Artificial Neural Networks (ANN)

Prof. Mateus Mendelson
mendelson.mateus@gmail.com

mmendelson.com

1. Introdução

• **Redes neurais artificiais** surgiram inspiradas nas redes neurais biológicas presentes no sistema nervoso central humano.

• Tais redes consistem em um arranjo de neurônios interconectados que trocam informações entre si, permitindo a detecção, aprendizado e aplicação de padrões.

• Redes neurais artificiais são muito utilizadas em áreas ligadas à inteligência artificial, principalmente para resolver problemas de classificação, de ajuste de função (regressão), de robótica, de controle, etc.

1. Introdução

• Um neurônio biológico possui três estruturas básicas: o corpo celular, os dendritos e o axônio.

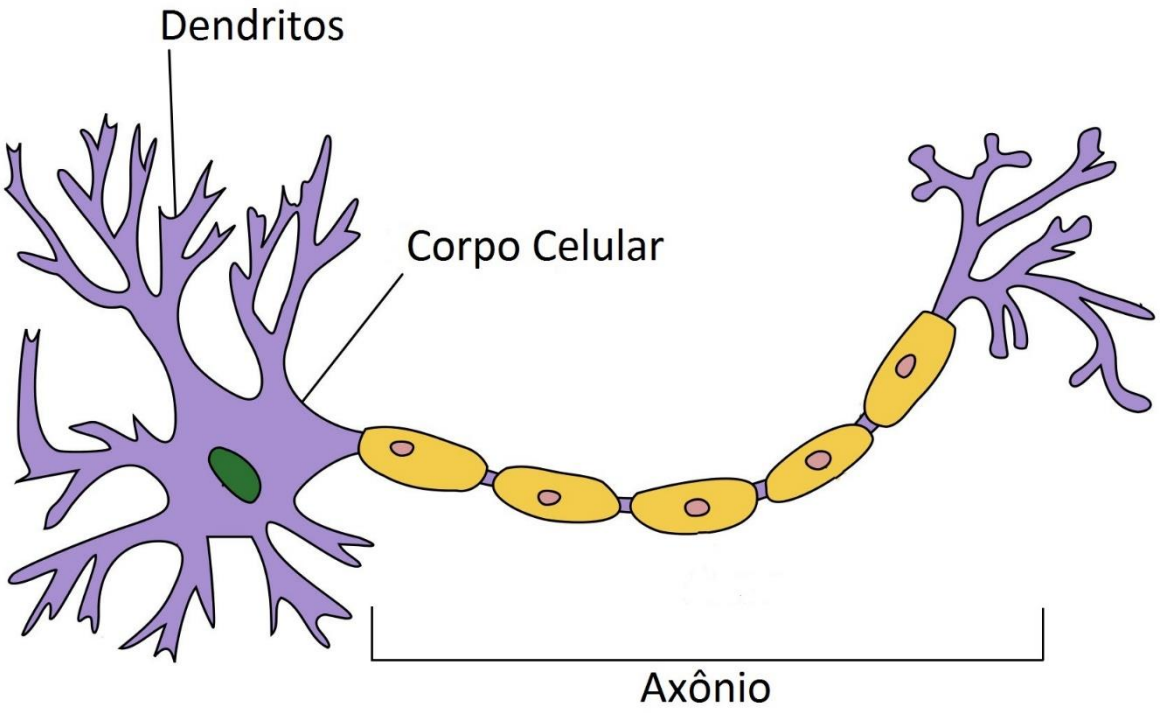
• O corpo celular contém o núcleo da célula e é responsável por praticamente toda a produção de proteínas e membranas, tal como a respiração celular.

• Os dendritos e os axônios são responsáveis por realizar a conexão entre neurônios. Dendritos se conectam apenas a axônios; axônios, apenas a dendritos.

• Os dendritos são responsáveis por receber os impulsos provenientes de outro neurônio e transmiti-los para o corpo celular.

• O corpo celular excita o axônio, que transmite novas informações aos dendritos do neurônio conectado à ele.

