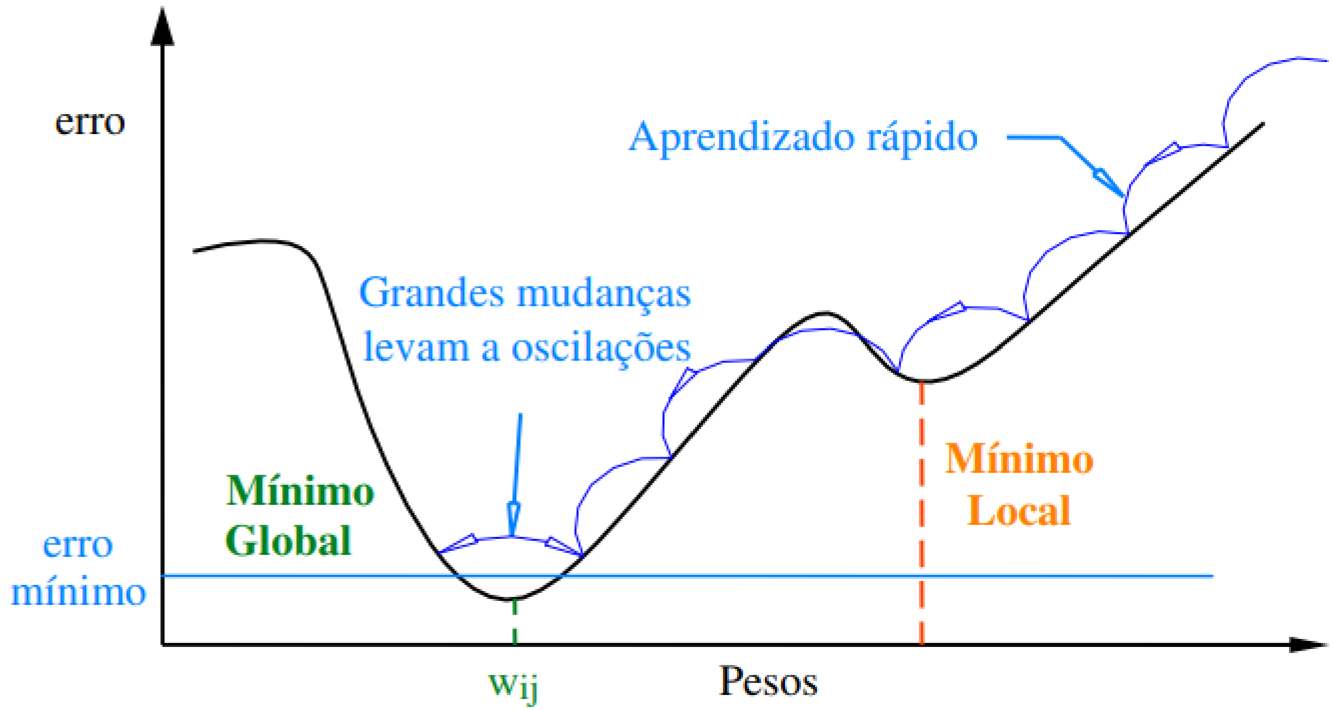
Importância da taxa de aprendizado (sugestão que seja baixa)



Extraído de https://www.maxwell.vrac.puc-rio.br/32823/32823_4.PDF

Backpropagation – Ajuste dos pesos

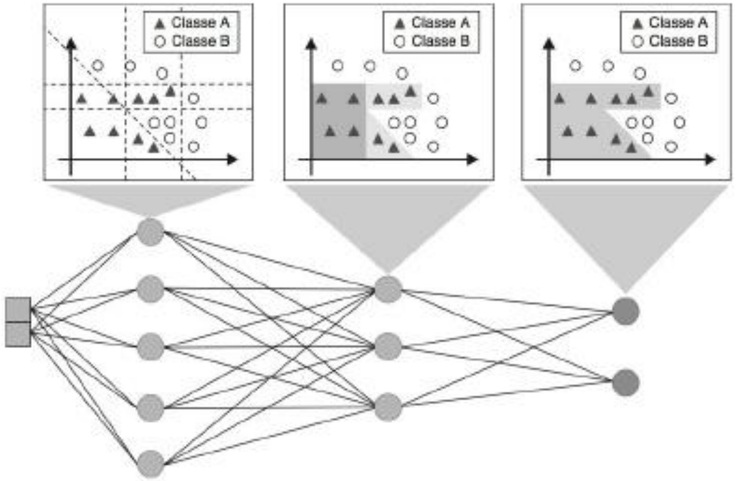
Importância da taxa de aprendizado (valores altos de taxas de aprendizado)



