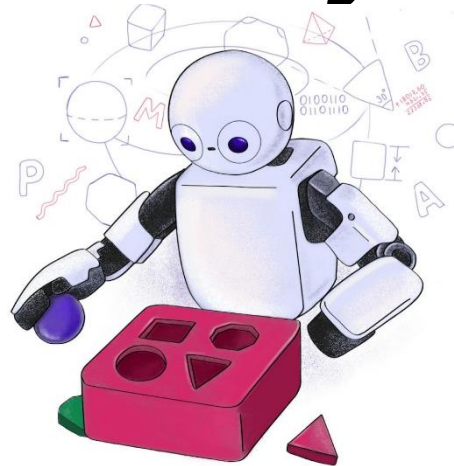


# T320 - Introdução ao Aprendizado de Máquina II: *Redes Neurais Artificiais (Parte III)*



***Inatel***

Felipe Augusto Pereira de Figueiredo  
felipe.figueiredo@inatel.br

# Recapitulando

- Na última aula, fomos apresentados às redes neurais.
- Vimos que elas são formadas por camadas de perceptrons que se conectam através dos pesos sinápticos.
- Aprendemos que as funções de ativação logística e tangente hiperbólica causam o problema do desaparecimento do gradiente, o qual pode ser solucionado usando-se a função retificadora.
- Vimos algumas topologias diferentes de redes neurais.
- E aprendemos que as redes neurais são aproximadoras universais de funções.
- Nesta aula, veremos como as redes neurais aprendem, ou seja, são treinadas.

# Aprendizado em Redes Neurais

- Consideramos agora, o **processo de otimização**, ou seja, de **atualização dos pesos sinápticos**.
- Assim como vimos anteriormente, o processo de otimização corresponde a um **problema de minimização** de uma **função custo (ou de perda),  $J(w)$** , com respeito a um vetor de pesos  $w$ .
- Portanto, o problema de aprendizado em redes neurais pode ser formulado como

$$\min_w J(w)$$

- Normalmente, esse processo de otimização é **conduzido de forma iterativa**, o que dá um **sentido mais natural à noção de aprendizado** (i.e., um processo gradual).
- Existem **vários métodos de otimização** aplicáveis, mas, sem dúvida, **os mais utilizados são aqueles baseados nas derivadas da função custo,  $J(w)$** .

# Aprendizado em Redes Neurais

- Dentre esses métodos, existem os de ***primeira ordem*** e os de ***segunda ordem***.
- Os métodos de ***primeira ordem*** são baseados nas derivadas parciais de primeira ordem da ***função custo***, agrupadas no ***vetor gradiente***:

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \frac{\partial J(\mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_K} \end{bmatrix}$$

- Como já vimos, o ***gradiente aponta na direção de maior crescimento da função*** e portanto, ***caminhar em sentido contrário*** a ele é uma forma adequada de se ***buscar iterativamente a minimização da função de custo***.

# Aprendizado em Redes Neurais

- Desta maneira, temos a seguinte **equação de atualização dos pesos**

$$\mathbf{w}(k + 1) \leftarrow \mathbf{w}(k) - \alpha \nabla J(\mathbf{w}(k)),$$

onde  $\alpha$  é o **passo de aprendizagem** e  $k$  é a iteração de atualização.

- Já os métodos de **segunda ordem**, são baseados na informação trazida pela **derivada parcial de segunda ordem da função custo**. Essa informação está contida na **matriz Hessiana,  $H$** :

$$H(\mathbf{w}) = \nabla^2 J(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2 J(\mathbf{w})}{\partial w_1^2} & \frac{\partial^2 J(\mathbf{w})}{\partial w_1 \partial w_2} & \dots & \frac{\partial^2 J(\mathbf{w})}{\partial w_1 \partial w_K} \\ \frac{\partial^2 J(\mathbf{w})}{\partial w_2 \partial w_1} & \frac{\partial^2 J(\mathbf{w})}{\partial w_2^2} & \dots & \frac{\partial^2 J(\mathbf{w})}{\partial w_2 \partial w_K} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J(\mathbf{w})}{\partial w_K \partial w_1} & \frac{\partial^2 J(\mathbf{w})}{\partial w_K \partial w_2} & \dots & \frac{\partial^2 J(\mathbf{w})}{\partial w_K^2} \end{bmatrix}.$$

# Aprendizado em Redes Neurais

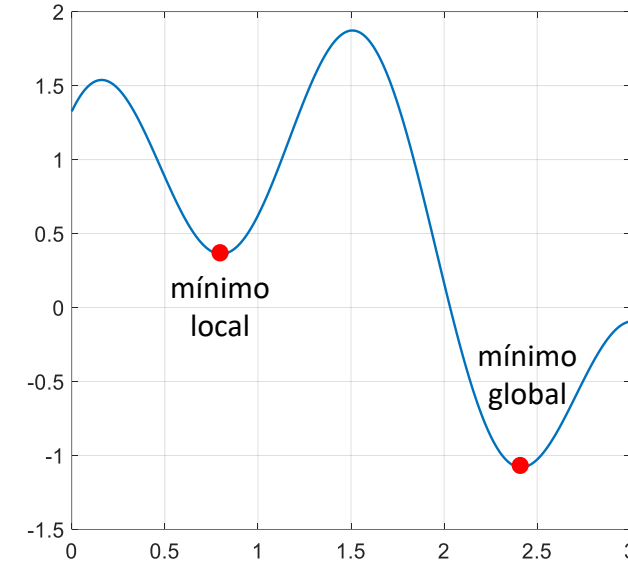
- De posse da **matriz Hessiana**, é possível fazer uma aproximação de Taylor de segunda ordem da **função de custo**, o que leva à seguinte expressão para adaptação dos pesos:

$$\mathbf{w}(k + 1) \leftarrow \mathbf{w}(k) - \alpha \mathbf{H}^{-1}(\mathbf{w}(k)) \nabla J(\mathbf{w}(k)).$$

- Essa expressão requer que a **matriz Hessiana** seja **inversível** e **definida positiva** a cada iteração,  $k$ , i.e.,  $\mathbf{z}^T \mathbf{H} \mathbf{z} > 0, \forall \mathbf{z} \neq \mathbf{0}$  (vetor nulo).
- A aproximação de Taylor com informação de segunda ordem é mais precisa que a fornecida por métodos de primeira ordem.
- Portanto, a tendência é que métodos de **segunda ordem** convirjam em menos passos que métodos de **primeira ordem**.
- Entretanto, o cálculo exato da **matriz Hessiana** pode ser complicado em vários casos práticos.
  - Por exemplo, se tivermos 10 pesos para otimizar, a matriz Hessiana teria 10x10 elementos. Portanto, essa abordagem direta não é eficiente se o número de pesos for muito grande.
- Porém, há um conjunto de métodos de segunda ordem que evitam esse cálculo direto, como os métodos **quasi-Newton** ou os métodos de **gradiente escalonado**, os quais aproximam a matriz Hessiana.