1. Introdução



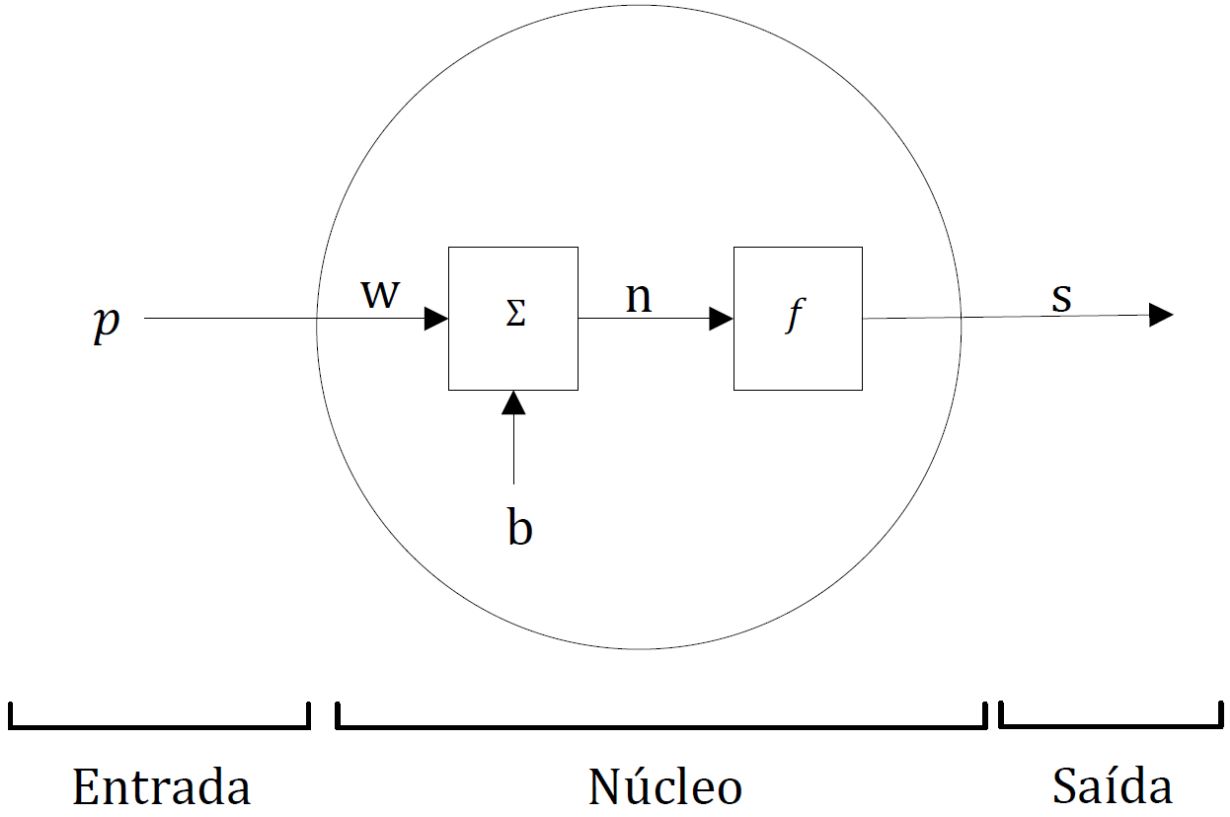
1. Introdução

- Analogamente, um neurônio artificial tem sua estrutura dividida em três partes: núcleo, entrada e saída.

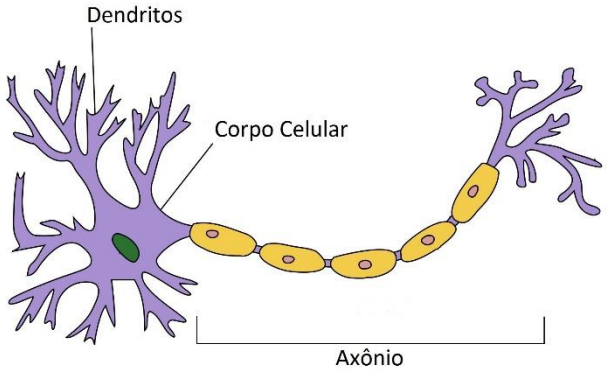
- Entrada e saída são, respectivamente, dendritos e axônio, tendo a função de apenas conectar um neurônio a outro.

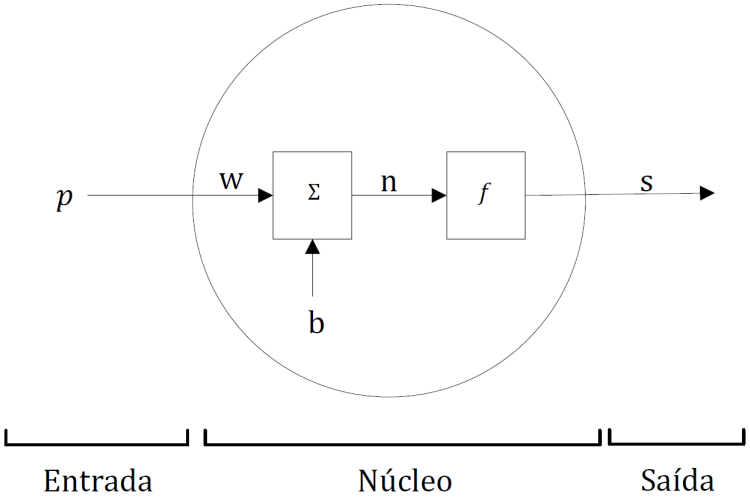
- O núcleo é o responsável por realizar os cálculos.

