Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Aprendizado Profundo

Treinamento de RNAs e Algoritmo da Retropropagação

Leonardo Vidal

Thais Gaudencio

Tiago Maritan

Motivação

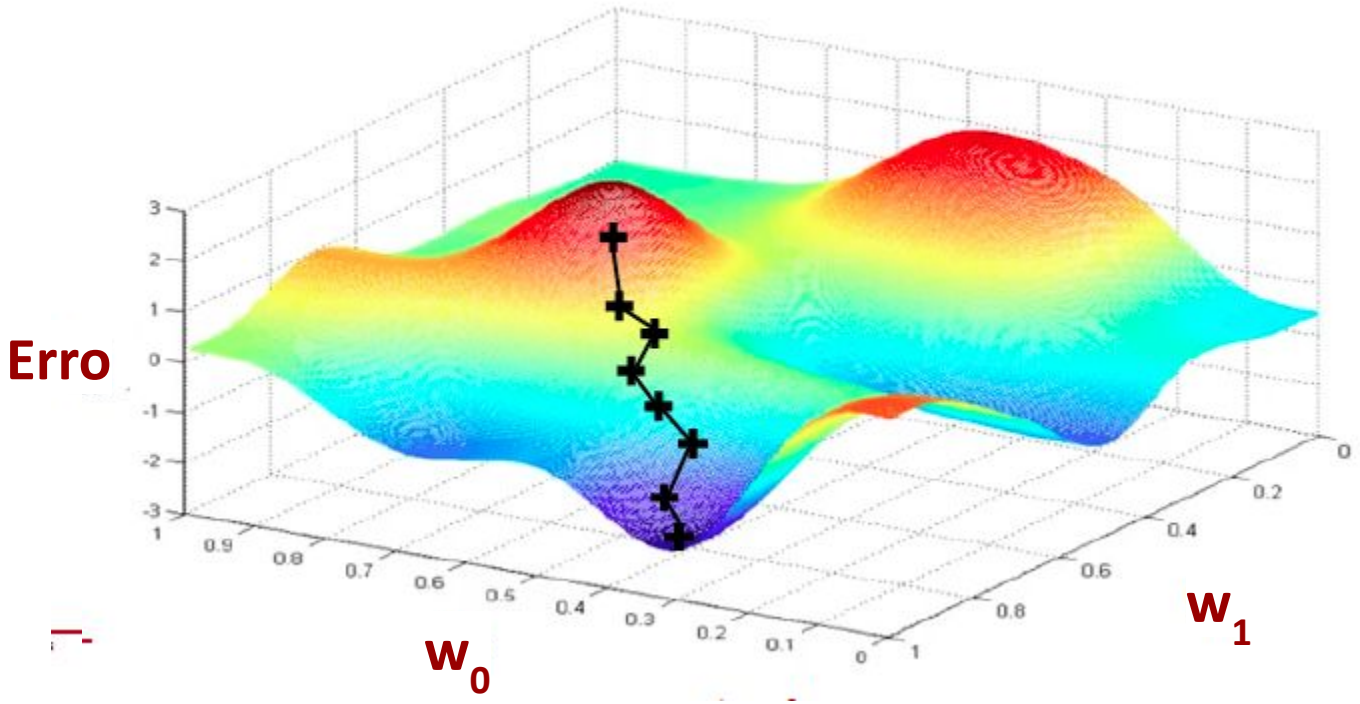
- Treinamento de uma RNA é um **Problema de Otimização.**

"Encontre o vetor (matriz) de pesos sinápticos (W*)
que minimizem o Erro (*E*) entre a
saída da RNA e a saída desejada".

$$E(\mathbf{W}^*) \leq E(\mathbf{W})$$

Treinamento de Redes Neurais

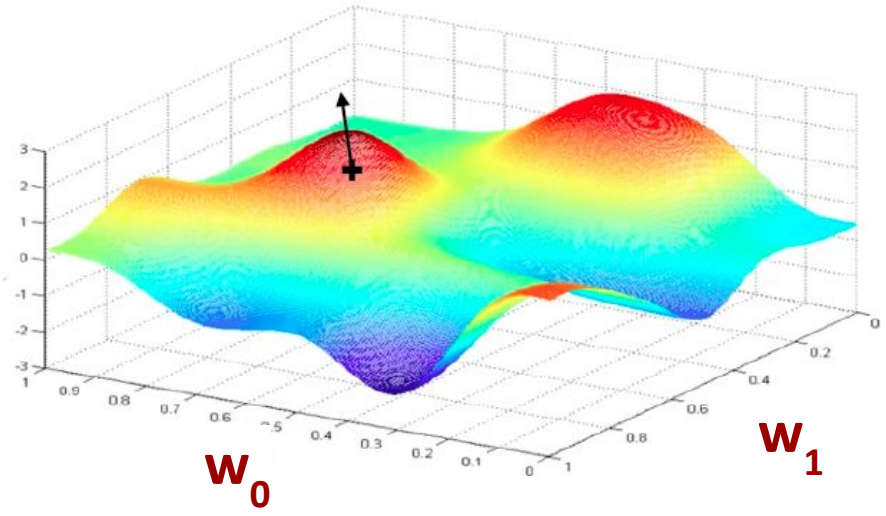
- Algoritmo clássico é baseado no **Método do Gradiente**.
  - Método iterativo que, em cada passo, toma-se a direção de descida do Gradiente (**direção de máximo declive**).



Gradiente de uma Função

- Vetor que indica **o sentido e a direção** na qual obtém-se **o maior incremento possível no valor de uma grandeza** a partir de um ponto especificado.

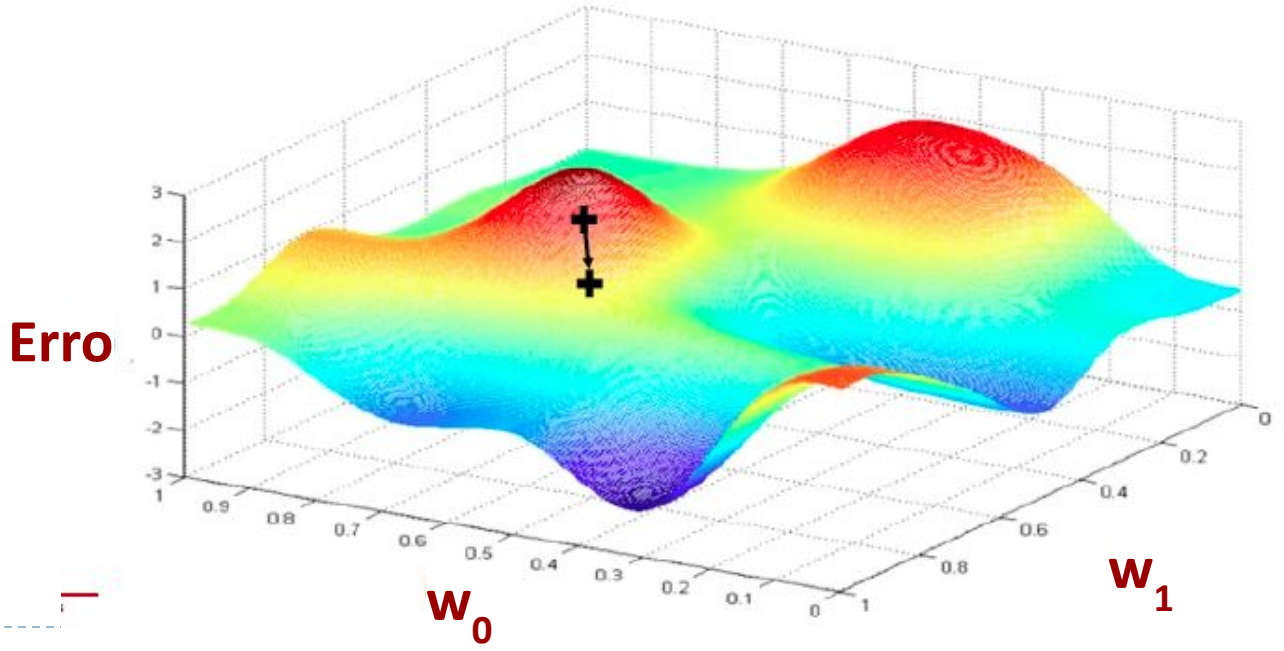
$$\nabla E = \left(\frac{\partial E}{w_0}, \frac{\partial E}{w_1}, \ldots, \frac{\partial E}{w_n} \right)$$



Gradiente de uma Função

- Direção oposta a do gradiente é a **direção de maior declive**.

$$-\nabla E = -\left(\frac{\partial E}{w_0}, \frac{\partial E}{w_1}, \ldots, \frac{\partial E}{w_n}\right)$$



Algumas Consideracões...

- Erro na saída do neurônio *j* para o exemplo *n*:

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

- Erro instantâneo para todos os neurônios da camada de saída *C, para um dado exemplo n:*

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

- Erro médio quadrático para todo o conjunto de treinamento:

$$E_{med} = \frac{1}{N}\sum_{n=1}^{N} E(n)$$

Treinamento de Redes Neurais

```
function rna-learning(training-examples)
    network <- initialize_weights(randomly)
    start loop
        for each example in training_examples do
            network_out = rna_output(network, example)
            example_err = actual_out - network_out
            Atualiza pesos baseado na direção oposta ao
            gradiente(network, example_err)
        end for
    end loop quando todos os exemplos forem
    corretamente preditos ou critérios de parada
    forem atingidos
    return network
```

Treinamento de Redes de Múltiplas Camadas

- Considere a rede de múltiplas camadas a seguir:

Entrada x(n)

Saída y(n)

Base de dados:
[x(1), d(1)]
[x(2), d(2)]
...
[x(m), d(m)]