# Mínimos Locais, Globais, Pontos de Sela e Platôs

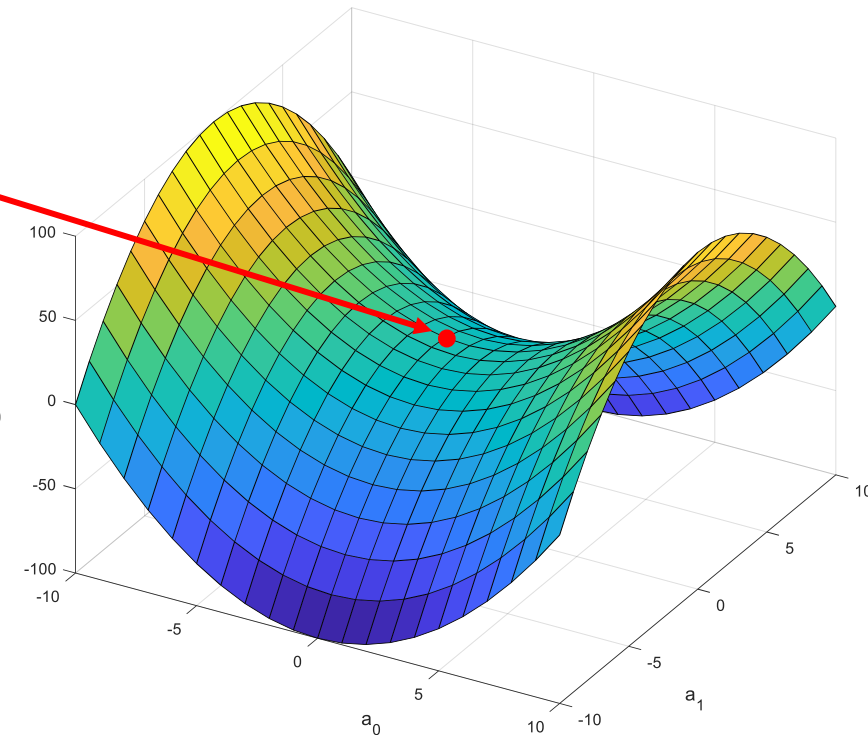
- É importante ressaltarmos que todos esses métodos são métodos de **busca local**, ou seja, eles têm **convergência assegurada para mínimos locais**.
- Um **mínimo** (local ou global) sempre atrai o vetor de pesos quando este se encontra em sua vizinhança.
- Para lembrarmos o que é um mínimo local, vejamos a figura ao lado onde existem dois mínimos:
  - Um deles é uma **solução ótima em relação apenas a seus vizinhos**, ou seja, um **mínimo local**.
  - O outro também é uma solução ótima em relação a seus vizinhos (**mínimo local**), mas também **em relação a todo o domínio da função de custo**. Este é um **mínimo global**.
- Por serem formadas pela combinação de vários nós com funções de ativação não-lineares, as superfícies de erro de redes neurais **não são convexas**, ou seja, são **altamente irregulares**, podendo conter vários mínimos locais.



**IMPORTANTE:** Para muitos problemas envolvendo redes neurais, quase todos os mínimos locais têm um valor muito semelhante ao do mínimo global e, portanto, encontrar um mínimo local já é bom o suficiente para um dada problema.

# Mínimos Locais, Globais, Pontos de Sela e Platôs

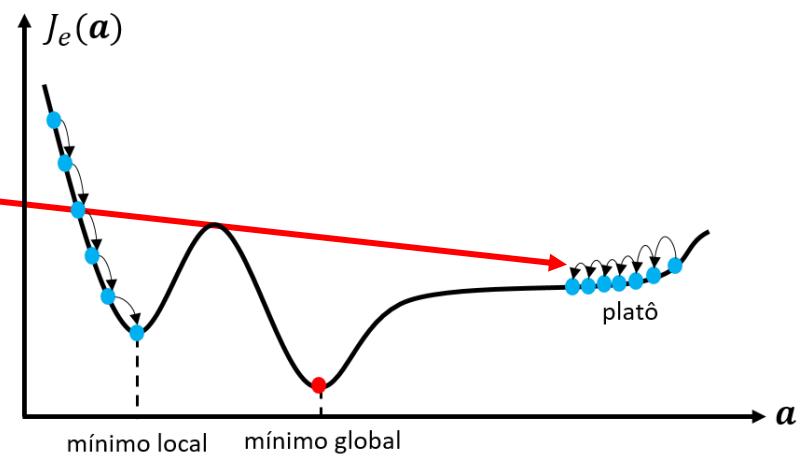
- Outra irregularidade que podemos encontrar são os chamados ***pontos de sela***:
  - Um ponto que é um mínimo ao longo de um eixo, mas um máximo ao longo de outro.
  - Em algumas direções são ***atratores*** (i.e., alta declividade), mas em outras não.
- O algoritmo de minimização da função de custo pode passar um longo período de tempo sendo atraído por eles, o que prejudica seu desempenho.
- Para escapar destes pontos, usa-se métodos de ***segunda ordem*** ou ***versões ruidosas do gradiente descendente***, como, por exemplo, o ***Gradiente Descendente Estocástico***.



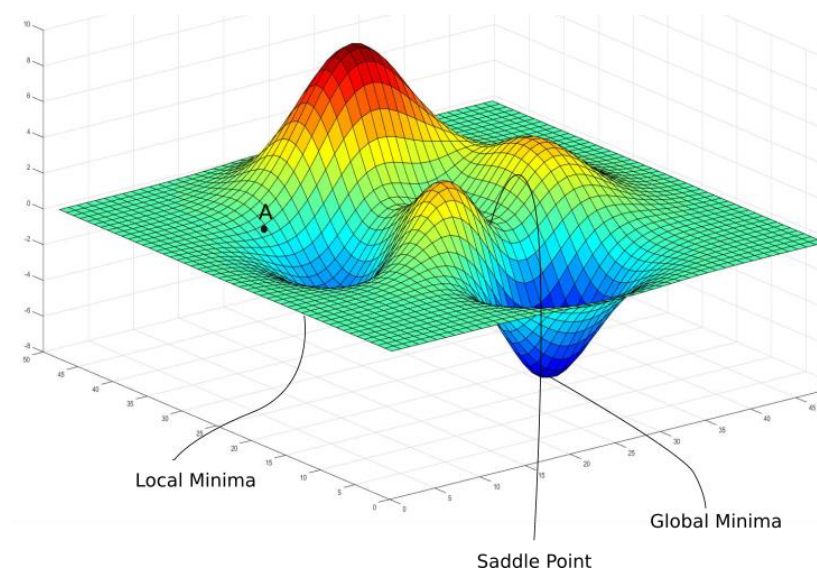


# Mínimos Locais, Globais, Pontos de Sela e Platôs

- Outro tipo de irregularidade são os **platôs**: regiões planas, mas com erro elevado.
  - Como a inclinação nesta região é próxima de zero (consequentemente o gradiente é próximo de zero) o algoritmo pode levar muito tempo para atravessá-la.
- Para se escapar destas regiões, usa-se métodos de **aprendizado adaptativo** como AdaGrad, RMSProp, Adam, etc.
- Portanto, como garantir que o mínimo encontrado é bom o suficiente?
  - Treina-se o modelo várias vezes, sempre inicializando os **pesos aleatoriamente**, com a esperança de que em alguma dessas vezes ele inicialize mais próximo do mínimo global ou de um bom mínimo local.



Exemplo da superfície de erro de uma rede neural




# Tarefa


- **Quiz:** “*T320 - Quiz – Redes Neurais Artificiais (Parte V)*” que se encontra no MS Teams.
- **Avaliação Presencial**
  - Data: 27/05/2023 às 08:00 na sala I-18.
  - Faremos apenas o exercício #1 do projeto #2.
- **Projeto #2**
  - Projeto já está no github, logo abaixo do lab. # 9, e pode ser feito em grupos de no máximo 3 alunos.
  - Entrega: **25/06/2023 até às 23:59.**
  - Leiam os enunciados atentamente.