1. Introdução

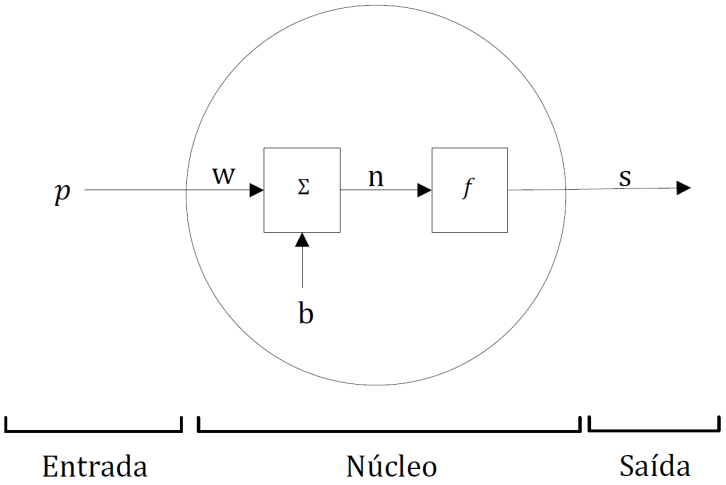


1. Introdução

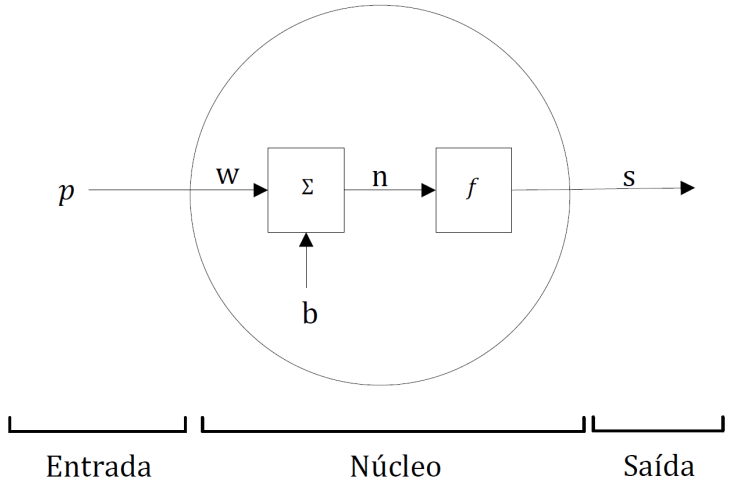




1. Introdução

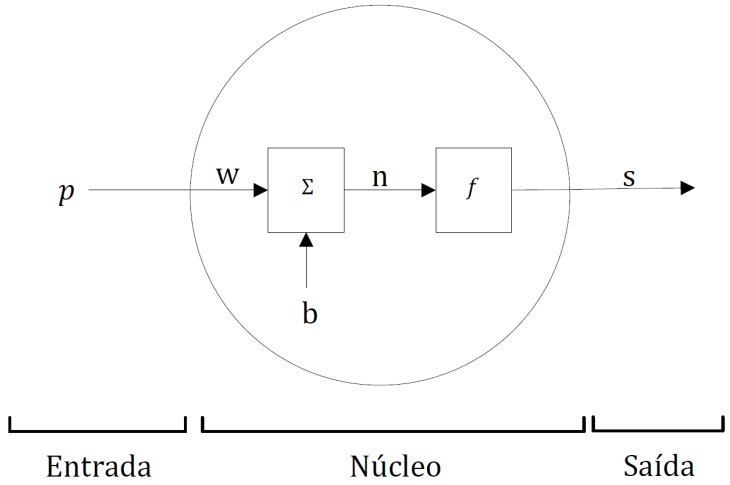


1. Introdução



$$n = \left(\sum_{i=1}^{R} p_i \times w_i \right) + b$$

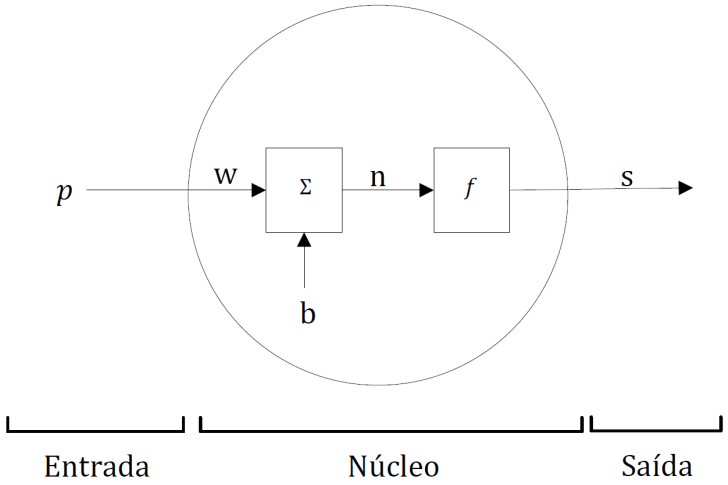
1. Introdução



$$s = f(p \cdot w + b) = f(n)$$

$$n = \left(\sum_{i=1}^{R} p_i \times w_i \right) + b$$

1. Introdução



$$s = f(p \cdot w + b) = f(n)$$

$$n = \left(\sum_{i=1}^{R} p_i \times w_i \right) + b$$

f(n) é a função de ativação e há diversas funções que podem ser utilizadas aqui, mas o processo continua o mesmo.

1. Introdução

- A função degrau é a função de ativação mais simples utilizada.

- A saída s é composta por apenas um único valor, 0 ou 1.

- O neurônio precisa ser configurado com um parâmetro, indicado por T.

1. Introdução

• A função degrau é a função de ativação mais simples utilizada.

• A saída s é composta por apenas um único valor, 0 ou 1.

• O neurônio precisa ser configurado com um parâmetro, indicado por T.

$$f(n) = u(n-T) = \begin{cases} 1, & \text{se } n - T \geq 0 \\ 0, & \text{se } n - T < 0 \end{cases}$$

1. Introdução

- Por exemplo:

  - ✓ p = [1, 0, 1]
  - ✓ w = [6, 7, 5]
  - ✓ b = 0
  - ✓ T = 15

1. Introdução

- Por exemplo:
  - ✓ p = [1, 0, 1]
  - ✓ w = [6, 7, 5]
  - ✓ b = 0
  - ✓ T = 15

$$n = \left(\sum_{i=1}^{3} p_i \times w_i \right) + b = 1 \times 6 + 0 \times 7 + 1 \times 5 + 0 = 11$$

$$\implies u(n - T) = u(11 - 15)$$

$$\implies s = 0$$

1. Introdução

- Apesar da simplicidade da função degrau, o mais comum é que outras funções sejam utilizadas.

- Saídas binárias podem prejudicar a precisão do sistema.

- Pequenas alterações nos pesos poderiam causar uma mudança brusca na saída do neurônio.

- Assim, deseja-se uma solução na qual pequenas variações nos valores dos pesos e do deslocamento causem pequenas variações na saída.

- Ao adotar uma função que permita saídas entre 0 e 1, o neurônio também fornece saídas entre 0 e 1.

- Assim temos um modelo que recebe e devolve valores reais.

1. Introdução

- Define-se a função de ativação sigmoid como abaixo.

1. Introdução

- Define-se a função de ativação sigmoid como abaixo.

$$\sigma(p) = f(p) = \frac{1}{1 + e^{-\left(\sum_{i=1}^{R} p_i \times w_i\right) + b}}$$

1. Introdução

- Define-se a função de ativação sigmoid como abaixo.

$$\sigma(\quad +b$$

1. Introdução

- Define-se a função de ativação sigmoid como abaixo.

$$\sigma(p) = f(p) = \frac{1}{1 + e^{-\left(\sum_{i=1}^{R} p_i \times w_i\right) + b}}$$

$$\sigma(n) = f(n) = \frac{1}{1 + e^{-n}}$$

1. Introdução

- Vamos refazer o exemplo anterior:

  - ✓ p = [1, 0, 1]
  - ✓ w = [6, 7, 5]
  - ✓ b = 0

1. Introdução

- Vamos refazer o exemplo anterior:
  - ✓ p = [1, 0, 1]
  - ✓ w = [6, 7, 5]
  - ✓ b = 0

$$n = \left(\sum_{i=1}^{3} p_i \times w_i \right) + b = 1 \times 6 + 0 \times 7 + 1 \times 5 + 0 = 11$$

$$\implies \sigma(n) = \sigma(11) = \frac{1}{1 + e^{-11}}$$

$$\implies s = 0.99$$

1. Introdução

- Nos limites, a função sigmoid se aproxima à função degrau.