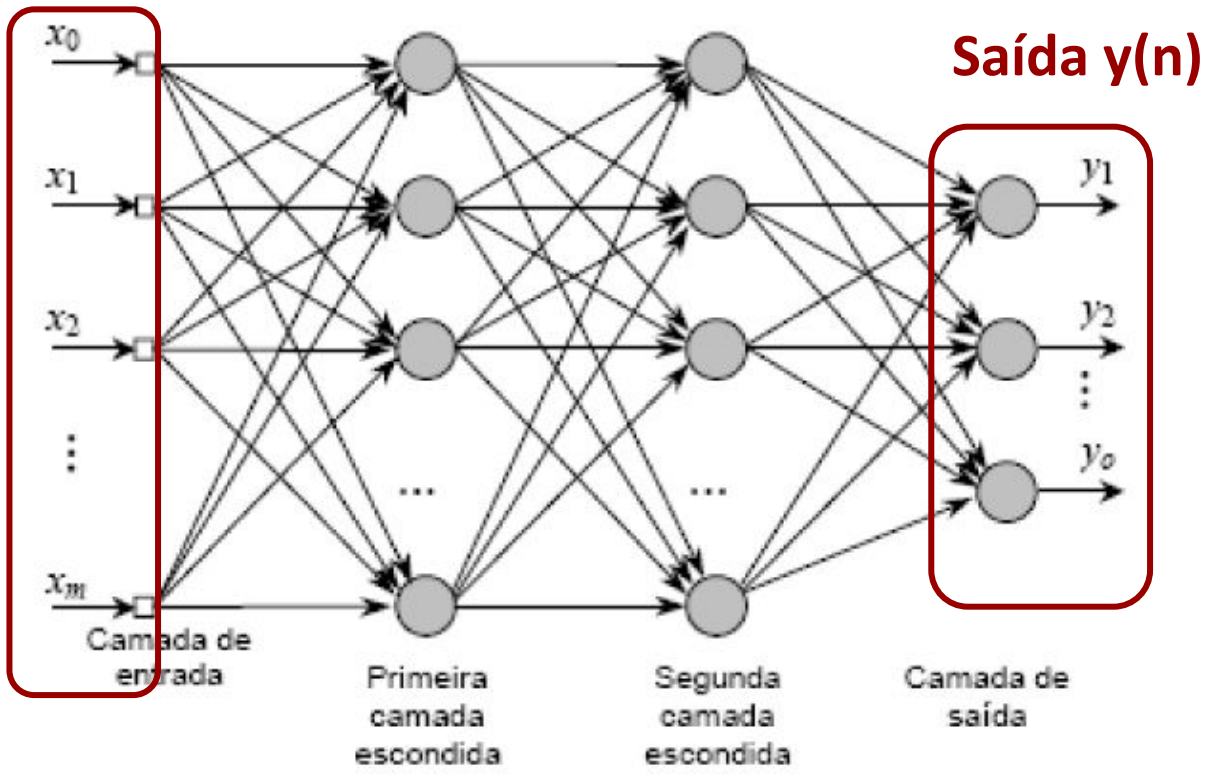


Figura 1 – Rede Neural Artificial Multicamadas.

Treinamento de Redes de Múltiplas Camadas

- **Problema 1: Sinal de erro só pode ser calculado diretamente nos neurônios da camada de saída**

Entrada x(n)

Saída y(n)

Base de dados:
[x(1), d(1)]
[x(2), d(2)]
...
[x(m), d(m)]

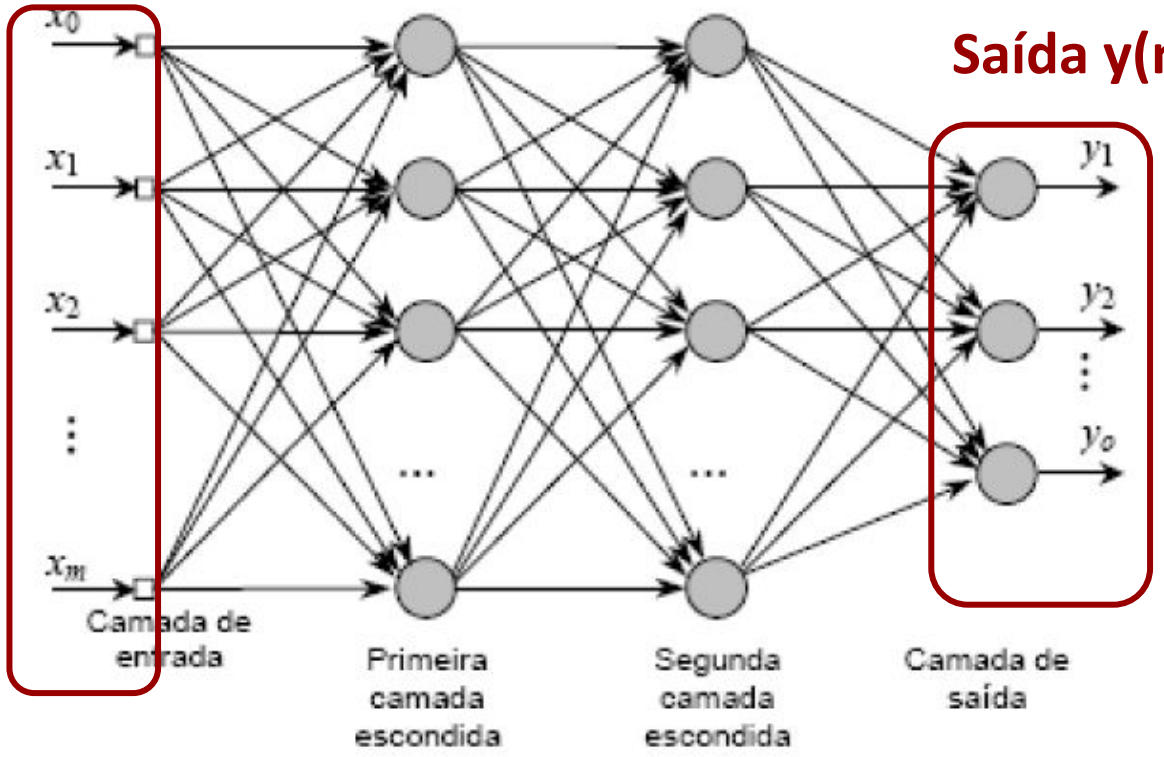


Figura 1 – Rede Neural Artificial Multicamadas.

Treinamento de Redes de Múltiplas Camadas

- **Então, como calcular o erro e ajustar os pesos sinápticos das camadas anteriores (ex: camadas ocultas)?**

Base de dados:
[x(1), d(1)]
[x(2), d(2)]
...
[x(m), d(m)]

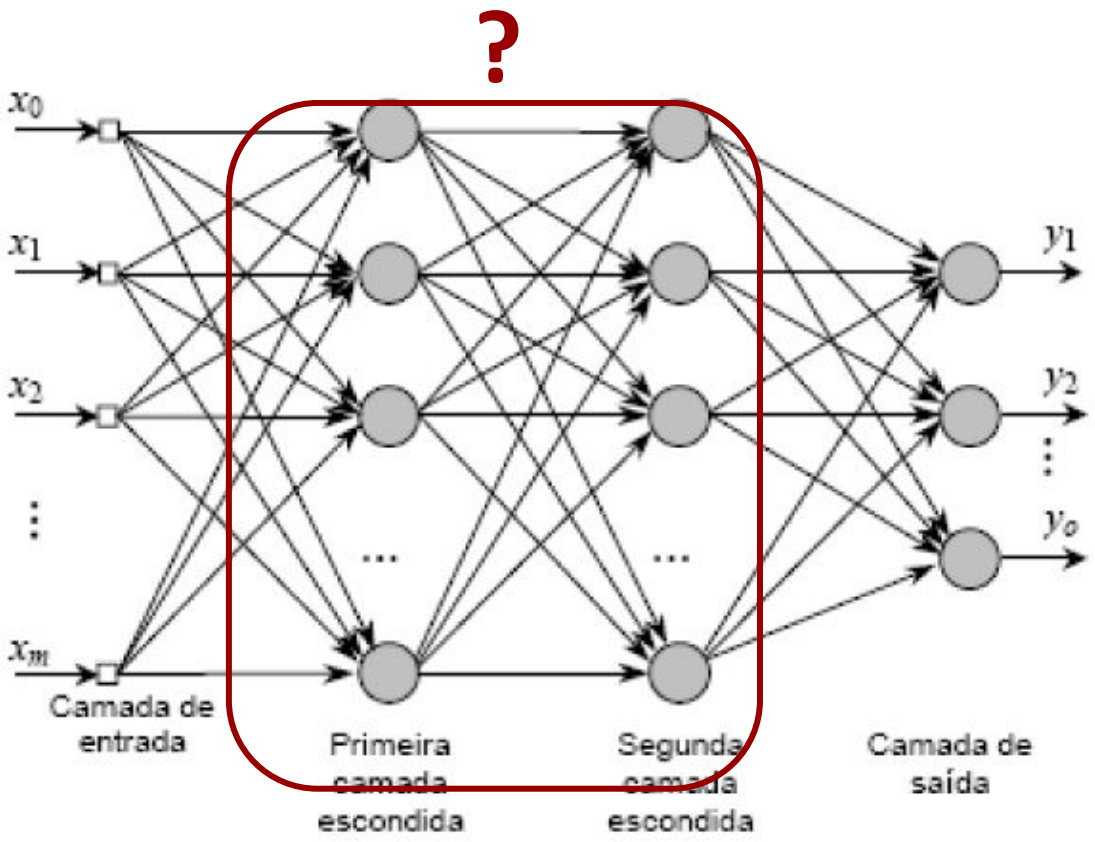
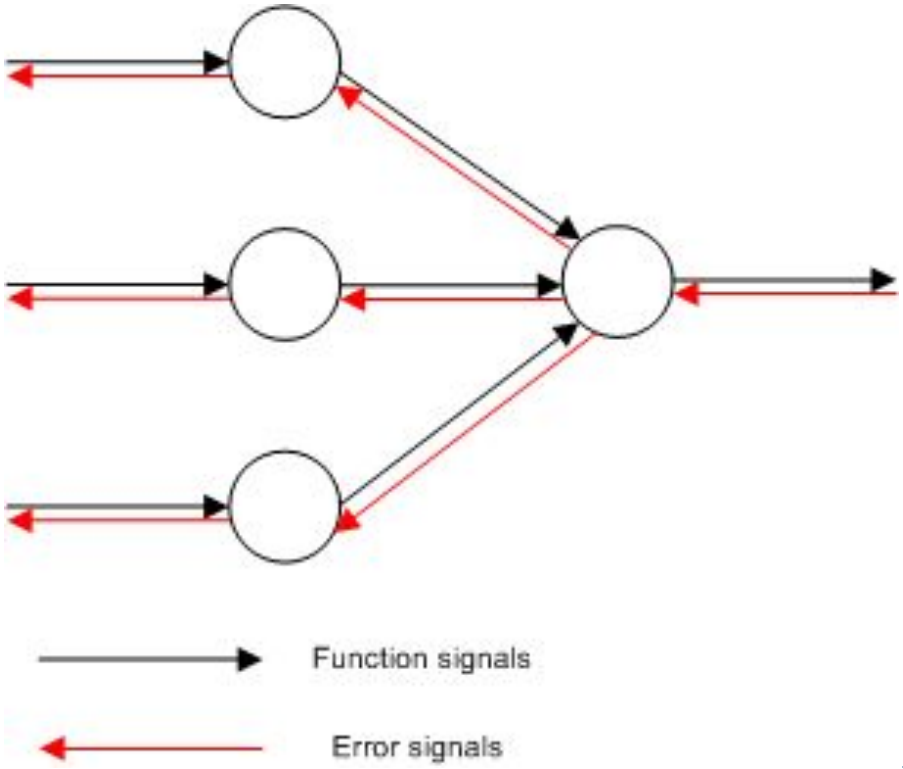


Figura 1 – Rede Neural Artificial Multicamadas.