Laboratório 9

 Launch on Google Colab

 launch binder

Projeto #2

 Launch on Google Colab

 launch binder

# Retropropagação do Erro

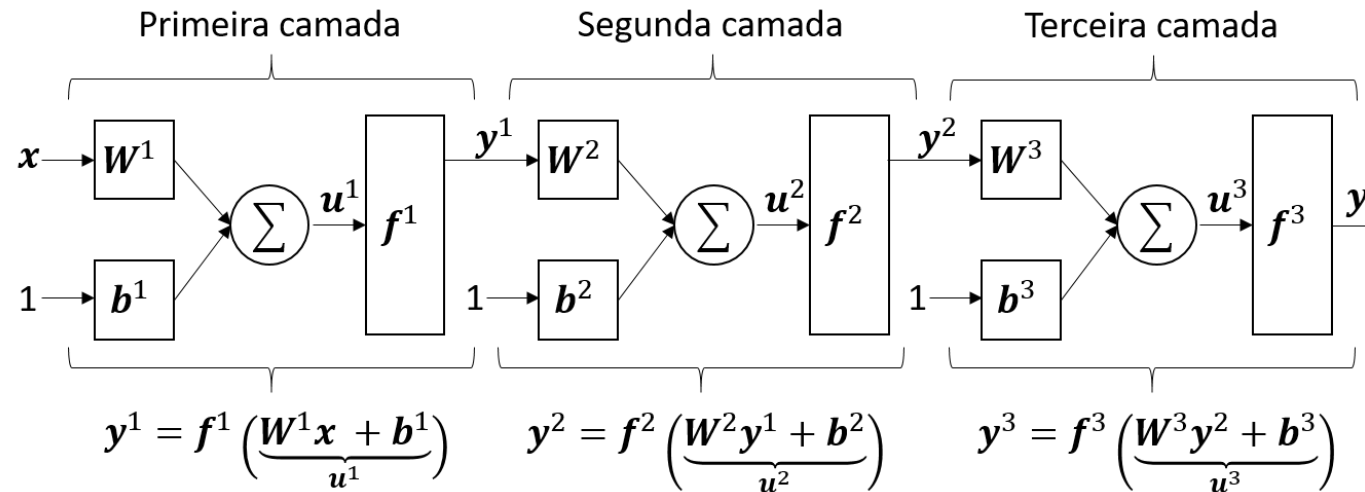
- Conforme nós discutimos anteriormente, os métodos fundamentais de ***aprendizado*** para ***redes neurais*** são baseados no cálculo das ***derivadas parciais*** da ***função de erro*** (ou de ***custo/perda***) com relação aos ***pesos sinápticos***.
- Esses métodos têm como objetivo encontrar o ***conjunto de pesos sinápticos*** que minimize a ***métrica (função) de erro*** escolhida.
- Para isso, é necessário encontrar uma maneira de se calcular o ***vetor gradiente*** da ***função de custo*** com respeito aos ***pesos sinápticos das várias camadas de uma rede neural***.
- Essa tarefa pode parecer óbvia, mas não é o caso.
  - Como podemos calcular a influência dos pesos das camadas ocultas no erro da camada de saída?
- Foram necessários 17 anos desde a criação do ***Perceptron*** até que se ***“descobrisse”*** uma forma de treinar RNAs.

# Retropropagação do Erro

- Para que entendamos melhor o porquê de não ser uma tarefa trivial, nós iremos considerar a notação abaixo, a qual será muito útil a seguir.
  - O peso sináptico,  $w_{i,j}^m$ , corresponde ao  $j$ -ésimo peso do  $i$ -ésimo **nó** da  $m$ -ésima camada da **rede neural** e  $W^m$  é a matriz com todos os pesos da  $m$ -ésima camada.
  - O peso de bias,  $b_i^m$ , corresponde ao peso do  $i$ -ésimo **nó** da  $m$ -ésima camada da **rede neural** e  $b^m$  é o vetor com todos os pesos de bias da  $m$ -ésima camada.
  - A **ativação**,  $u_i^m$ , corresponde à **combinação linear** das entradas do  $i$ -ésimo **nó** da  $m$ -ésima camada da **rede neural** e  $u^m$  é o **vetor de ativações** com as **combinações lineares** das entradas de todos os nós da  $m$ -ésima camada.
  - $f^m(.)$  é a função de ativação da  $m$ -ésima camada da **rede neural**.
  - Com essa notação, obter o **vetor gradiente** significa calcular, de maneira genérica,  $\frac{\partial J(w)}{\partial w_{i,j}^m}$ , ou seja, calcular essa derivada para todos os pesos de todos os **nós**.

# Retropropagação do Erro

- A figura abaixo apresenta um exemplo de como uma rede MLP pode ser descrita segundo essa notação.



**OBS.:** Para facilitar nossa análise, não vamos considerar as entradas como uma camada, apenas as camadas ocultas e de saída.

- O mapeamento realizado pela rede MLP acima é dado por:

$$y^3 = f^3\left(\underbrace{W^3 \underbrace{f^2\left(W^2 \underbrace{f^1(W^1 x + b^1)}_{y^1} + b^2\right)}_{y^2} + b^3\right)$$

- Para facilitar nosso trabalho, iremos supor, sem nenhuma perda de generalidade, que a **função de custo** escolhida é o **erro quadrático médio** (MSE).

# Retropropagação do Erro

- Nós vamos assumir que a **última camada da rede MLP** (definida como a  $M$ -ésima camada) tenha uma quantidade genérica,  $N_M$ , de **nós**. Assim, o MSE é dado por

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{N_{\text{dados}} N_M} \sum_{n=1}^{N_{\text{dados}}} \sum_{j=1}^{N_M} e_j^2(n) \\ &= \frac{1}{N_{\text{dados}} N_M} \sum_{n=1}^{N_{\text{dados}}} \sum_{j=1}^{N_M} \left( d_j(n) - y_j^M(n) \right)^2, \end{aligned}$$

onde  $N_{\text{dados}}$  é o número de exemplos,  $d_j(n)$  e  $y_j^M(n)$  são o valor desejado da  $j$ -ésima saída (i.e., rótulo) e a saída do  $j$ -ésimo nó da  $M$ -ésima camada, respectivamente, ambos correspondentes ao  $n$ -ésimo exemplo de entrada.

- Para treinar a rede (i.e., atualizar os pesos), devemos derivar a **função custo** com respeito aos **pesos sinápticos**.
- Porém, percebam que os **pesos das camadas ocultas não aparecem explicitamente** na expressão do erro,  $J(\mathbf{w})$ , apenas os da camada de saída, como veremos a seguir.