1. Introdução

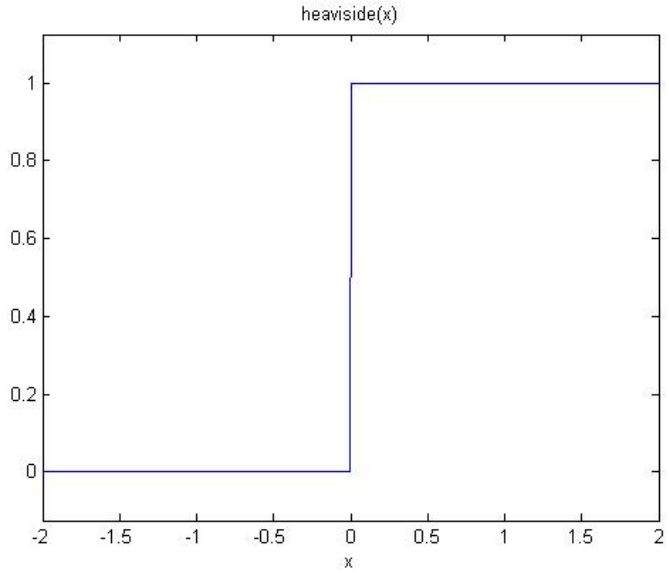
• Nos limites, a função sigmoid se aproxima à função degrau.

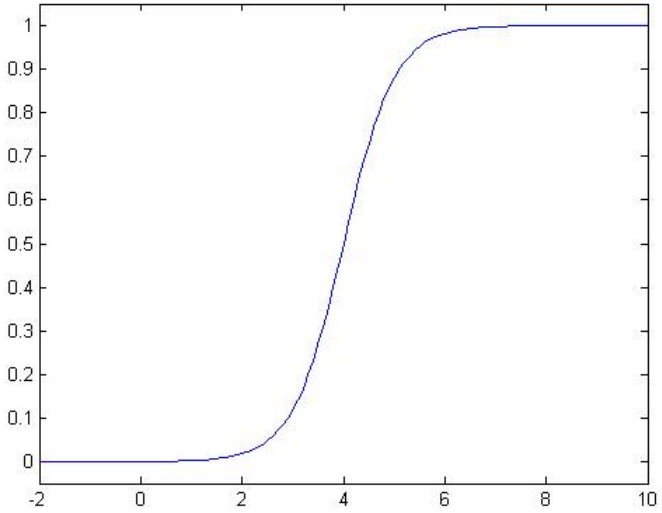
$$\lim_{n \to -\infty} \sigma(n) = \lim_{n \to -\infty} \frac{1}{1 + e^{-n}} = 0$$

$$\lim_{n \to \infty} \sigma(n) = \lim_{n \to \infty} \frac{1}{1 + e^{-n}} = 1$$

1. Introdução

• Nos limites, a função sigmoid se aproxima à função degrau.





1. Introdução

- Como é possível notar, as funções de ativação são responsáveis por definir a maneira com a qual o neurônio irá tratar as entradas. Se de forma binária, se com variações mais suaves ou mais abruptas.

- Há diversas funções que podem ser utilizadas como funções de ativação para neurônios artificiais, entretanto, além das já citadas, convém citar pelo menos mais duas delas, geralmente utilizadas em redes de múltiplas camadas: a tangente hiperbólica e a linear.

1. Introdução

- A tangente hiperbólica apresenta a seguinte fórmula:

1. Introdução

- A tangente hiperbólica apresenta a seguinte fórmula:

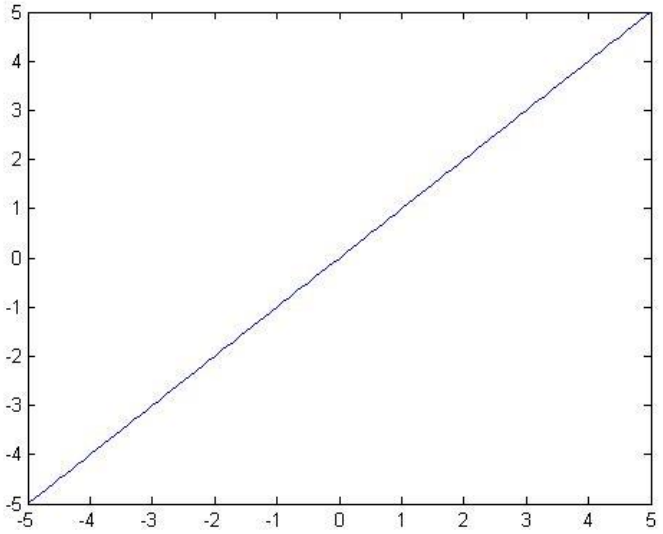
$$tanh(n) = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$$

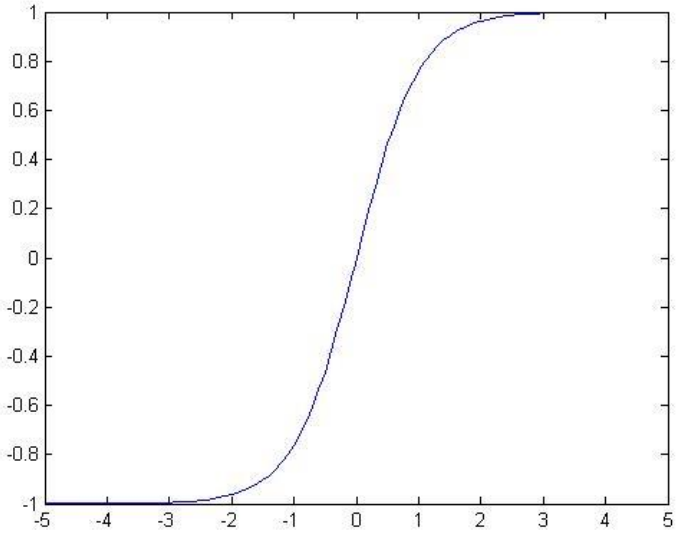
1. Introdução

• Dada sua simplicidade, a função de ativação linear não será apresentada, porém vale a pena mencionar que a tangente hiperbólica, caso seu declive seja adequadamente ajustado, pode vir a se assemelhar à uma função linear.

1. Introdução

• Dada sua simplicidade, a função de ativação linear não será apresentada, porém vale a pena mencionar que a tangente hiperbólica, caso seu declive seja adequadamente ajustado, pode vir a se assemelhar à uma função linear.