Treinamento de Redes de Múltiplas Camadas

- **Solução: Algoritmo da Retropropagação (Backpropag)**
  - Sinal de erro se origina em um neurônio de saída e se propaga para trás (camada por camada) através da rede



BackPropagation - Regra Delta

- Correção dos pesos é definida pela **regra delta**:

$$\Delta w_{ji}(n) = \eta * \delta j(n) * y_i(n)$$

Correção de peso

Taxa de aprendizagem

Gradiente local

Sinal de entrada do neurônio *j*

- ***n - corresponde a um exemplo do conjunto de treinamento***
- ***j - é o neurônio atual***
- ***i - é o neurônio da camada anterior***

Gradiente local varia dependendo se *j* é um neurônio oculto ou da camada de saída.

BackPropagation

- Gradiente local:
  a. **Neurônio da camada de saída:** Gradiente local é igual ao produto da derivada associada pelo sinal de erro.

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

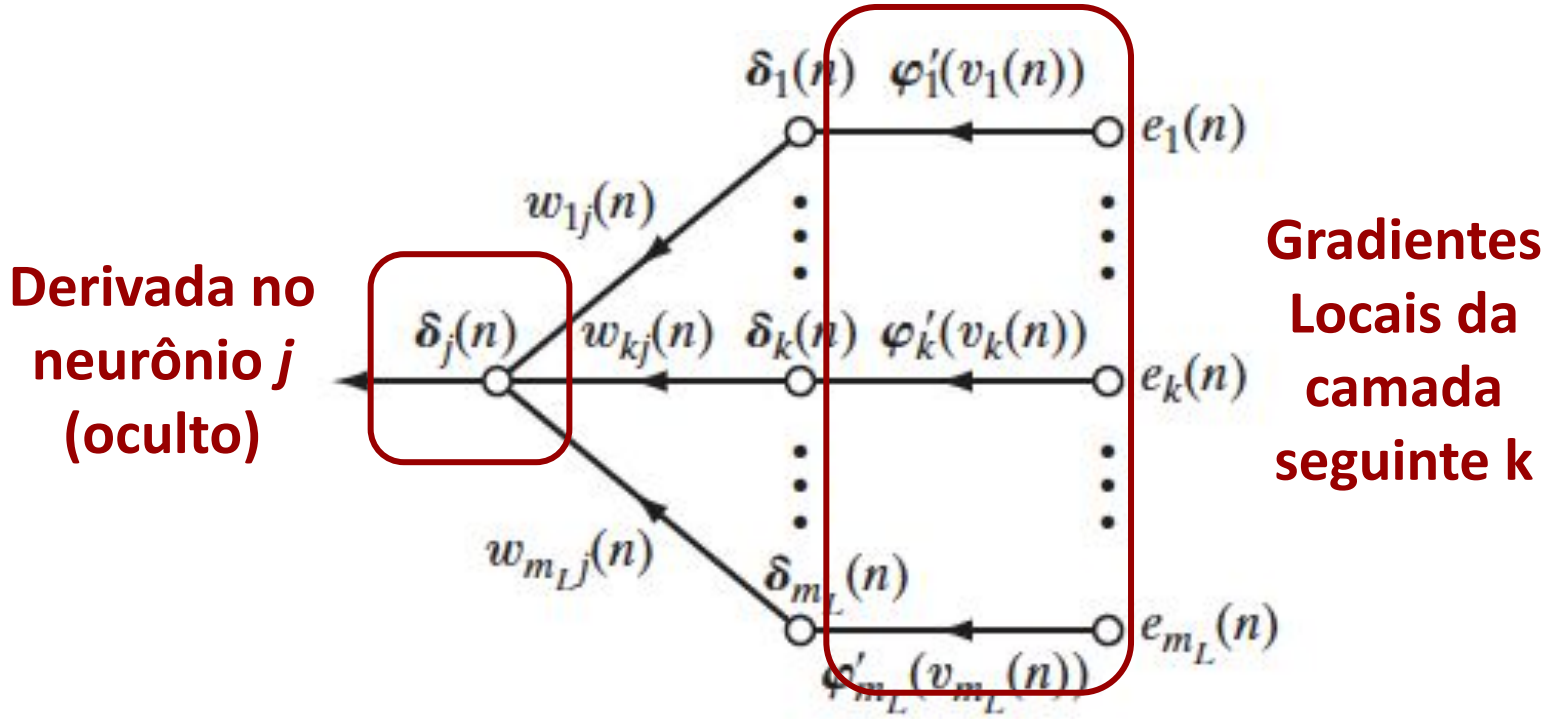
Gradiente Local

Sinal de erro na saída de *j*

Derivada da função de ativação no neurônio j

BackPropagation

- Gradiente local:
  a. **Neurônio da camada oculta:** Produto da **derivada** associada pela **soma ponderada dos gradientes locais** calculados dos neurônios da camada seguinte.



BackPropagation

- Gradiente local:
  a. **Neurônio da camada oculta:** Produto da derivada pelo somada ponderada dos gradientes locais calculados para os neurônios da próxima camada

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

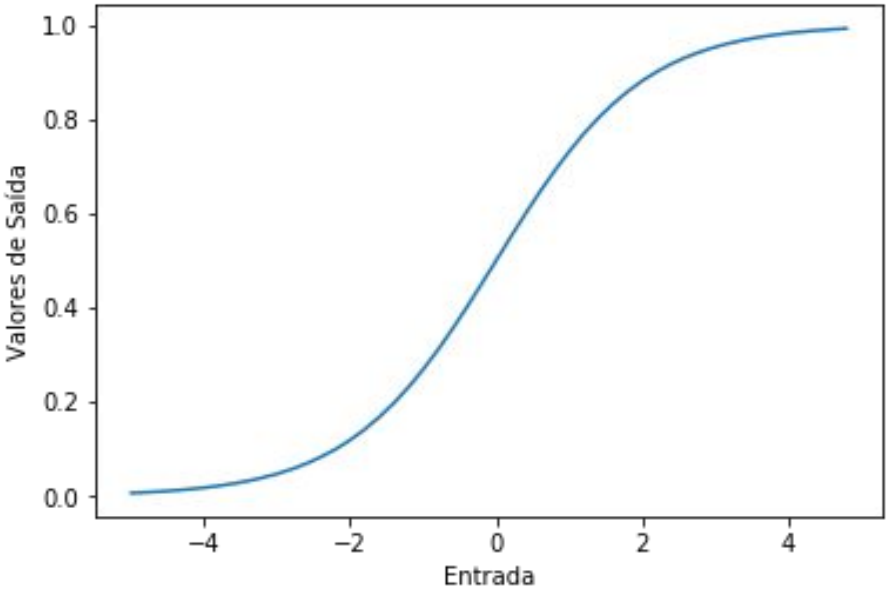
Gradiente Local

Derivada associada ao neurônio j

Soma ponderada dos gradientes locais da camada seguinte k

Funções de Ativação e suas Derivadas

- Sigmóide (Logística):



Função:

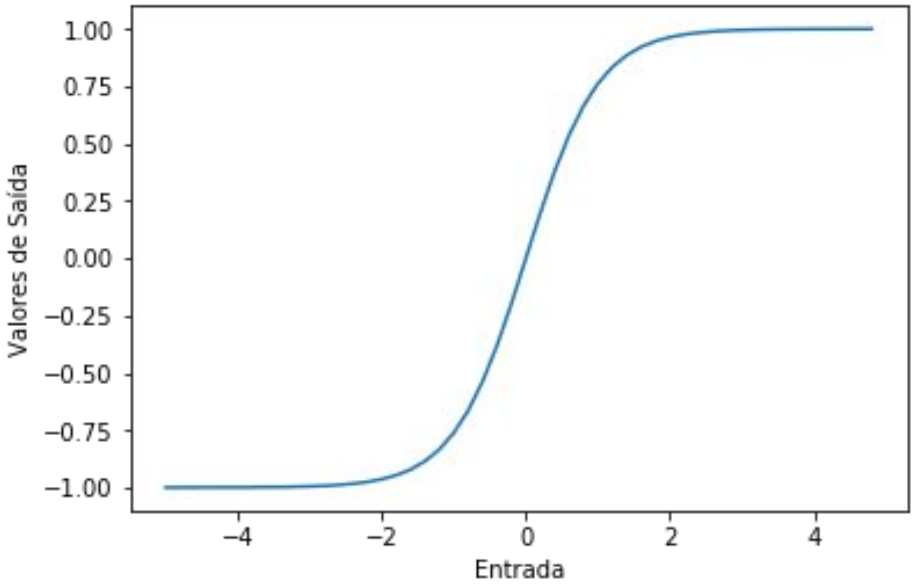
$$\varphi(x) = \frac{1}{1 + e^{-x}}$$

Derivada da Função:

$$\varphi'(x) = \varphi(x)(1 - \varphi(x))$$

Funções de Ativação e suas Derivadas

- Tangente Hiperbólica:



Função:

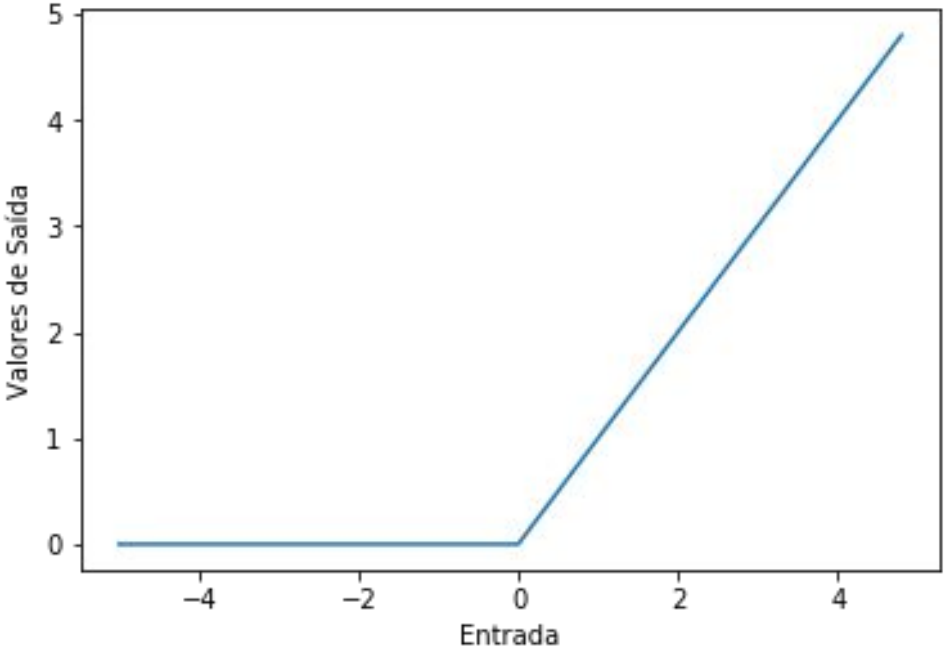
$$\varphi(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Derivada da Função:

$$\varphi'(x) = \frac{1}{2}(1 - \varphi^2(x))$$

Funções de Ativação e suas Derivadas

- ReLU:



Função:

$$\varphi(x) = \max(0, x)$$

Derivada da Função:

$$\varphi'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & c.c \end{cases}$$

Obs: Maior parte das redes DL usam ReLU nas camadas ocultas, porque o treinamento fica mais rápido, e atenua o problema do desaparecimento do gradiente (gradiente ≃0)

Algoritmo da Retropropagação (Backpropagation)

```
function backpropagation-algorithm
    (network, training_examples, learning_rate)
    network <- initialize_weights(randomly)
    start loop
        for each example in training-examples do
            // Computação para frente (Propagação)
            network_out = rna_output(network, example)

            // Computacao para trás (Retropropagação)
            example_err = actual_out - network_out
            local_gradient = calc_gradient(example_err)
```

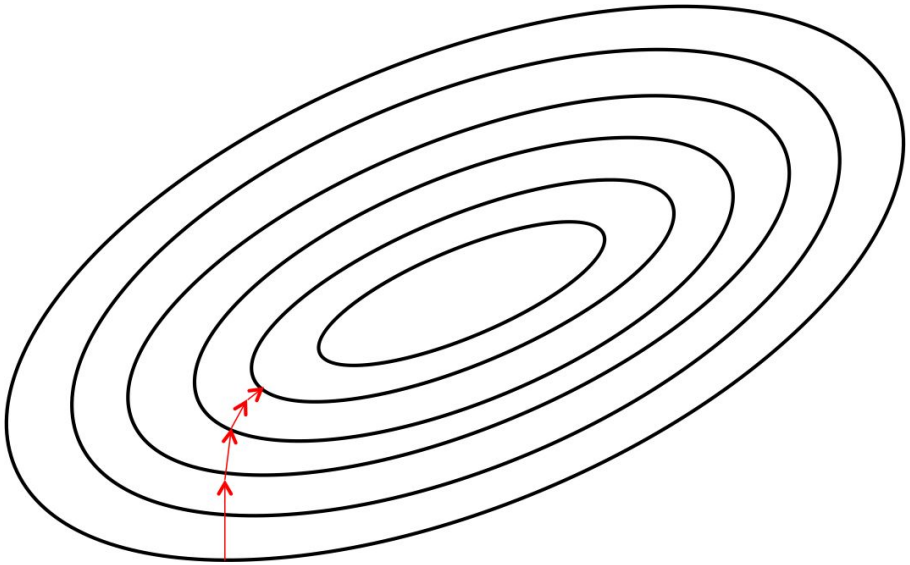
Algoritmo da Retropropagação (Backpropagation)

```
        Atualiza os pesos dos neurônios da camada de
        saída j (regra delta)

        for each previous-layer in network do
            Compute o erro em cada nó
            Atualiza os pesos dos neurônios da camada
            (regra delta)
        end for

    end for
end loop quando rede convergir
return network
```

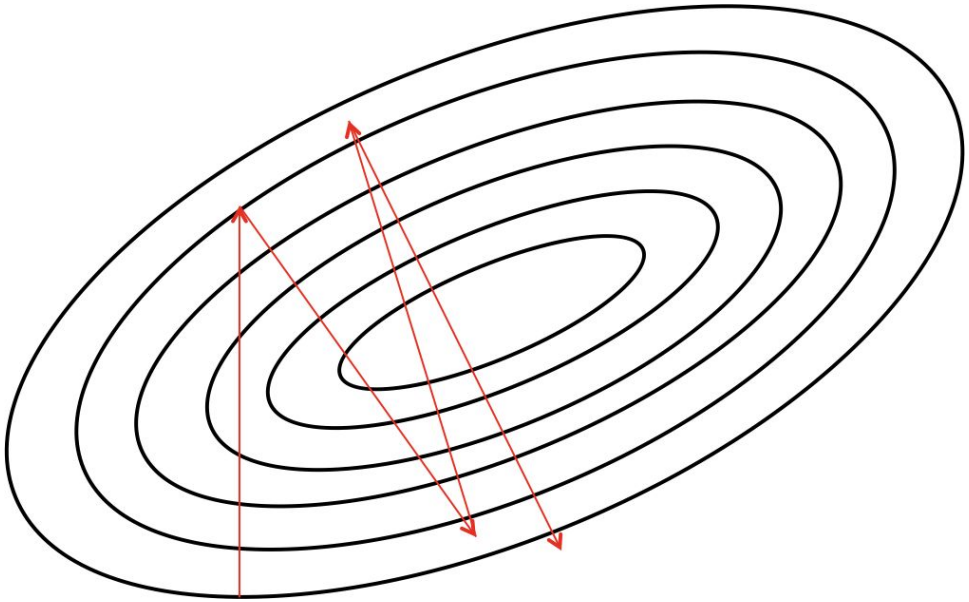
Taxa de Aprendizagem

Taxa de Aprendizagem (η)

η muito pequeno
Aprendizagem e convergência lenta



η muito grande
Rede instável



Regra Delta com Termo do Momento

- Permite aumentar a η, regulando o risco de instabilidade
  - Leva em conta a **variação de pesos da iteração anterior**.

$$\Delta w_{ji}(n) = \alpha\, \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

Constante do momento

Variação de pesos da última iteração

Regra delta

$$0 \leq \alpha < 1$$

Regra Delta com Termo do Momento

- Considerando duas iterações consecutivas:
  - **Variação de pesos tem o mesmo sinal**: Acelera a descida;
  - **Variação de pesos têm sinais opostos:** Reduz o ajuste nos pesos (estabiliza o algoritmo).

- Reduz a chance do processo de aprendizagem terminar em um mínimo local raso.

$$\Delta w_{ji}(n) = \alpha\, \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$$

Taxa de Aprendizagem Variável (Regra Delta-Bar-Delta)

- Em vez de utilizar um $\boldsymbol{\eta}$ constante, pode-se utilizar um $\boldsymbol{\eta}_{ij}$ **variável**, **dependente da conexão (peso sináptico).**

- Permite, por exemplo que alguns pesos da rede sejam ajustáveis ($\boldsymbol{\eta}_{ij} \neq 0$), enquanto outros ficam fixos ($\boldsymbol{\eta}_{ij} = 0$).

Playground Tensorflow

- Aplicação visual que ajuda a visualizar como funciona o treinamento de uma RNA
- https://playground.tensorflow.org

Modos de Treinamento: Estocástico, por Lote e Mini-Lote

Modos de Treinamento

- Para que a rede aprenda, tipicamente é necessário **apresentar o conjunto de treinamento (CT) MUITAS VEZES à rede;**

- **ÉPOCA**: Uma apresentação completa do CT à rede**.**

- Processo de aprendizagem é mantido de **ÉPOCA em ÉPOCA** até os pesos estabilizarem e o erro médio convergir para um valor mínimo.