# Retropropagação do Erro

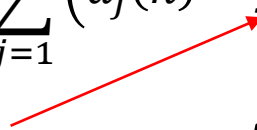
- Para fazer com que a dependência dos pesos apareça de maneira clara na expressão do erro, nós precisamos recorrer a aplicações sucessivas da **regra da cadeia**.
- Usando a notação de **Leibniz**, essa regra nos mostra que:

$$\frac{\partial f(g(h(x)))}{\partial x} = \frac{\partial f(g(h(x)))}{\partial g(h(x))} \frac{\partial g(h(x))}{\partial h(x)} \frac{\partial h(x)}{\partial x}.$$

- Por exemplo, vamos considerar que  $f(g(x)) = e^{x^2}$  e que queremos obter  $\frac{\partial f(g(x))}{\partial x}$ .
- Nós podemos fazer  $g(x) = x^2$  e usar a **regra da cadeia**:

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x} = e^{g(x)} 2x = 2xe^{x^2}.$$

# Retropropagação do Erro

$$J(\mathbf{w}) = \frac{1}{N_{\text{dados}} N_M} \sum_{n=1}^{N_{\text{dados}}} \sum_{j=1}^{N_M} \left( d_j(n) - y_j^M(n) \right)^2$$


- Agora voltamos à equação do MSE e vemos que ***as saídas da M-ésima camada (i.e., saída) da rede aparecem de maneira direta na equação.***
- Isso significa que é ***simples se obter as derivadas com respeito aos pesos desta camada.***
- Porém, quando precisamos avaliar as ***derivadas com respeito aos pesos das camadas anteriores (i.e., ocultas)***, a situação fica mais complexa, pois não existe uma dependência direta.
- Portanto surge a pergunta, como podemos atribuir a cada ***nó*** de uma camada oculta da rede, e, conseqüentemente a seus pesos, sua devida influência na composição dos valores de saída e, conseqüentemente, do erro?
  - Propaga-se o erro calculado na saída da rede neural para suas camadas anteriores até a primeira camada oculta usando-se um algoritmo, baseado na regra da cadeia, conhecido como ***backpropagation*** ou ***retropropagação do erro.***



# Retropropagação do Erro

- A seguir, veremos de maneira mais **sistemática** como a **retropropagação do erro** é realizada.
- Inicialmente, nós devemos observar um fato fundamental. O cálculo da derivada do MSE com respeito a um peso qualquer é dada por:

$$\frac{\partial J(\mathbf{w})}{\partial w_{i,j}^m} = \frac{\partial \sum_{n=1}^{N_{\text{dados}}} \sum_{k=1}^{N_M} e_k^2(n)}{\partial w_{i,j}^m} = \sum_{n=1}^{N_{\text{dados}}} \sum_{k=1}^{N_M} \frac{\partial e_k^2(n)}{\partial w_{i,j}^m}.$$

**OBS.:** mudei o índice do erro de  $j$  para  $k$ .

- **OBS.1:** Operação da derivada parcial é **distributiva**.
- **OBS.2:** A divisão pelo número de amostras é omitida, pois não afeta a otimização.
- A equação acima mostra que é necessário se calcular a derivada parcial apenas do quadrado do erro associado ao  $n$ -ésimo exemplo de entrada da  $k$ -ésima saída, pois o gradiente será a **média destes gradientes particulares** (ou **locais**).

# Retropropagação: Algumas noções básicas

- Considerando a derivada geral  $\frac{\partial J(\mathbf{w})}{\partial w_{i,j}^m}$  (i.e., um elemento genérico do vetor gradiente) e usando a **regra da cadeia**, podemos reescrevê-la como:

$$\frac{\partial J(\mathbf{w})}{\partial w_{i,j}^m} = \frac{\partial J(\mathbf{w})}{\partial u_i^m} \frac{\partial u_i^m}{\partial w_{i,j}^m}.$$

- A primeira derivada após a igualdade é a derivada da **função de custo** com respeito à **ativação**,  $u_i^m$ , do  $i$ -ésimo **nó** da  $m$ -ésima camada.
- Essa grandeza será chamada de **sensibilidade** e é denotada pela letra grega  $\delta$ . Desta forma:

$$\delta_i^m = \frac{\partial J(\mathbf{w})}{\partial u_i^m}.$$

Sensibilidade do  $i$ -ésimo nó da  $m$ -ésima camada.

- O termo  $\delta_i^m$  é único para cada **nó** da  $m$ -ésima camada.
- O outro termo, por sua vez, varia ao longo das entradas do **nó** em questão. Como adotamos nós do **tipo perceptron**, a ativação,  $u_i^m$ , é uma **combinação linear** das entradas:


$$u_i^m = \sum_{j \in \text{entradas}} w_{i,j}^m y_j^{m-1} + b_i^m.$$

# Retropropagação: Algumas noções básicas

- Assim

$$\frac{\partial u_i^m}{\partial w_{i,j}^m} = y_j^{m-1}.$$

Saída da  
camada  
anterior.



- Caso a derivada seja em relação ao termo de **bias**,  $b_i^m$ , teremos o seguinte resultado

$$\frac{\partial u_i^m}{\partial b_i^m} = 1.$$

- Desta forma, vemos que ***todas as derivadas da função de custo com respeito aos pesos (sinápticos/bias) são produtos de uma sensibilidade,  $\delta_i^m$ , por uma entrada do i-ésimo nó da rede (ou, no caso dos termos de bias, pela unidade).***

$$\frac{\partial J(\mathbf{w})}{\partial w_{i,j}^m} = \frac{\partial J(\mathbf{w})}{\partial u_i^m} \frac{\partial u_i^m}{\partial w_{i,j}^m} = \delta_i^m y_j^{m-1},$$

ou, para o peso de bias,  $b_i^m$

$$\frac{\partial J(\mathbf{w})}{\partial b_i^m} = \frac{\partial J(\mathbf{w})}{\partial u_i^m} \frac{\partial u_i^m}{\partial b_i^m} = \delta_i^m.$$

- São os valores de **sensibilidade**,  $\delta_i^m$ , que trazem mais dificuldades em seu cálculo, pois a derivada  $\frac{\partial u_i^m}{\partial w_{i,j}^m}$  é trivial (ela é apenas o valor de uma entrada daquele nó).

# Retropropagando o erro

- Portanto, a estratégia de otimização adotada para atualização dos pesos (sinápticos e de bias) da rede neural é a seguinte:
  1. Começa-se pela saída, onde o erro é calculado.
    - Etapa chamada de **direta**, pois aplica-se as entradas à rede e calcula-se o erro de saída.
  2. Encontra-se uma **regra recursiva** que gere os valores de **sensibilidade** para os **nós** das camadas anteriores até a primeira camada oculta.
    - Etapa chamada de **reversa**, pois calcula-se a **contribuição de cada nó** das camadas ocultas no erro de saída.
- Esse processo é chamado de **retropropagação do erro** ou **backpropagation**.
- Para facilitar a **retropropagação do erro**, nós vamos inicialmente agrupar todas as **sensibilidades** da  $m$ -ésima camada,  $\delta_i^m, \forall i$ , em um vetor,  $\delta^m$ .
- Em seguida, vamos encontrar uma regra que fará a transição  $\delta^m \rightarrow \delta^{m-1}$ .
- Ou seja, a partir do vetor de **sensibilidades** da camada  $m$ , iremos encontrar o vetor de **sensibilidades** da camada anterior,  $m - 1$ .