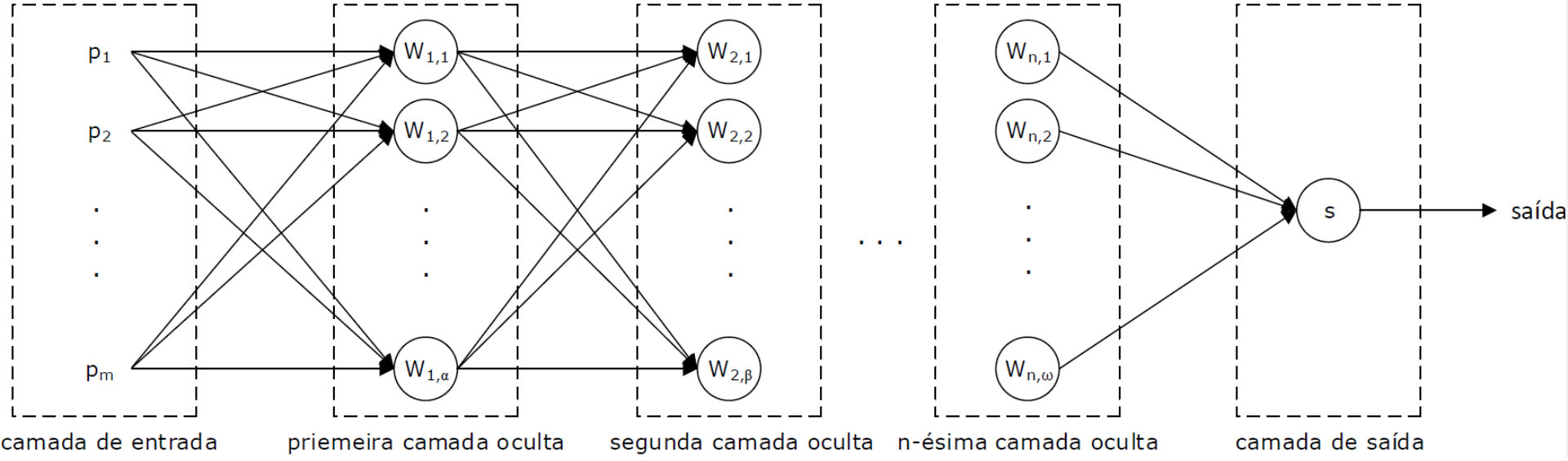
1. Introdução

• É importante notar que a saída de um neurônio pode ser configurada para abranger o intervalo de saída desejado, não se limitando apenas ao intervalo [0,1].

• Isso é feito por meio da calibração dos parâmetros do neurônio e pela escolha da função de ativação.

2. Redes *Feed-forward*

- Uma rede neural é formada por meio da ligação entre neurônios.



2. Redes *Feed-forward*

- Uma rede neural é formada por meio da ligação entre neurônios.

- Cada uma dessas colunas forma o que é chamado de camada.

- Cada camada, com exceção da última, fornece novos valores de entrada para a seguinte.

- Diz-se que o conjunto de entradas inicial para a rede forma uma camada: a camada de entrada. Ela não realiza qualquer tipo de computação, sendo formada simplesmente pelos valores de entrada para a rede.

- A última camada é chamada de camada de saída; as demais, de camadas ocultas. Redes neurais podem ter uma ou múltiplas camadas.

2. Redes *Feed-forward*

• Uma rede neural artificial *feed-forward* é um tipo de rede neural com as seguintes características:

- ✓ possui pelo menos duas camadas;
- ✓ cada neurônio de uma camada se conecta com todos os neurônios da camada seguinte e não se comunica com camadas anteriores; e
- ✓ não há conexão entre neurônios de uma mesma camada.

• Assim, esse tipo de rede não possui loops nem realimentação de neurônios. Os dados fluem a partir da entrada da rede sempre em direção à saída. Este tipo de rede é muito utilizado na resolução de problemas de reconhecimento de padrões.

3. Processo de Treinamento

- Para fazer uso de uma rede neural, é recomendado definir 3 conjuntos de dados mutuamente excludentes, ou seja, os elementos de cada conjunto devem pertencer somente a este conjunto, não estando presentes em nenhum outro:
  - ✓ Treinamento: esse conjunto deve ser utilizado para calibrar/treinar a rede, definindo os pesos w e deslocamentos b.

3. Processo de Treinamento

• Para fazer uso de uma rede neural, é recomendado definir 3 conjuntos de dados mutuamente excludentes, ou seja, os elementos de cada conjunto devem pertencer somente a este conjunto, não estando presentes em nenhum outro:

- ✓ Treinamento: esse conjunto deve ser utilizado para calibrar/treinar a rede, definindo os pesos w e deslocamentos b.
- ✓ Validação: esse conjunto é utilizado para evitar que a rede fique viciada (*overfitting*) no conjunto de treinamento. O conjunto de validação é submetido à rede a cada época do treinamento. Quando a taxa de erro sobre o conjunto de treinamento e a taxa de erro sobre o conjunto de validação se distanciam acima de um certo limiar, o treinamento é interrompido, pois houve indicação de que a rede está ficando viciada no conjunto de treinamento.

3. Processo de Treinamento

• Para fazer uso de uma rede neural, é recomendado definir 3 conjuntos de dados mutuamente excludentes, ou seja, os elementos de cada conjunto devem pertencer somente a este conjunto, não estando presentes em nenhum outro:

- ✓ Treinamento: esse conjunto deve ser utilizado para calibrar/treinar a rede, definindo os pesos w e deslocamentos b.
- ✓ Validação: esse conjunto é utilizado para evitar que a rede fique viciada (*overfitting*) no conjunto de treinamento.
- ✓ Teste: é o conjunto de elementos que realmente serão colocados à prova no sistema, ou seja, é o conjunto de dados que gera as métricas finais da rede.

3. Processo de Treinamento

• Para fazer uso de uma rede neural, é recomendado definir 3 conjuntos de dados mutuamente excludentes, ou seja, os elementos de cada conjunto devem pertencer somente a este conjunto, não estando presentes em nenhum outro:

- ✓ Treinamento: esse conjunto deve ser utilizado para calibrar/treinar a rede, definindo os pesos w e deslocamentos b.
- ✓ Validação: esse conjunto é utilizado para evitar que a rede fique viciada (*overfitting*) no conjunto de treinamento.
- ✓ Teste: é o conjunto de elementos que realmente serão colocados à prova no sistema, ou seja, é o conjunto de dados que gera as métricas finais da rede.