Modos de Treinamento

- Baseado na forma de atualização dos pesos, temos três diferentes modos de treinamento:
  - **Estocástico (*Stochastic* ou Sequencial)**: Ajuste dos pesos é realizado após a apresentação de cada exemplo;
  - **Por Lote (*Batch*):** Ajuste dos pesos é realizado após a apresentação de todos os exemplos à rede (fim da época);
  - **Por Mini-Lote (*Mini-Batch*):** Ajuste de pesos é realizado após a apresentação de um subconjunto de exemplos;

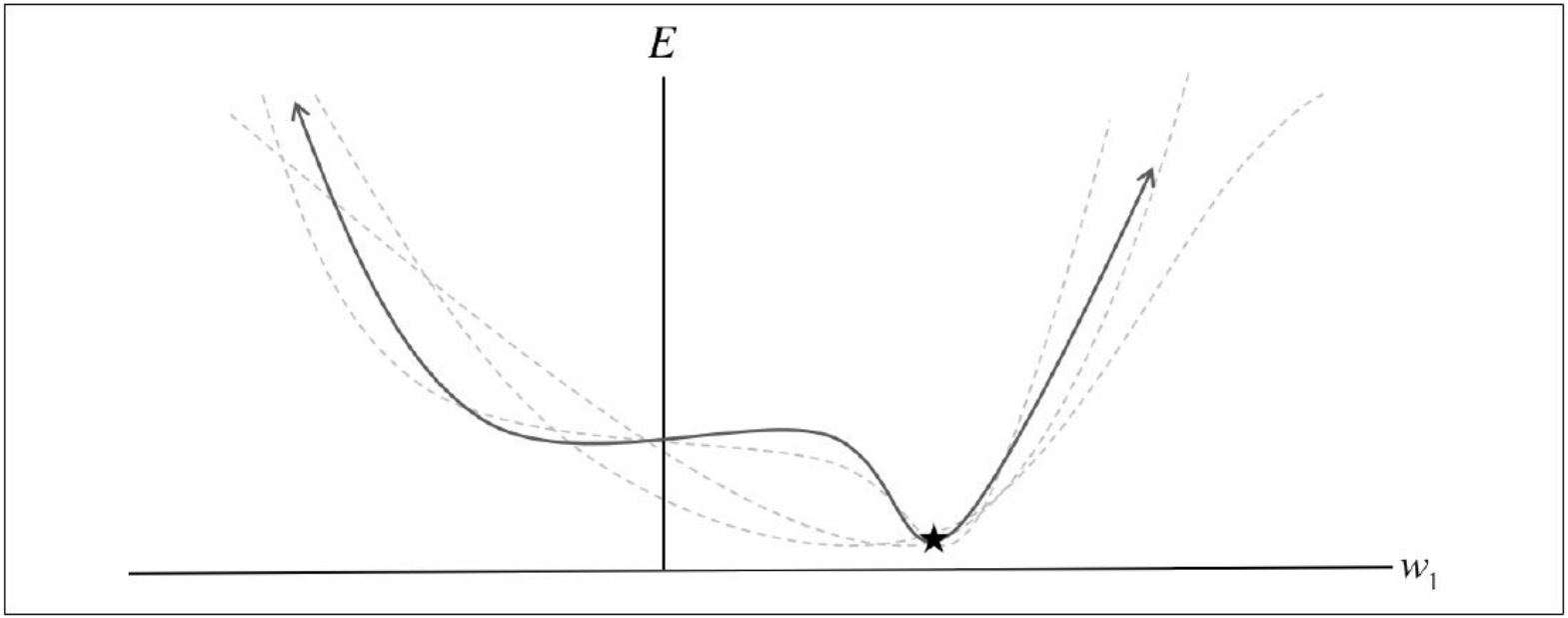
Modo de Treinamento Estocástico (Online ou Sequencial)

- Chamado também de **Gradiente de Descida Estocástico**
  - Ajuste de pesos é realizado de exemplo em exemplo;
  - Recomendável que os **exemplos sejam apresentados em ordem aleatória** de época em época.

- Processo de aprendizagem torna-se estocástico (**não determinístico)**;

Treinamento Estocástico

- **Vantagens:** Superfície de erro é estimada com relação a um único exemplo de treinamento (superfície dinâmica);
  - Ajuda a escapar de mínimos locais;
  - Mais simples de implementar;
  - Pode tirar vantagem de **dados (exemplos) redundantes**;



Treinamento Estocástico

- **Desvantagens:** Estimar o erro baseado em um único exemplo **pode não ser uma boa aproximação da superfície de erro real**.
  - Pode resultar num **treinamento muito lento**.
  - Mais difícil **provar teoricamente** que o **algoritmo converge**;

Treinamento por Lote (Batch)

- Ajuste dos pesos é realizado após a **apresentação de todos os exemplos de treinamento** (época).

- Função do erro médio quadrático:

$$E_{med} = \frac{1}{2N} \sum_{n=1}^{N} \sum_{j \in C} e_j^2(n)$$

Exemplo *n*

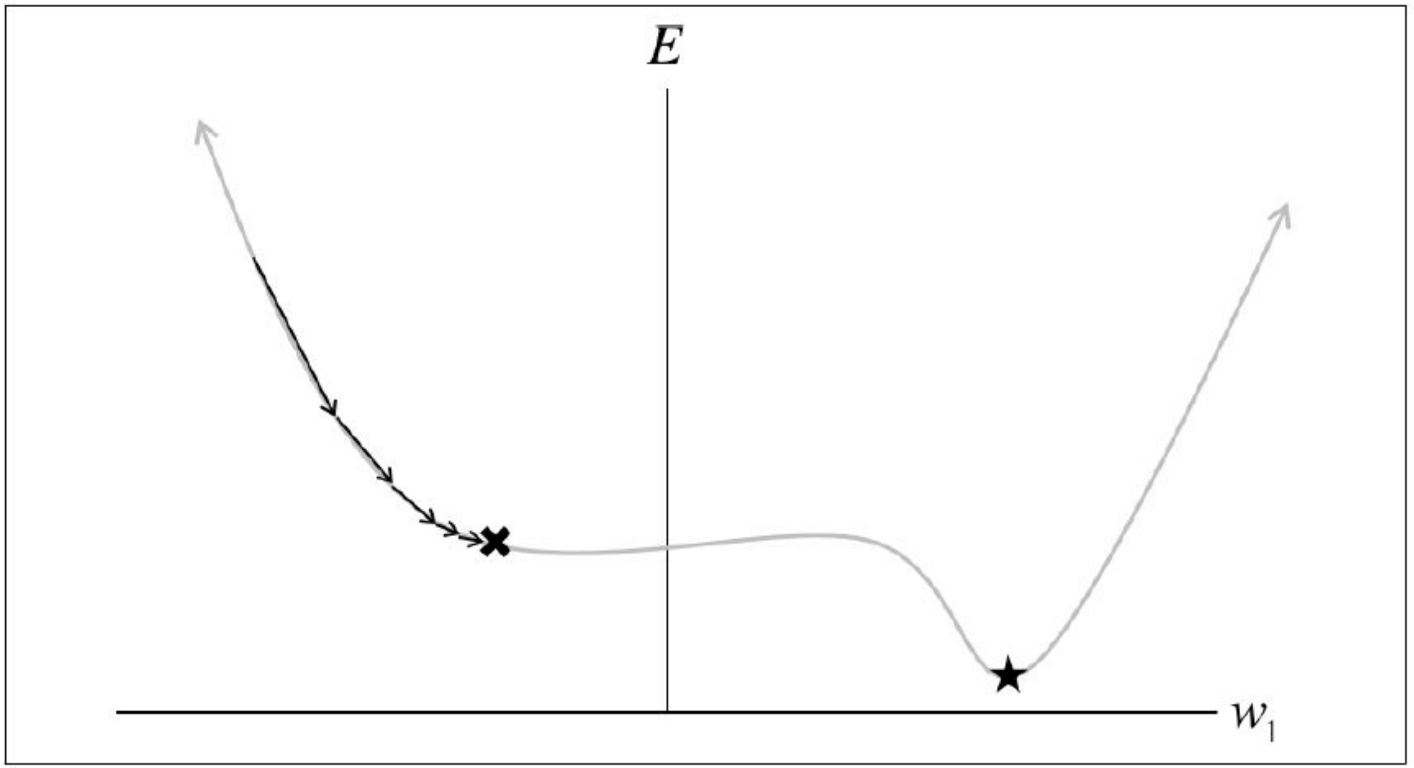
Erro em cada neurônio j da camada de saída

Treinamento por Lote (Batch)

- **Vantagens:**
  - Estimativa precisa do gradiente;
  - Convergência mais rápida sob condições simples;
  - Mais fácil de paralelizar;

Treinamento por Lote (Batch)

- **Desvantagens**:
  - Pode ficar presos em mínimos locais ou pontos de sela na superfície de erro;



Treinamento por Mini-Lote (Mini-Batch)

- Ajustes de pesos é realizado após a apresentação de um subconjunto de exemplos;

- Bom balanço entre **modo estocástico** e **modo por lote:**
  - **Convergência mais rápida (modo por lote);**
  - **Evita mínimos locais (modo estocástico);**

Critérios de Parada

Critérios de Parada

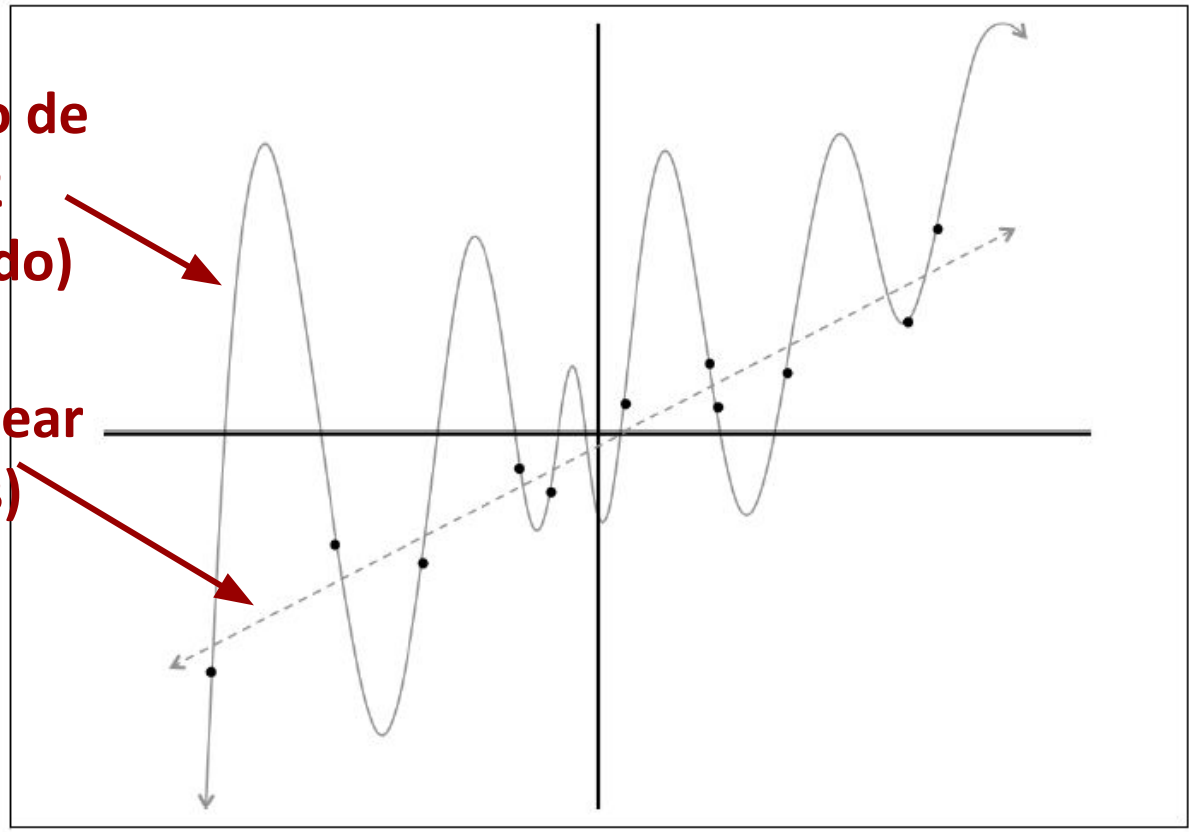
- Em geral, não existem critérios bem definidos para encerrar o algoritmo da retropropagação.
  - Não se pode demonstrar que o algoritmo convergiu.