# Retropropagando o erro

- Em resumo, o processo de **retropropagação do erro** é iniciado calculando-se o **vetor de sensibilidades** da última camada,  $\delta^M$ , e, de maneira **recursiva**, obtém-se os **vetores de sensibilidades** de todas as camadas anteriores.
- Para calcular  $\delta^M$  (vetor de sensibilidades da camada de saída) consideramos  $N_M$  saídas e, assim, temos que o  $j$ -ésimo elemento do vetor  $\delta^M$  é dado por:

$$\begin{aligned}\delta_j^M &= \frac{\partial e_j^2}{\partial u_j^M} = \frac{\partial (d_j - y_j^M)^2}{\partial u_j^M} \stackrel{\text{Regra da cadeia}}{=} \frac{\partial (d_j - y_j^M)^2}{\partial y_j^M} \frac{\partial y_j^M}{\partial u_j^M} = -2(d_j - y_j^M) \frac{\partial y_j^M}{\partial u_j^M} \\ &= -2(d_j - y_j^M) f'^M(u_j^M),\end{aligned}$$

onde

$$\begin{aligned}y_j^M &= f^M(u_j^M), \\ f'^M(u_j^M) &= \frac{\partial f^M(u_j^M)}{\partial u_j^M}.\end{aligned}$$

Função logística

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

Função tangente hiperbólica

$$\frac{\partial f(u)}{\partial u} = (1 - \tanh^2(u))$$

# Retropropagando o erro

- Matricialmente nós podemos expressar o vetor  $\delta^M$  como:

$$\delta^M = -2\mathbf{F}'^M(\mathbf{u}^M)(\mathbf{d} - \mathbf{y}),$$

onde a matriz  $\mathbf{F}'^M(\mathbf{u}^M)$  é uma **matriz diagonal** com as derivadas das funções de ativação em relação às ativações dos  $N_M$  nós da  $M$ -ésima camada,

$$\mathbf{F}'^M(\mathbf{u}^M) = \begin{bmatrix} f'^M(u_1^M) & 0 & \cdots & 0 \\ 0 & f'^M(u_2^M) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f'^M(u_{N_M}^M) \end{bmatrix},$$

$\mathbf{d}$  e  $\mathbf{y}$  são vetores coluna de dimensão  $N_M \times 1$  com os valores esperados e de saída da rede neural, respectivamente.

- Desta forma, a aplicação sucessiva da **regra da cadeia** leva a uma recursão que, em termos matriciais, é simples e dada por

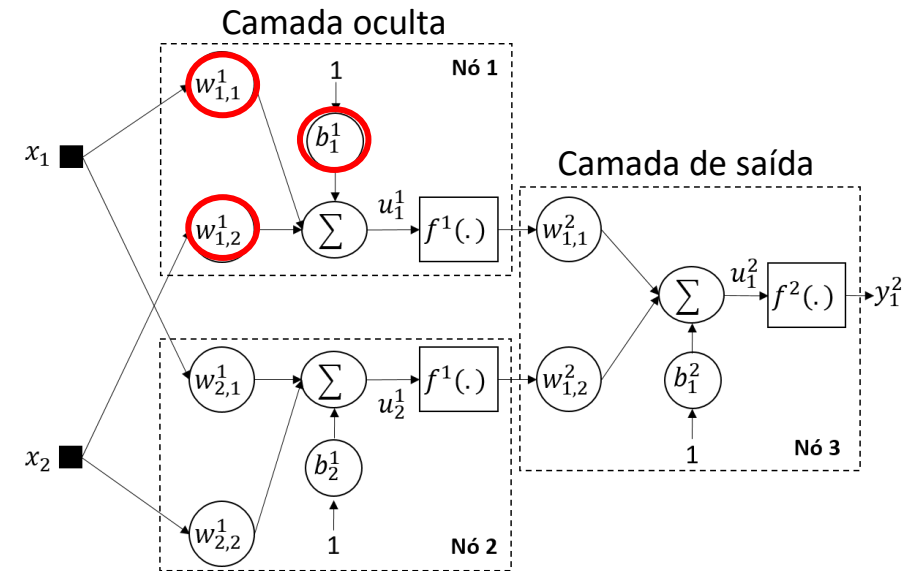
$$\delta^{m-1} = \mathbf{F}'^{m-1}(\mathbf{u}^{m-1})(\mathbf{W}^m)^T \delta^m.$$

Matriz ou vetor com os pesos que conectam a camada  $m - 1$  à camada  $m$ .

# Exemplo da aplicação da retropropagação

- Encontrar o ***vetor gradiente*** para todos os pesos do nó 1 (camada oculta) da rede neural abaixo.

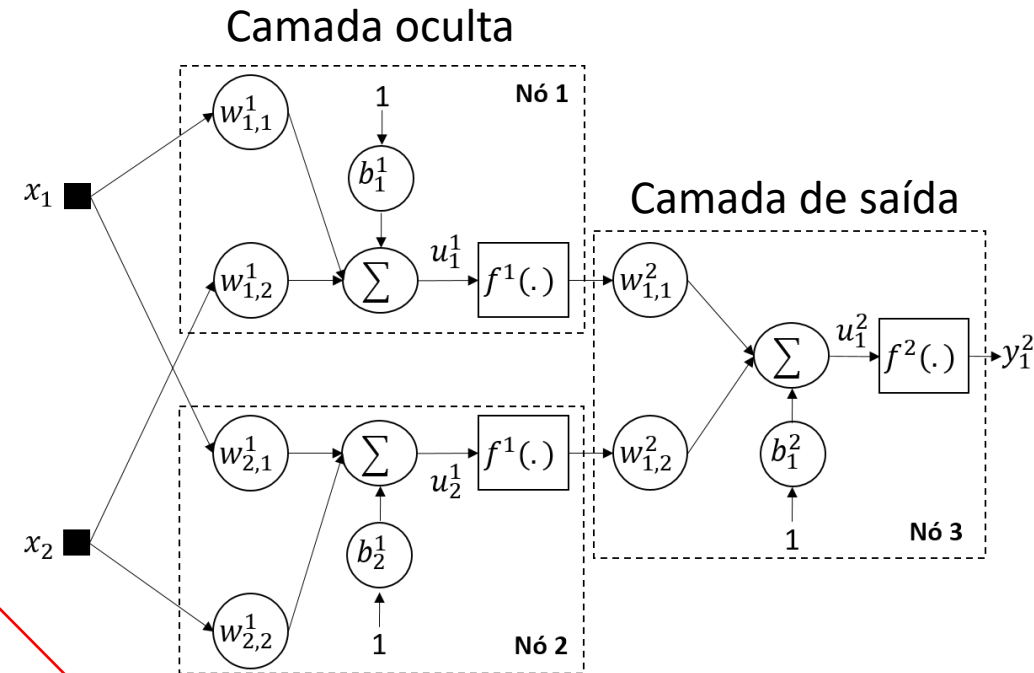
$$\begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_{1,1}^1} \\ \frac{\partial J(\mathbf{w})}{\partial w_{1,2}^1} \\ \frac{\partial J(\mathbf{w})}{\partial b_1^1} \end{bmatrix} = ?$$



- **OBS.:** vamos deixar as derivadas da função de ativação em relação às ativações de forma genérica, ou seja, sem assumir um tipo específico de função de ativação.