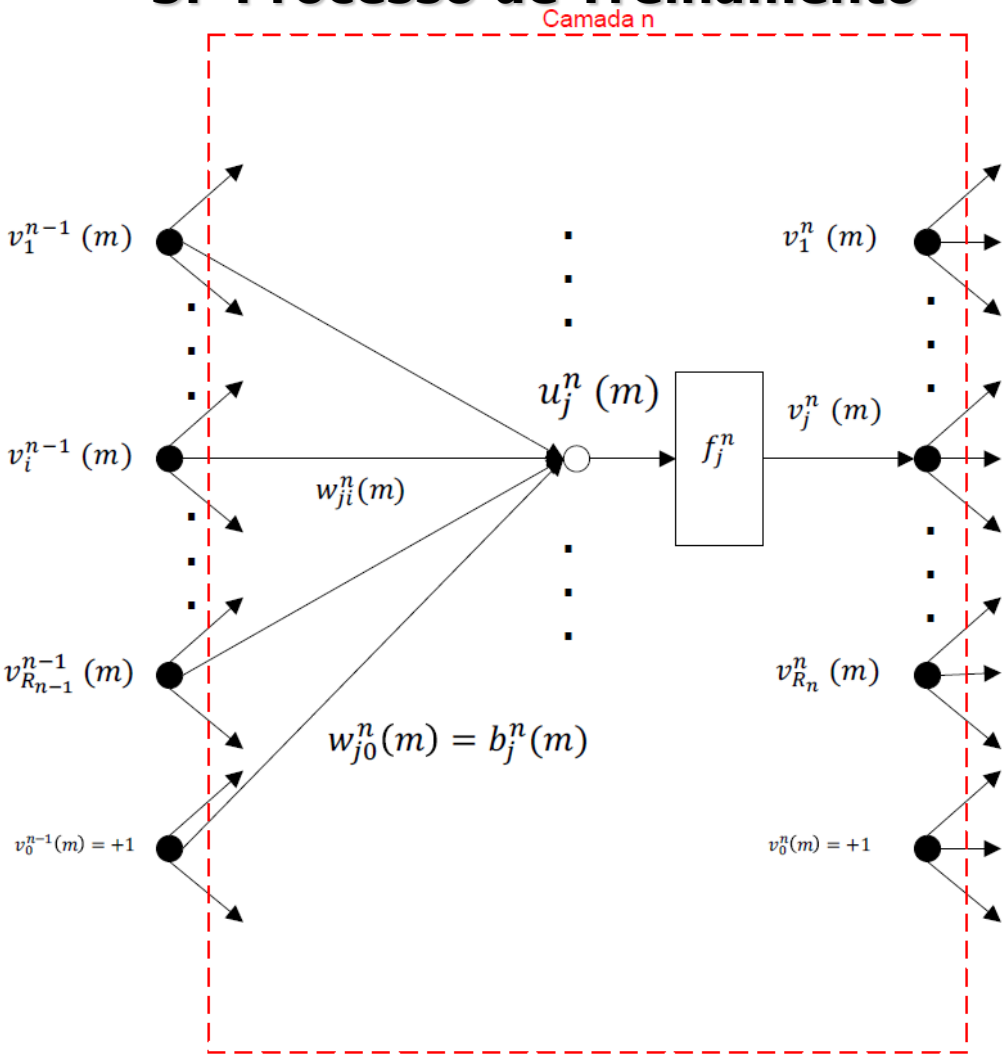
O primeiro conjunto treina a rede, o segundo conjunto evita que ela fique viciada e o terceiro é utilizado para verificar o desempenho da rede.

3. Processo de Treinamento

- O processo de treinamento passa por duas etapas principais: propagação e retropropagação (*backpropagation*).

- Para explicar esses processo, adotaremos alguns novos símbolos e nomenclaturas, então fique atento para conseguir acompanhar!

3. Processo de Treinamento



3. Processo de Treinamento

$$u_j^n(m) = \sum_{i=0}^{R_{n-1}} w_{ji}^n(m) v_i^{n-1}(m)$$

$$v_j^n(m) = f_j^n(u_j^n(m))$$

$u_j^n(m)$ ⟶ Campo local induzido do neurônio j da camada n na iteração m.

$v_i^{n-1}(m)$ ⟶ Elemento i do sinal de saída da camada n - 1 na iteração m.

$w_{ji}^n(m)$ ⟶ Valor da sinapse que vem do elemento i da camada n - 1 para o neurônio j da camada n na iteração m.

R_{n-1} ⟶ Número de elementos do vetor de saída da camada n - 1.

3. Processo de Treinamento

• Com esses cálculos, realizamos a propagação da entrada, gerando a saída da rede.

• Após a saída ter sido realizada, precisamos calcular em quanto erramos e propagar esse erro na direção oposta: da saída para a entrada, realizando as devidas correções.

• O processo de propagar o erro da saída em direção à entrada é chamado de *backpropagation*.

• Esse método nos permite encontrar bons valores para os pesos w e os deslocamentos b.

• Vamos lá!

3. Processo de Treinamento

• O erro *e* de um neurônio j de uma camada n é calculado através da subtração do resultado esperado t pelo resultado obtido pelo processo de propagação do sinal de entrada v.

$$e_j^n(m) = t_j^n(m) - v_j^n(m)$$

3. Processo de Treinamento

• Calculado o erro *e*, é necessário retropropagá-lo ajustando as sinapses e vieses da rede.

• O primeiro passo da retropropagação dos erros consiste em calcular os gradientes locais para cada neurônio.

• Para cada neurônio da camada n, os gradientes são dados pela equação abaixo.

$$\delta_j^n(m) = f'^{\,n}_{\,j}(v_j^n(m)) \sum_{i=0}^{R_{n+1}} \delta_k^{n+1}(m) w_{kj}^{n+1}(m)$$

3. Processo de Treinamento

- Para o caso em que a camada n é a camada de saída:

$$\delta_j^n(m) = f'^{\,n}_{\,j}(v_j^n(m))e_j^n(m)$$

3. Processo de Treinamento

- Uma vez calculados os gradientes locais, eles são utilizados para calcular a atualização das sinapses e vieses.