- Existem alguns critérios razoáveis para a convergência:
  - Vetor gradiente alcançar um limiar suficiente pequeno;
  - Taxa de variação do erro (médio quadrático) muito pequena entre épocas (ex: 0,1%, 0,01%, 1%);
  - **Rede apresenta um bom desempenho de generalização (i.e, funciona bem com um outro conjunto de exemplos - conjunto de validação)**

Generalização

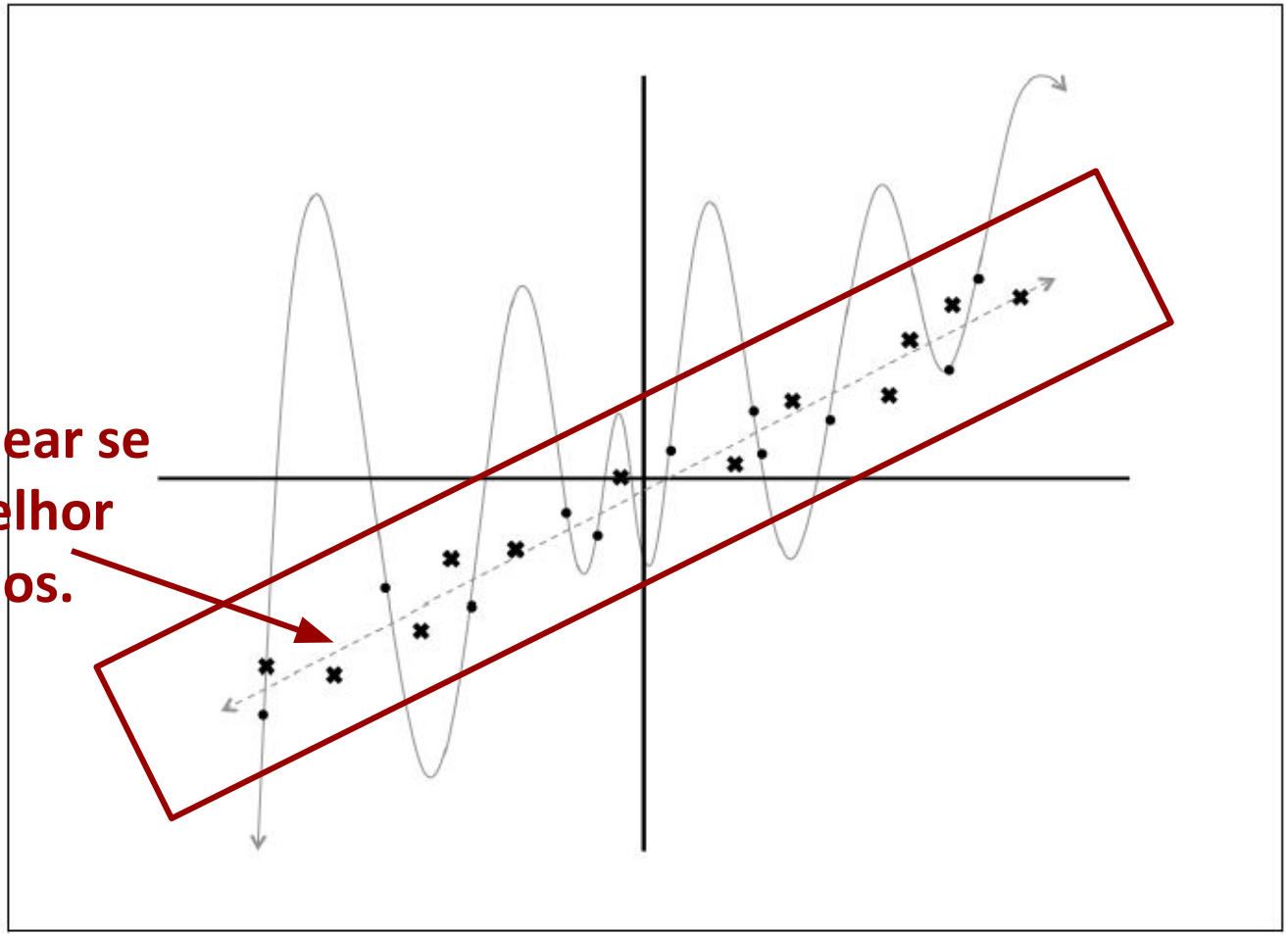
Generalização

- Ex: Qual é a curva (modelo) que melhor descreve o conjunto de dados abaixo?



Generalização

- Vamos adicionar novos dados para avaliar...



Modelo linear se ajusta melhor aos dados.

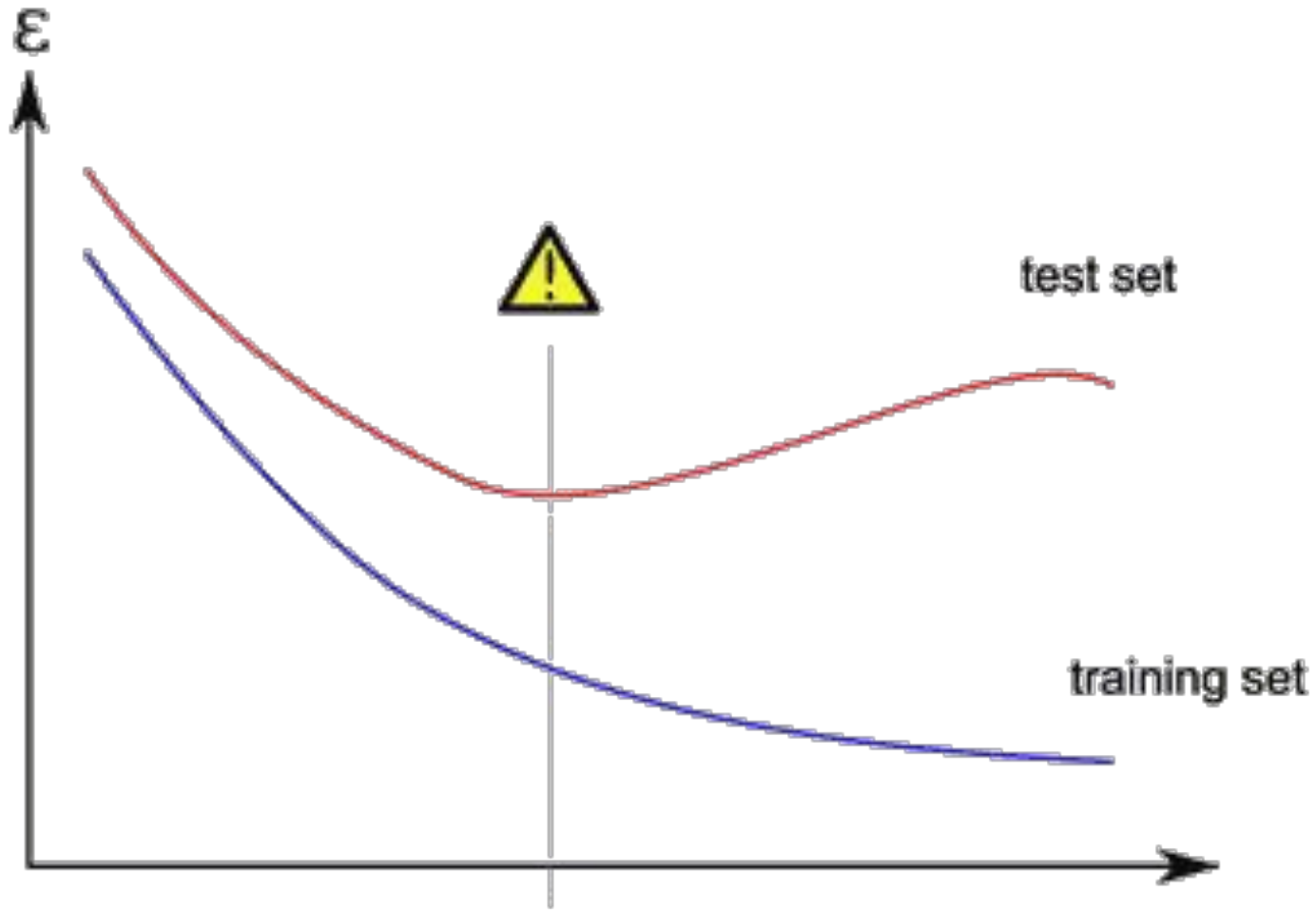
Generalização

- Uma RNA **GENERALIZA BEM** quando ela tem um bom desempenho para **dados não utilizados no treinamento**.

- No exemplo, o modelo mais complexo (Polinômio de Grau 12) se ajustava melhor aos dados de treinamento mas **não generalizava bem**.
  - Teve um desempenho pobre com os novos dados.

- Esse fenômeno é chamado de **Overfitting**
  - Excesso de ajuste ou excesso de treinamento.