# Exemplo da retropropagação do erro

- Vamos considerar uma rede MLP com uma camada oculta com dois nós e uma **camada de saída com um único nó**, portanto  $M = 2$ .
- Devemos começar calculando  $\delta^2$ .
- Percebam que essa **sensibilidade** é um escalar pois há apenas um **nó** na camada de saída.
- Vamos considerar um **único exemplo de entrada**  $\mathbf{x} = [x_1, x_2]$  e a respectiva saída desejada,  $d$ .
- Supomos que os pesos de todos os nós têm uma certa configuração inicial (e.g., dist. normal).
- Assim, quando a entrada,  $\mathbf{x}$ , é apresentada à rede, é possível calcular todos os valores de interesse ao longo dela até sua saída.
- Essa é a etapa **direta** (ou do inglês, **forward**).



$$\frac{\partial J(\mathbf{w})}{\partial w_{i,j}^m} = \frac{1}{N_{\text{dados}} N_M} \sum_{n=1}^{N_{\text{dados}}} \sum_{k=1}^{N_M} \frac{\partial e_k^2(n)}{\partial w_{i,j}^m} = \frac{\partial e_k^2(n)}{\partial w_{i,j}^m}$$



# Exemplo da retropropagação do erro

- Portanto, temos então a saída  $y_1^2$ , onde o erro pode ser calculado como

$$e_1 = d - y_1^2.$$

- De posse do erro, podemos calcular a sensibilidade do **nó** da camada de saída

$$\delta^2 = -2(d - y_1^2)f'^2(u_1^2).$$

- Temos, portanto, nossa primeira **sensibilidade**. Agora, usamos a equação de recursão para **retropropagar** o erro até a camada anterior. A fórmula nos diz:

$$\delta^1 = \mathbf{F}'^1(\mathbf{u}^1)(\mathbf{W}^2)^T \delta^2,$$

onde  $(\mathbf{W}^2)^T = [w_{1,1}^2, w_{1,2}^2]^T$  e

$$\mathbf{F}'^1(\mathbf{u}^1) = \begin{bmatrix} f'^1(u_1^1) & 0 \\ 0 & f'^1(u_2^1) \end{bmatrix}.$$

**OBS.:** Notem que  $.^2$  aqui não significa “ao quadrado”, mas sim a indicação de que se trata de um valor da camada  $m = 2$ .

# Exemplo da retropropagação do erro

- Portanto,

$$\boldsymbol{\delta}^1 = \begin{bmatrix} \delta_1^1 \\ \delta_2^1 \end{bmatrix} = \begin{bmatrix} w_{1,1}^2 f'^1(u_1^1) \\ w_{1,2}^2 f'^1(u_2^1) \end{bmatrix} \delta^2.$$

- Agora, para obtermos o vetor gradiente, multiplicamos as **sensibilidades** pelas entradas correspondentes.
- Por exemplo, as derivadas parciais com relação aos pesos do **nó**  $i = 1$  da camada  $m = 1$  são mostradas abaixo

$$\begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_{1,1}^1} \\ \frac{\partial J(\mathbf{w})}{\partial w_{1,2}^1} \\ \frac{\partial J(\mathbf{w})}{\partial b_1^1} \end{bmatrix} = \delta_1^1 \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \delta^2 w_{1,1}^2 f'^1(u_1^1) \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = -2(d - y_1^2) f'^2(u_1^2) w_{1,1}^2 f'^1(u_1^1) \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}.$$

Os pesos de **bias** estão ligados a entradas com valores constantes iguais a 1.

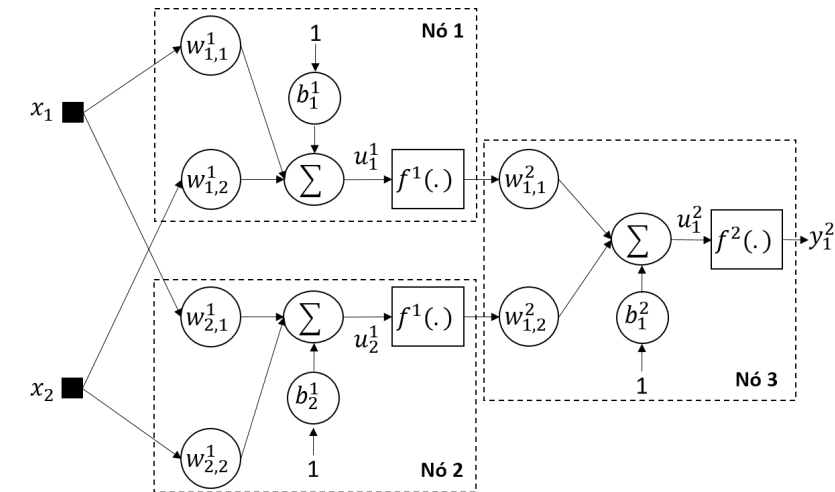
# Exemplo da retropropagação do erro

- Se fôssemos calcular as derivadas **aplicando a regra da cadeia diretamente**, elas seriam calculadas como mostrado abaixo.

$$\frac{\partial J(\mathbf{w})}{\partial w_{1,1}^1} = \underbrace{\frac{\partial \left( d - f^2(u_1^2) \right)^2}{\partial f^2(u_1^2)}}_{\delta^2} \underbrace{\frac{\partial f^2(u_1^2)}{\partial u_1^2} \frac{\partial u_1^2}{\partial f^1(u_1^1)} \frac{\partial f^1(u_1^1)}{\partial u_1^1}}_{\delta_1^1} \underbrace{\frac{\partial u_1^1}{\partial w_{1,1}^1}}_{x_1}$$

- Resolvendo as derivadas parciais, temos

$$\frac{\partial J(\mathbf{w})}{\partial w_{1,1}^1} = -2(d - y_1^2) f'^2(u_1^2) w_{1,1}^2 f'^1(u_1^1) x_1$$



# Exemplo da retropropagação do erro

- Aplicando-se o mesmo procedimento aos outros pesos, temos:

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial w_{1,1}^1} &= \frac{\partial e^2}{\partial w_{1,1}^1} = \frac{\partial \left( d - f^2(u_1^2) \right)^2}{\partial f^2(u_1^2)} \frac{\partial f^2(u_1^2)}{\partial u_1^2} \frac{\partial u_1^2}{\partial f^1(u_1^1)} \frac{\partial f^1(u_1^1)}{\partial u_1^1} \frac{\partial u_1^1}{\partial w_{1,1}^1} \\ \frac{\partial J(\mathbf{w})}{\partial w_{1,2}^1} &= \frac{\partial e^2}{\partial w_{1,2}^1} = \frac{\partial \left( d - f^2(u_1^2) \right)^2}{\partial f^2(u_1^2)} \frac{\partial f^2(u_1^2)}{\partial u_1^2} \frac{\partial u_1^2}{\partial f^1(u_1^1)} \frac{\partial f^1(u_1^1)}{\partial u_1^1} \frac{\partial u_1^1}{\partial w_{1,2}^1} \\ \frac{\partial J(\mathbf{w})}{\partial b_1^1} &= \frac{\partial e^2}{\partial b_1^1} = \frac{\partial \left( d - f^2(u_1^2) \right)^2}{\partial f^2(u_1^2)} \frac{\partial f^2(u_1^2)}{\partial u_1^2} \frac{\partial u_1^2}{\partial f^1(u_1^1)} \frac{\partial f^1(u_1^1)}{\partial u_1^1} \frac{\partial u_1^1}{\partial b_1^1}\end{aligned}$$

# Tarefas

- **Quiz:** “*T320 - Quiz – Redes Neurais Artificiais (Parte VI)*” que se encontra no MS Teams.
- **Exercício Prático:** Laboratório #8.
  - Pode ser baixado do MS Teams ou do GitHub.
  - Pode ser respondido através do link acima (na nuvem) ou localmente.
  - [Instruções para resolução e entrega dos laboratórios](#).
  - **Atividades podem ser feitas em grupo, mas as entregas devem ser individuais.**
- **Projeto #2**
  - Projeto já está no github e pode ser feito em grupos de no máximo 3 alunos.
  - Entrega: **25/06/2023 até às 23:59**.
  - Leiam os enunciados atentamente.