Extraído de https://www.maxwell.vrac.puc-rio.br/32823/32823_4.PDF

MPL – papel dos neurônios das camadas ocultas

- ✓ Em uma MLP, cada neurônio realiza uma função específica.
- ✓ A função implementada por um neurônio de uma dada camada é uma combinação das funções realizadas pelos neurônios da camada anterior que estão conectados a ele.
- ✓ À medida que o processamento avança de uma camada intermediária para a camada seguinte, o processamento realizado se torna mais complexo
- ✓ Na primeira camada, cada neurônio aprende uma função que define um hiperplano, o qual divide o espaço de entrada em duas partes
- ✓ Cada neurônio da camada seguinte combina um grupo de hiperplanos definidos pelos neurônios da camada anterior, formando regiões convexas.
- ✓ Os neurônios da camada seguinte combinam um subconjunto das regiões convexas em regiões de formato arbitrário.
- ✓ É a combinação das funções desempenhadas por cada neurônio da rede que define a função associada à RNA como um todo



Extraído do livro texto adotado na disciplina

Derivadas de algumas funções de ativação

Métodos de treinamento como o *backpropagation* exigem que a função de ativação possua derivada.

Função	Derivada
Função sigmoide: $o(x) = \frac{1}{1+e^{-x}}$	$f'(x) = o(x)(1-o(x))$
tangente hiperbólica: $o(x) = \frac{e^{2x}-1}{e^{2x}+1}$	$f'(x) = 1-o(x)^2$

Algoritmo Backpropagation - Considerações

- Para cada vetor de treinamento, o sinal deve ser propagado das entradas para as saídas para que o erro possa então propagar em sentido contrário e permitir o treinamento

- Há normalmente duas formas de atualização dos pesos:

  1- Atualiza-se os pesos a cada instância classificada incorretamente

  - ✓ Stochastic gradient descent

  2 - Atualiza-se os pesos ao final de todas as instâncias apresentadas

  - ✓ Batch gradient descent – e tem variação de 'mini batch gradiente descent', em que o usuário entra com o número de registros desejado

- Cada apresentação completa de todos os elementos do conjunto de treinamento e consequente ajuste de pesos é chamada *epoch*

- É aconselhável randomizar a sequência com que os vetores são apresentados à rede de uma *epoch* para a outra para acrescentar um componente estocástico ao treinamento e evitar ciclos limites indesejáveis na atualização dos pesos

Heurísticas para número de neurônios

- **Regra do valor médio** - De acordo com esta fórmula o número de neurônios da camada escondida é igual ao valor médio do número de entradas e o número de saídas da rede, ou seja:

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

- **Regra da raiz quadrada** - De acordo com esta fórmula o número de neurônios da camada escondida é igual a raiz quadrada do produto do número de entradas pelo número de saídas da rede, ou seja:

$$q = \sqrt{p \cdot M}$$

Heurísticas para número de neurônios

- **Regra de Kolmogorov** - De acordo com esta fórmula o número de neurônios da camada escondida é igual a duas vezes o número de entradas da rede adicionado de 1, ou seja:

$$q = 2p + 1$$

Outras sugestões:

- O número de neurônios deve estar entre o número de entrada e saída
- 2/3 do tamanho da camada de entrada, somado ao tamanho da camada de saída
- Menor que duas vezes o tamanho da camada de entrada

Exercício

- Abra uma base de dados do seu interesse e investigue o use o Backpropagation

- Investigue os hiperparâmetros:

hiddenLayers, batchSize, trainingTime, learningRate, momentum

- Investigue a relação entre learningRate e o trainingTime

- Investigue o desempenho da rede alterando o número de neurônios e camadas. Veja a estrutura da rede.

- Investigue também o hiperparâmetro NominaltoBinaryFilter para atributos nominais. Veja o seu efeito, mostrando a estrutura da rede visualmente.

Referência

Veja capítulo 7 do livro texto adotado na disciplina