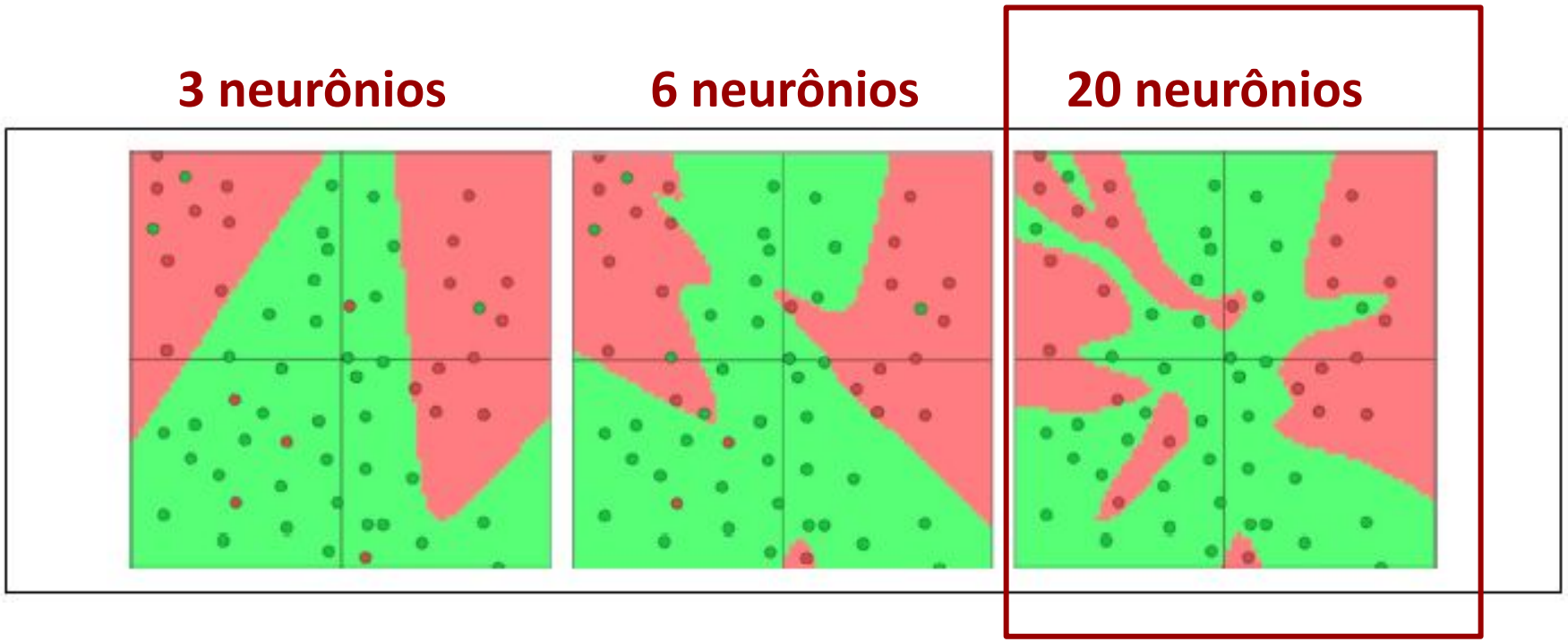
Overfitting



Overfitting

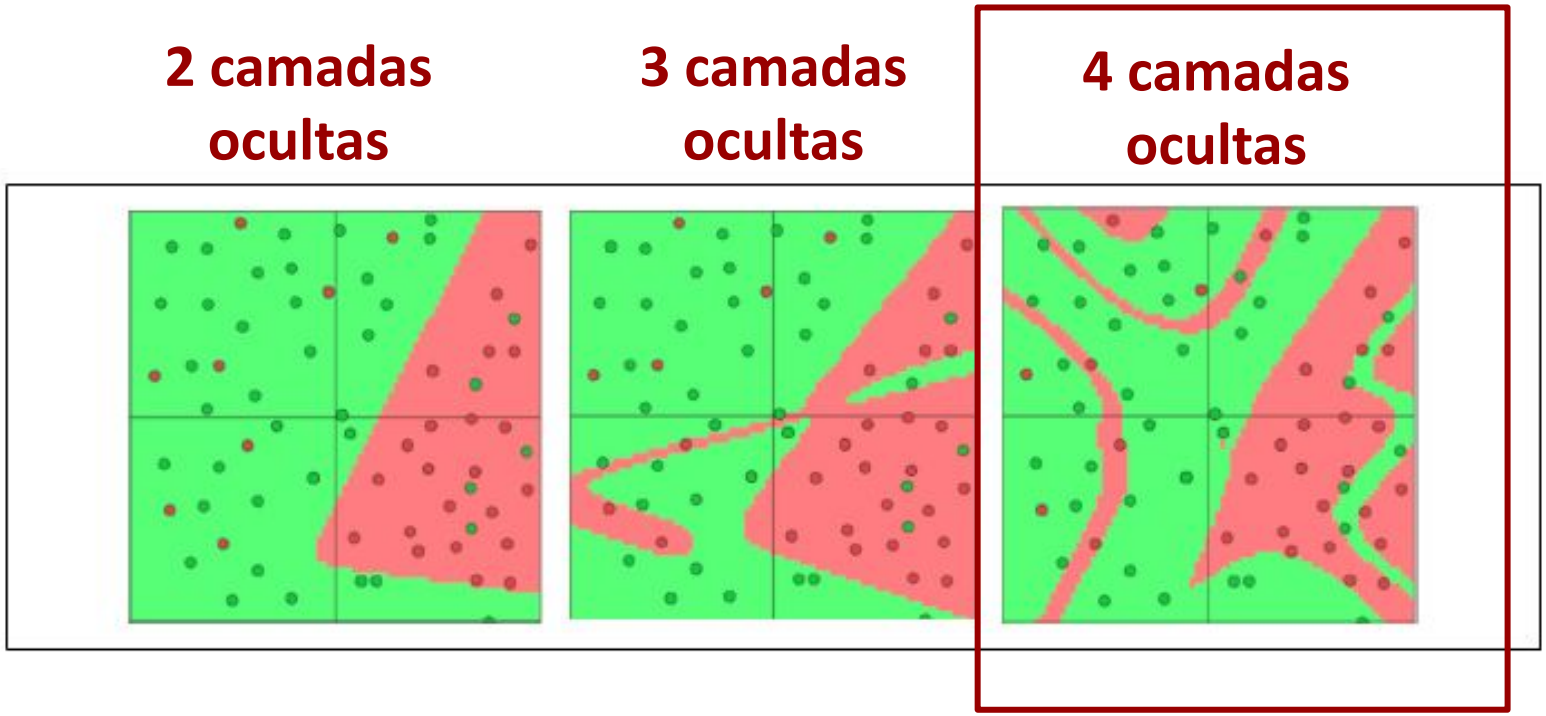
- Aumento no número de conexões implica em maior propensão a **overfitting.**

- Ex: RNA formada por 1 camada oculta contendo:



Overffiting

- Aumento no número de camadas também aumenta a propensão a **Overfitting**.



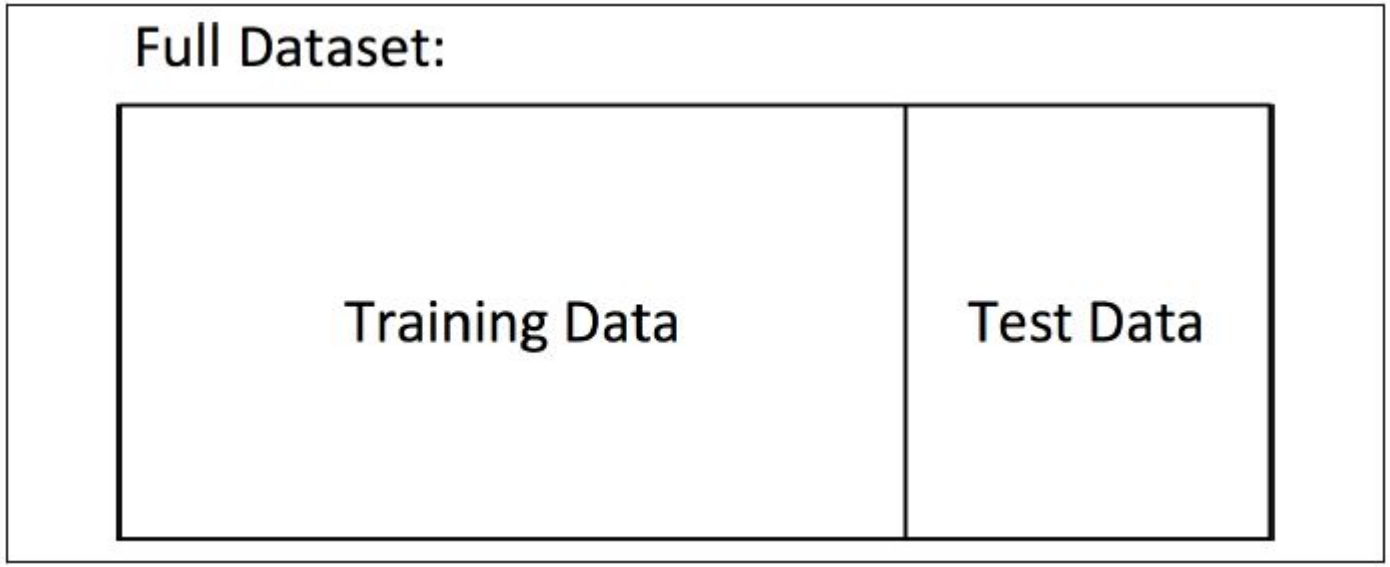
Overfitting

- Overfitting é um problema comum em DL.