Obrigado!

People with no idea  
about AI, telling me my  
AI will destroy the world



Me wondering why my  
neural network is  
classifying a cat as a dog..



## Deep Learning



What society thinks I do



What my friends think I do



What other computer  
scientists think I do



What mathematicians think I do

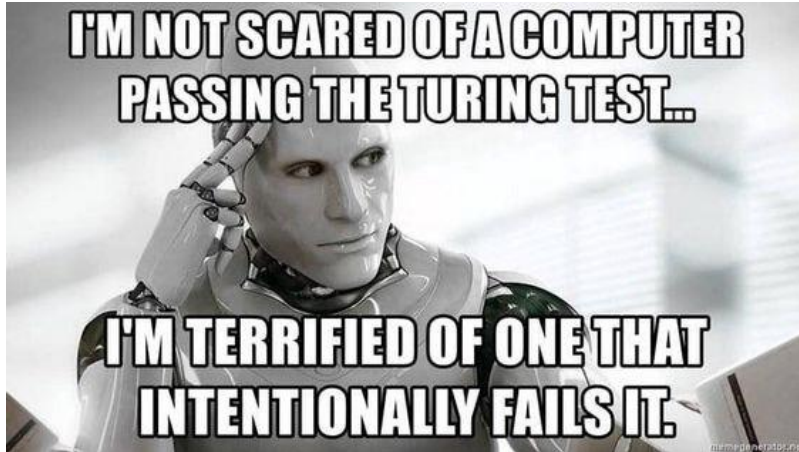


What I think I do

```
In [1]:  
import keras  
Using TensorFlow backend.
```

What I actually do

I'M NOT SCARED OF A COMPUTER  
PASSING THE TURING TEST...



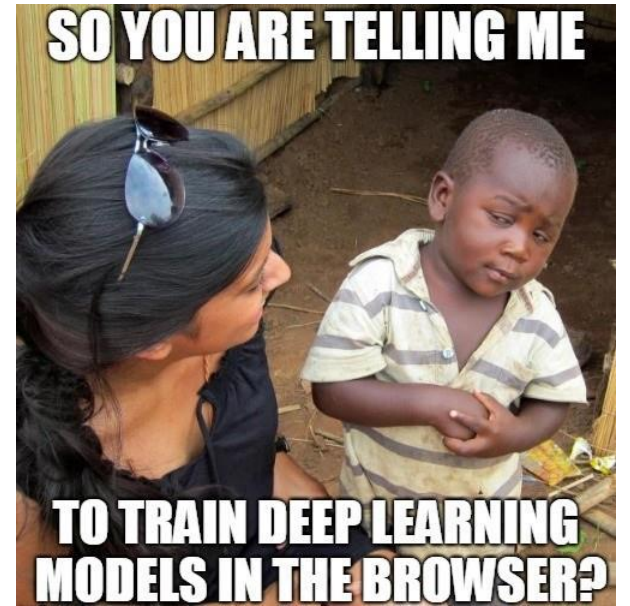
I'M TERRIFIED OF ONE THAT  
INTENTIONALLY FAILS IT.

Dog



I NEED GPU  
FOR MY DUMB  
NEURAL NETWORK

SO YOU ARE TELLING ME



TO TRAIN DEEP LEARNING  
MODELS IN THE BROWSER?