- A atualização dos valores das sinapses que ligam os neurônios da camada n - 1 aos neurônios da camada n é realizada conforme abaixo.

$$w_{ji}^{n}(m+1) = w_{ji}^{n}(m) + \eta \cdot v_{i}^{n-1}(m)\delta_{j}^{n}(m)$$

- Notem essa constante (*learning rate*).

3. Processo de Treinamento

- Finalmente, os vieses são atualizados conforme abaixo.

$$b_j^n(m+1) = b_j^n(m) + \eta \cdot \delta_j^n(m)$$

3. Processo de Treinamento

• Para melhor entendimento, vamos realizar um exemplo numérico de uma única iteração.

3. Processo de Treinamento

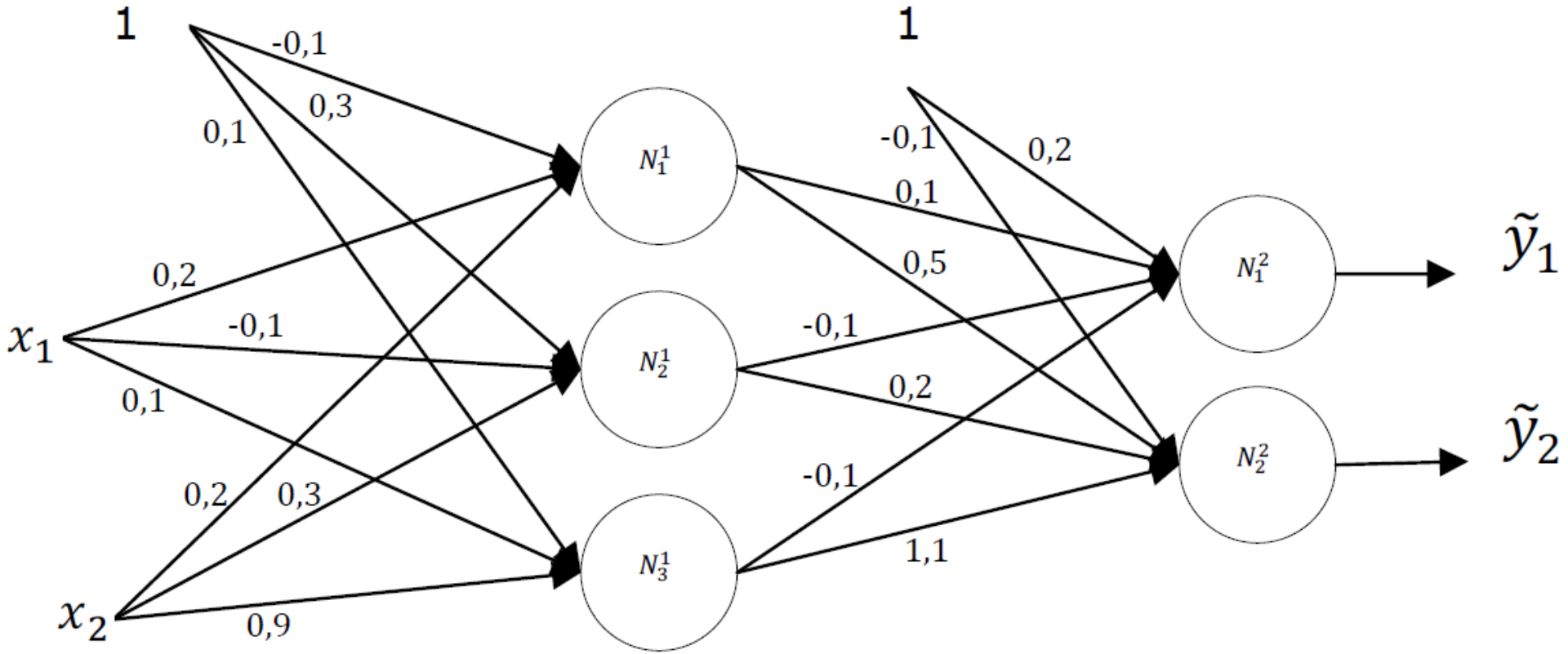
- Os valores iniciais estão na figura a seguir.

- A função de ativação de todos os neurônios é a tangente hiperbólica, exceto do neurônio 1 da camada de saída, que utiliza uma função linear.

- O objetivo é treinar a rede de tal forma que, para as entradas fornecidas, as saídas assumam os valores desejados $y_1 = 0.2$ e $y_2 = 1$.

- As entradas valem $x_1 = 0.1$ e $x_2 = 0.7$

3. Processo de Treinamento



3. Processo de Treinamento

• Calculando os valores propagados na primeira camada, temos:

$$u_1^1 = -0{,}1 + (0{,}2)\cdot(0{,}1) + (0{,}2)\cdot(0{,}7) = 0{,}06 \implies v_1^1 = tanh(0{,}06) = 0{,}06$$
$$u_2^1 = 0{,}3 + (-0{,}1)\cdot(0{,}1) + (0{,}3)\cdot(0{,}7) = 0{,}5 \implies v_2^1 = tanh(0{,}5) = 0{,}46$$
$$u_3^1 = 0{,}1 + (0{,}1)\cdot(0{,}1) + (0{,}9)\cdot(0{,}7) = 0{,}74 \implies v_3^1 = tanh(0{,}74) = 0{,}63$$

3. Processo de Treinamento

• Calculando os valores propagados na segunda camada, temos:

$$u_1^2 = 0{,}2 + (0{,}1) \cdot (0{,}06) + (-0{,}1) \cdot (0{,}46) + (-0{,}1) \cdot (0{,}63) = 0{,}097 \implies v_1^2 = 0{,}097$$

$$u_2^2 = -0{,}1 + (0{,}5) \cdot (0{,}06) + (0{,}2) \cdot (0{,}46) + (1{,}1) \cdot (0{,}63) = 0{,}715 \implies v_2^2 = tanh(0{,}715) = 0{,}614$$

• E as saídas da propagação da rede para essas duas entradas são:

$$\tilde{y}_1 = v_1^2 = 0{,}097$$

$$\tilde{y}_2 = v_2^2 = 0{,}614$$

3. Processo de Treinamento

• Comparando os valores calculados pela rede com os valores desejados, são obtidos os erros, onde e_i indica o erro alcançado na i-ésima saída da rede.

$$\epsilon_1 = y_1 - \tilde{y}_1 = 0{,}2 - 0{,}097 = 0{,}103$$
$$\epsilon_2 = y_2 - \tilde{y}_2 = 1 - 0{,}614 = 0{,}386$$

3. Processo de Treinamento

• Antes da próxima etapa, é necessário calcular as derivadas das funções de ativação nos pontos apropriados.