- DL procura resolver **problemas muito complexos** com **modelos complexos**
  - Necessário tomar medidas para prevenir o **overfitting**.
  - Veremos algumas dessas medidas

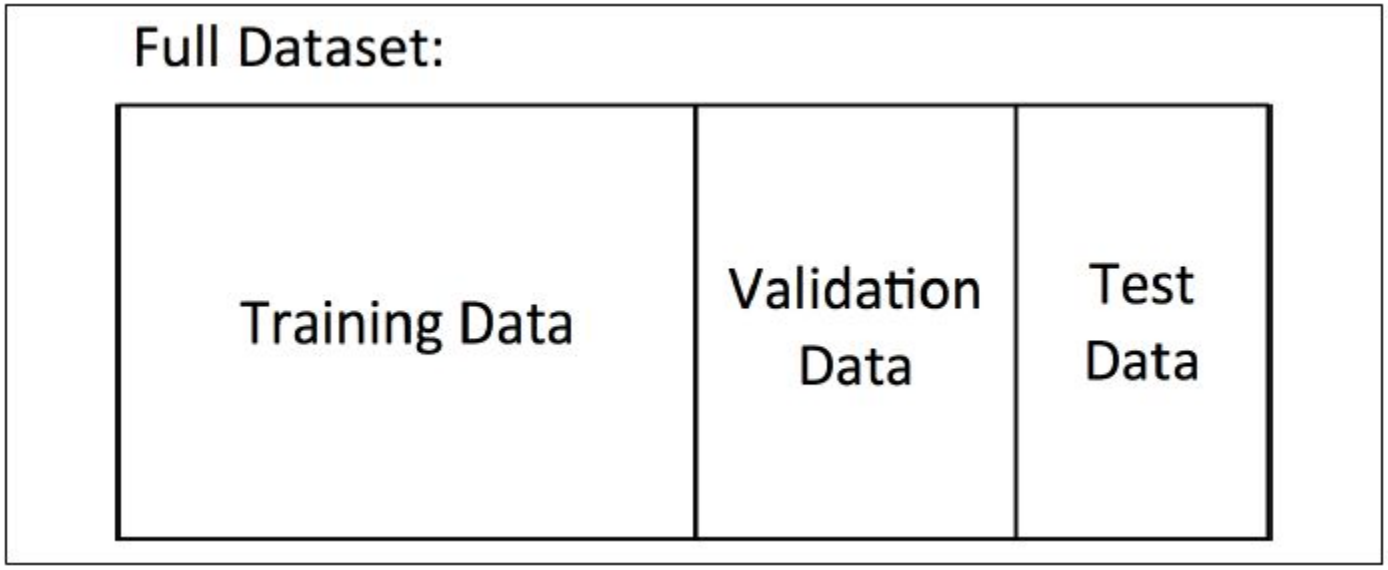
Prevenção de Overfitting

- **Medida 1: Não avalie seu modelo com os dados que você usou para treinar.**
  - Divida seu conjunto em conjunto de treinamento e conjunto de testes.



Prevenção de Overfitting

- **Medida 2: Crie também um conjunto de validação.**
  - Use-o para avaliar o treinamento de época em época;
  - Se o desempenho continuar **melhorando no treinamento** mas **piorando no conjunto de validação**, é hora de parar.
    - **Você pode estar em OVERFITTING.**



Regularização em DL
(Prevenção de Overfitting)

Regularização em DL

- Medida para proteger contra Overfitting;

- Modifica a função objetivo (função de erro) adicionando termos que **penalizam pesos grandes**.

$$\text{Erro} = Erro + \lambda f(\mathbf{w})$$

Força da regularização

***f*(w) cresce a medida que w cresce**

- **λ=0, sem proteção contra overfitting,**
- **λ muito grande, prioriza manter os pesos pequenos.**

Regularização L2

- **Regularização L2:** Aumenta a função de erro com o quadrado de todos os pesos da rede;
  - Para cada peso w, adiciona $0.5\lambda w^2$ a função de erro.

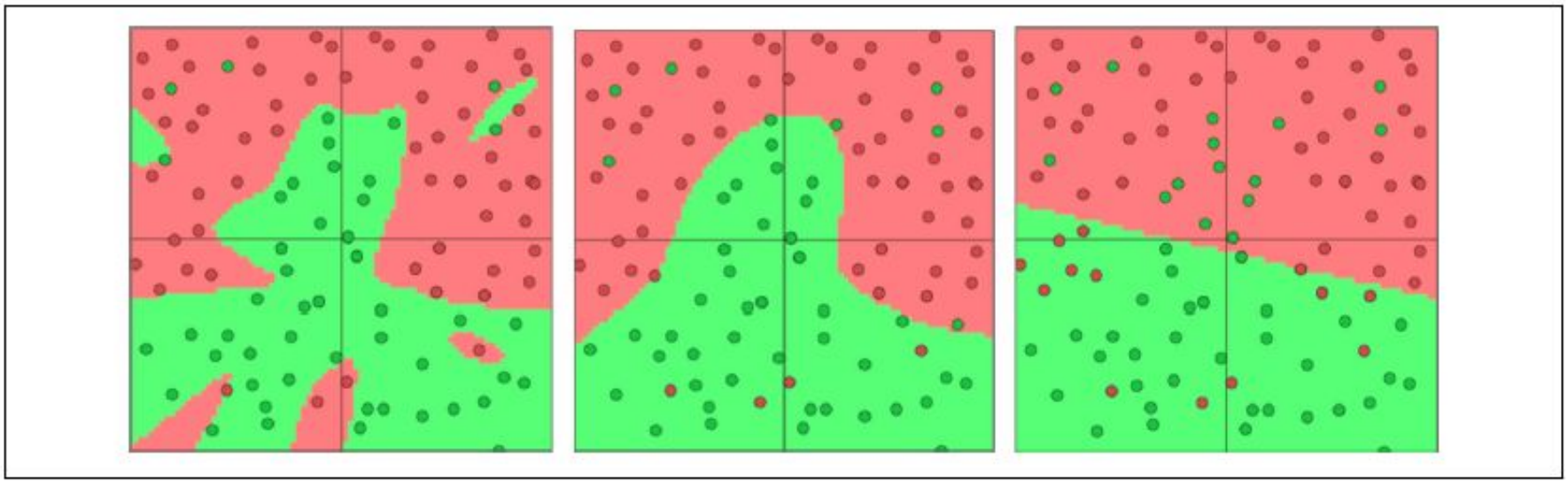
- Penaliza fortemente os pesos grandes, preferindo pesos difusos.

- Decaimento dos pesos: Pesos vão caindo para zero durante a descida do gradiente.

Regularização L2

- Exemplo:

λ=0,01 λ=0,1 λ=1



Regularização L1

- **Regularização L1:** Para cada peso w, adiciona $\lambda|w|$ a função de erro.

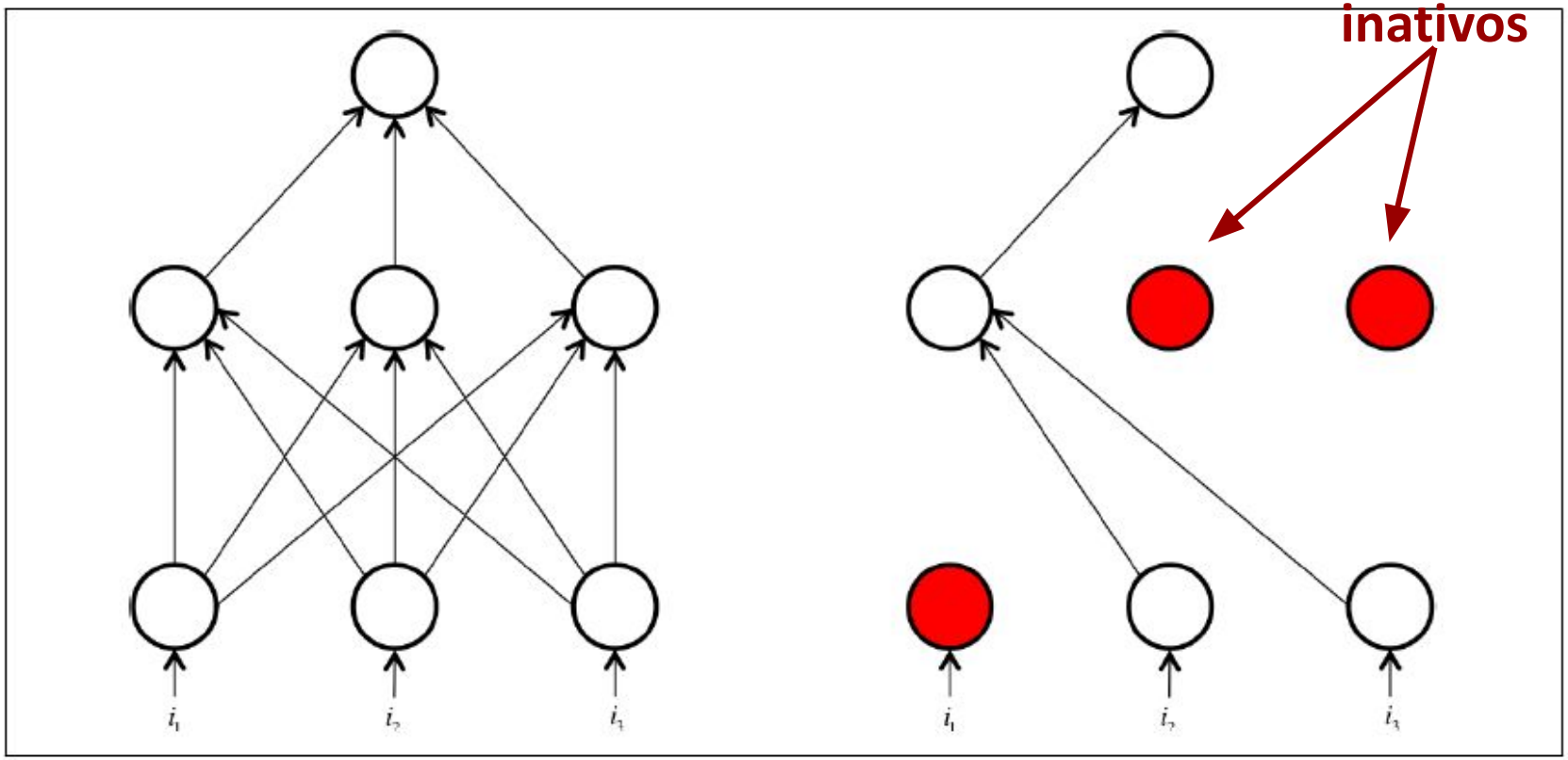
- Torna o vetor de pesos esparso (muitos zeros);
  - Apenas as entradas mais importantes serão consideradas (ie, com pesos diferente de zero);
  - Resistência a ruídos na entrada;

- Útil quando se pretende entender que features contribuem mais para uma decisão.
  - Se esse tipo de análise de características não é necessário, usar L2 porque tem um melhor desempenho.

Dropouts
(Prevenção de Overfitting)

Dropouts

- Durante o treinamento, mantém um neurônio ativo com uma probabilidade *p*.



Dropouts

- **Uma das estratégias preferidas, em DL, para prevenir overfitting.**

- Força a rede a ter uma boa acurácia mesmo na ausência de certas informações.

- Evita que a rede se torne muito dependente de um (ou um conjunto de) neurônios.

Universidade Federal da Paraíba

Centro de Informática

Departamento de Informática

Aprendizado Profundo

Treinamento de RNAs e Algoritmo da Retropropagação

Leonardo Vidal

Thais Gaudencio

Tiago Maritan