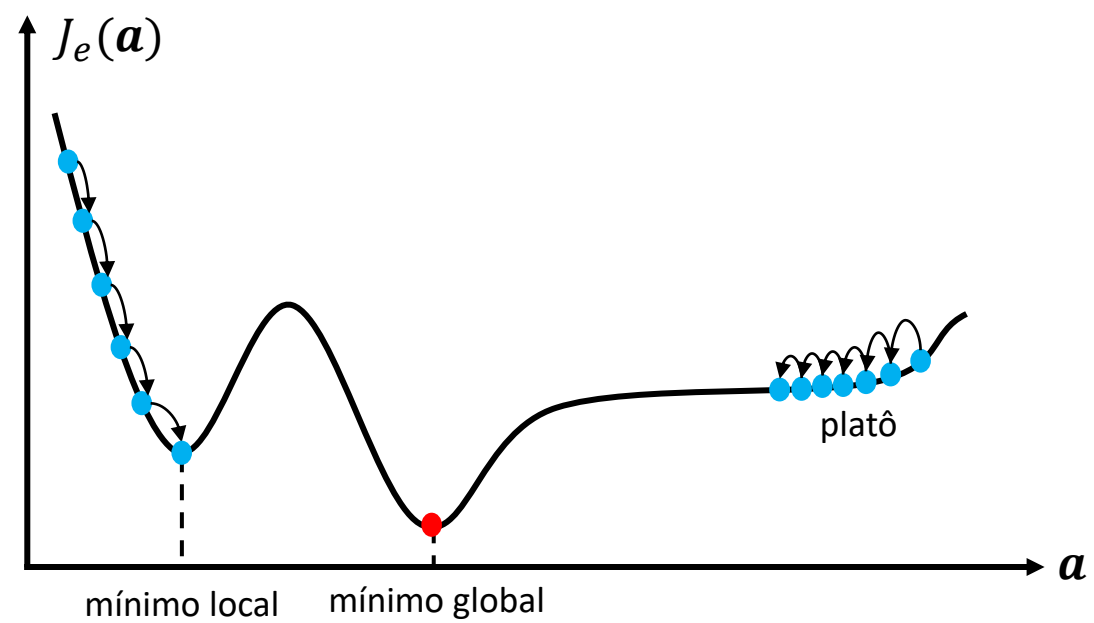
ONE DOES NOT SIMPLY

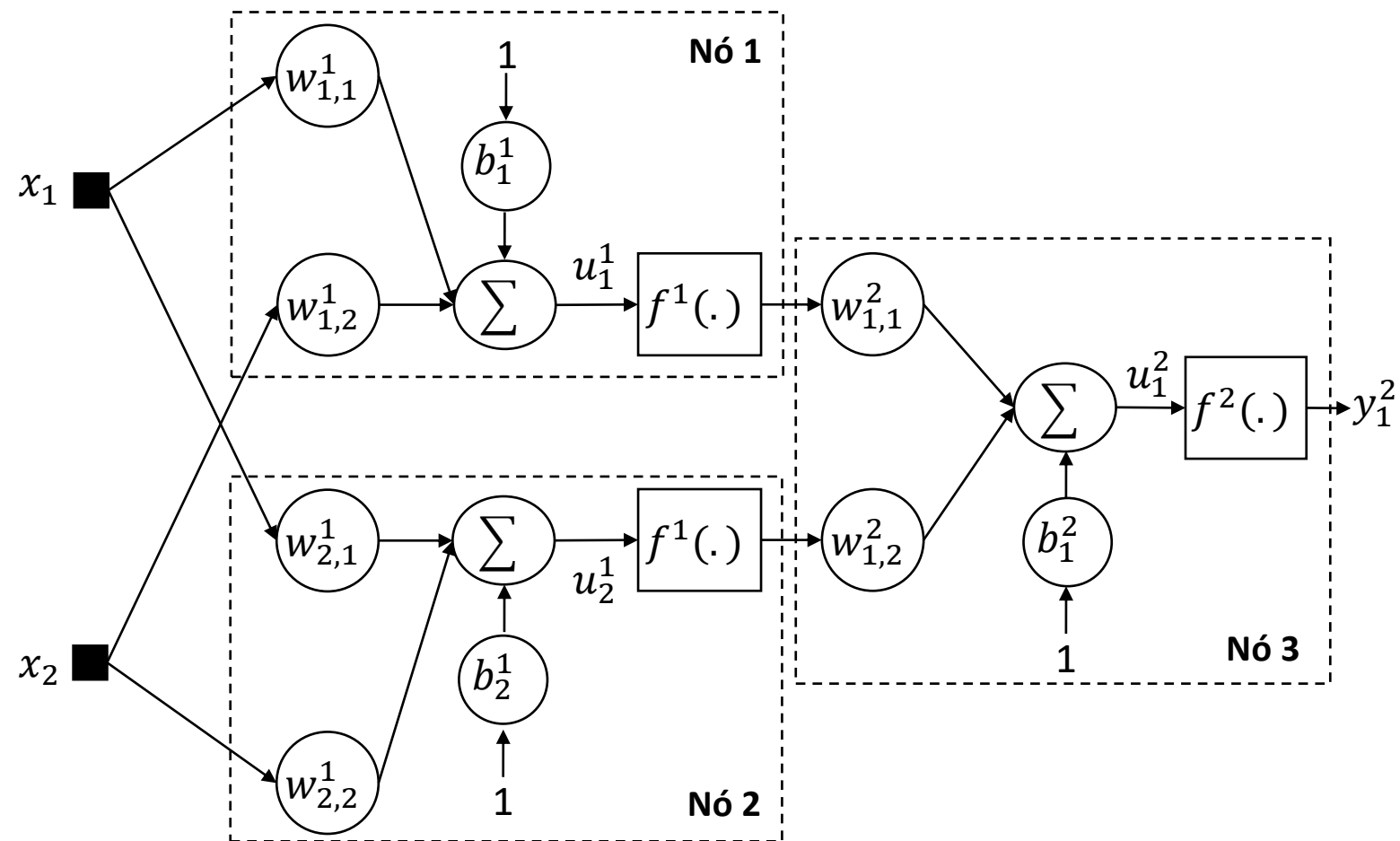


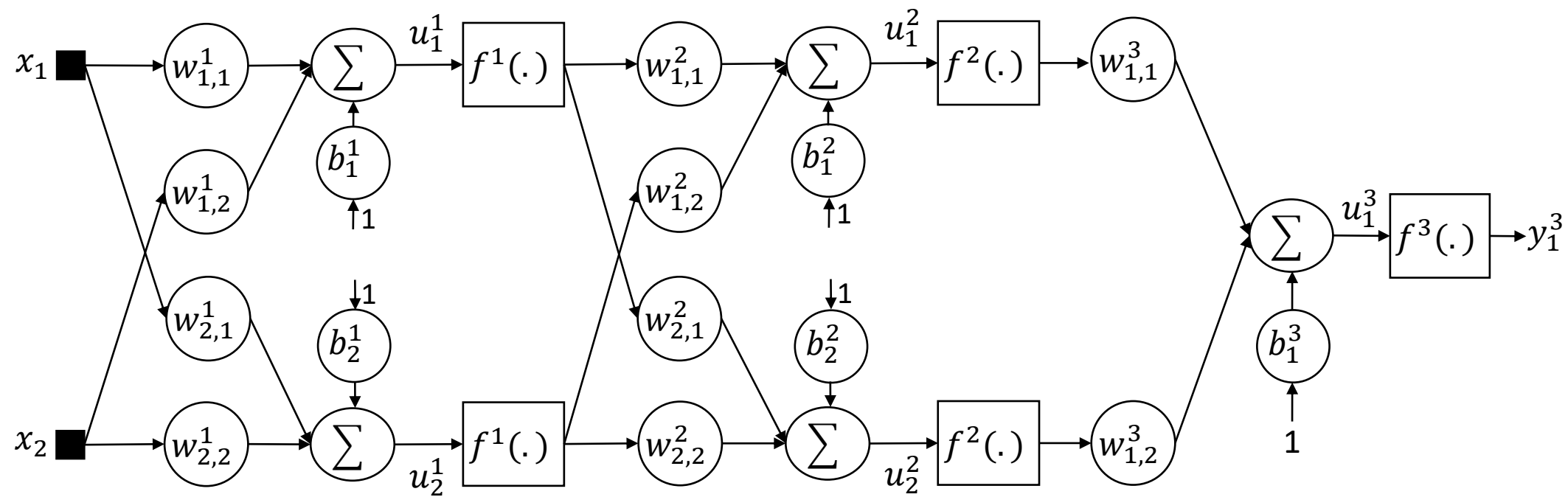
GENERATE MEMES USING DEEP  
LEARNING

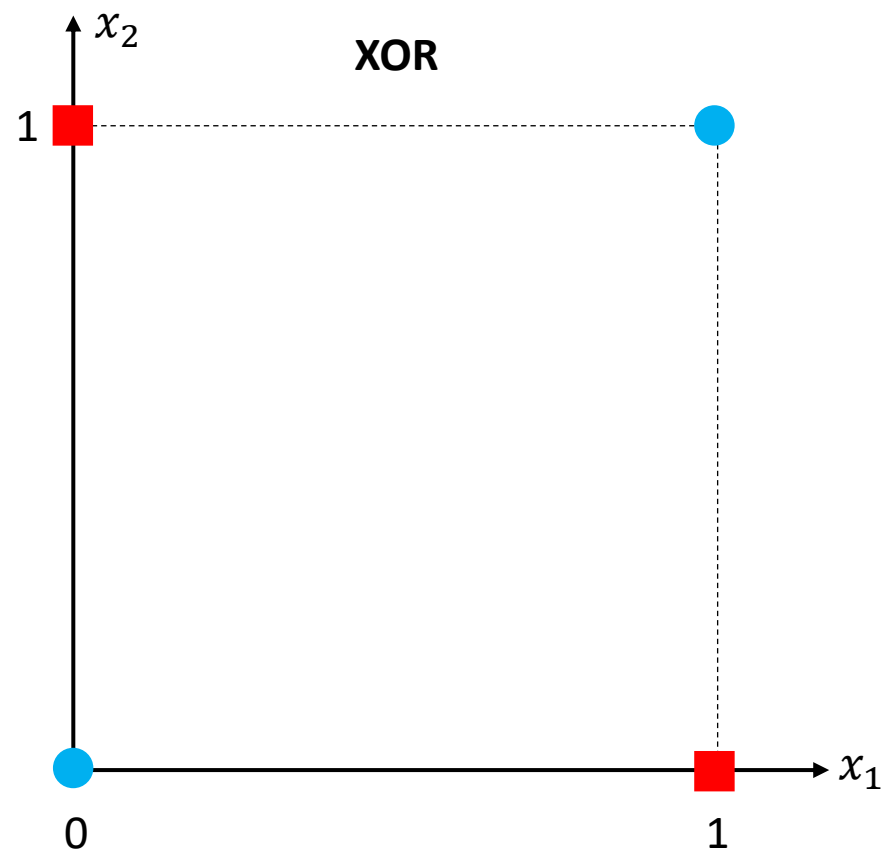
Figuras











● Classe 0 (nível lógico 0)

■ Classe 1 (nível lógico 1)