• Sabendo que

$$\frac{d}{dv}v = 1 \qquad \text{e} \qquad \frac{d}{dv}tanh(v) = 1 - v^2$$

temos que:

$$f'\,{}_1^1(v_1^1) = 1 - 0{,}06^2 = 1$$
$$f'\,{}_2^1(v_2^1) = 1 - 0{,}46^2 = 0{,}79$$
$$f'\,{}_3^1(v_3^1) = 1 - 0{,}63^2 = 0{,}6$$
$$f'\,{}_1^2(v_1^2) = 1$$
$$f'\,{}_2^2(v_2^2) = 1 - 0{,}61^2 = 0{,}63$$

3. Processo de Treinamento

• Após o cálculo das derivadas das funções de ativação, é possível calcular cada gradiente:

$$\delta_1^2 = f'\,{}_1^2 \cdot \epsilon_1^2 = 1 \cdot 0{,}103 = 0{,}103$$
$$\delta_2^2 = f'\,{}_2^2 \cdot \epsilon_2^2 = 0{,}63 \cdot 0{,}386 = 0{,}24$$
$$\delta_1^1 = f'\,{}_1^1 \cdot [(\delta_1^2) \cdot (w_{11}^2) + (\delta_2^2) \cdot (w_{21}^2)] = 1 \cdot [(0{,}103) \cdot (0{,}1) + (0{,}24) \cdot (0{,}5)] = 0{,}130$$
$$\delta_2^1 = f'\,{}_2^1 \cdot [(\delta_1^2) \cdot (w_{12}^2) + (\delta_2^2) \cdot (w_{22}^2)] = 0{,}79 \cdot [(0{,}103) \cdot (-0{,}1) + (0{,}24) \cdot (0{,}2)] = 0{,}03$$
$$\delta_3^1 = f'\,{}_3^1 \cdot [(\delta_1^2) \cdot (w_{13}^2) + (\delta_2^2) \cdot (w_{23}^2)] = 0{,}6 \cdot [(0{,}103) \cdot (-0{,}1) + (0{,}24) \cdot (1{,}1)] = 0{,}152$$

3. Processo de Treinamento

- Para a atualização das sinapses, segue-se a equação:

$$\Delta w_{ji}^{n} = 2 \cdot \alpha \cdot v_{j}^{n-1} \cdot \delta_{i}^{n}$$

3. Processo de Treinamento

• Adotando o valor de alpha como 0.1, serão calculados os novos valores de apenas duas sinapses e dois deslocamentos do nosso exemplo.

$$\Delta b_1^2 = 2 \cdot \alpha \cdot v_0^2 \cdot \delta_1^2 = 2 \cdot (0{,}1) \cdot 1 \cdot (0{,}103) = 0{,}021 \implies b_1^2 = 0{,}2 + 0{,}021 = 0{,}221$$
$$\Delta b_1^1 = 2 \cdot \alpha \cdot v_0^1 \cdot \delta_1^1 = 2 \cdot (0{,}1) \cdot 1 \cdot (0{,}130) = 0{,}026 \implies b_1^1 = -0{,}1 + 0{,}026 = -0{,}074$$
$$\Delta w_{11}^2 = 2 \cdot \alpha \cdot v_1^1 \cdot \delta_1^2 = 2 \cdot (0{,}1) \cdot (0{,}06) \cdot (0{,}103) = 0{,}001 \implies w_{11}^2 = 0{,}1 + 0{,}001 = 0{,}101$$
$$\Delta w_{32}^1 = 2 \cdot \alpha \cdot v_2^0 \cdot \delta_3^1 = 2 \cdot (0{,}1) \cdot (0{,}7) \cdot (0{,}152) = 0{,}021 \implies w_{32}^1 = 0{,}9 + 0{,}021 = 0{,}921$$

3. Processo de Treinamento

• Mas isso foi a atualização de apenas 4 parâmetros da nossa rede e corrigindo apenas para uma única entrada do dataset.

• Esse processo deve ser repetido para todas as entradas do conjunto de treinamento.

• Cada vez que o conjunto de treinamento é passado pela rede e seus erros são retropropagados, dizemos que se passou **1 época**.

• Também podemos escolher passar vários elementos do conjunto de treinamento de cada vez, em *batches*, calcular o erro médio e retropropagar essa média. Essa abordagem diminui a assertividade, porém acelera o treinamento. Definir a quantidade ideal de elementos em cada *batch* é um desafio por si só.

4. Exemplos em Código

- Agora que já vimos a teoria, vamos ver como implementar e aplicar isso utilizando PyTorch, a ferramenta de ML do Facebook.

- Vamos codificar e, durante a codificação, vocês serão apresentados à normalização de batches!

5. Mini-projeto

- Para este projeto, iremos utilizar o dataset do Kaggle que forneci junto com este material, sobre doenças cardíacas.

- Aqui, você irá utilizar apenas as colunas age, chol, thalach e target.

- Separe aleatoriamente e de forma equilibrada o dataset, utilizando 70% como conjunto de treinamento.

- Utilize, também de forma equilibrada, 15% para validação e 15% para teste.

- Implemente o processo de treinamento e validação do modelo, plotando as curvas com as evoluções dos erros de treinamento e de validação.

- Justifique seus critérios de parada de treinamento.

5. Mini-projeto

- Pesquise a respeito de matrizes de confusão para classificação binária, gere uma relativa ao seu conjunto de testes e calcule o máximo de métricas que você julgar interessantes para avaliar seu modelo.

- Dica: https://classeval.wordpress.com/introduction/basic-evaluation-measures/

- Desafio: resolva o problema utilizando todas as *features* da base de dados (pesquise sobre encoding de variáveis categóricas).