Processamento de Linguagem Natural

Redes Neurais

Yuri Malheiros (yuri@ci.ufpb.br)

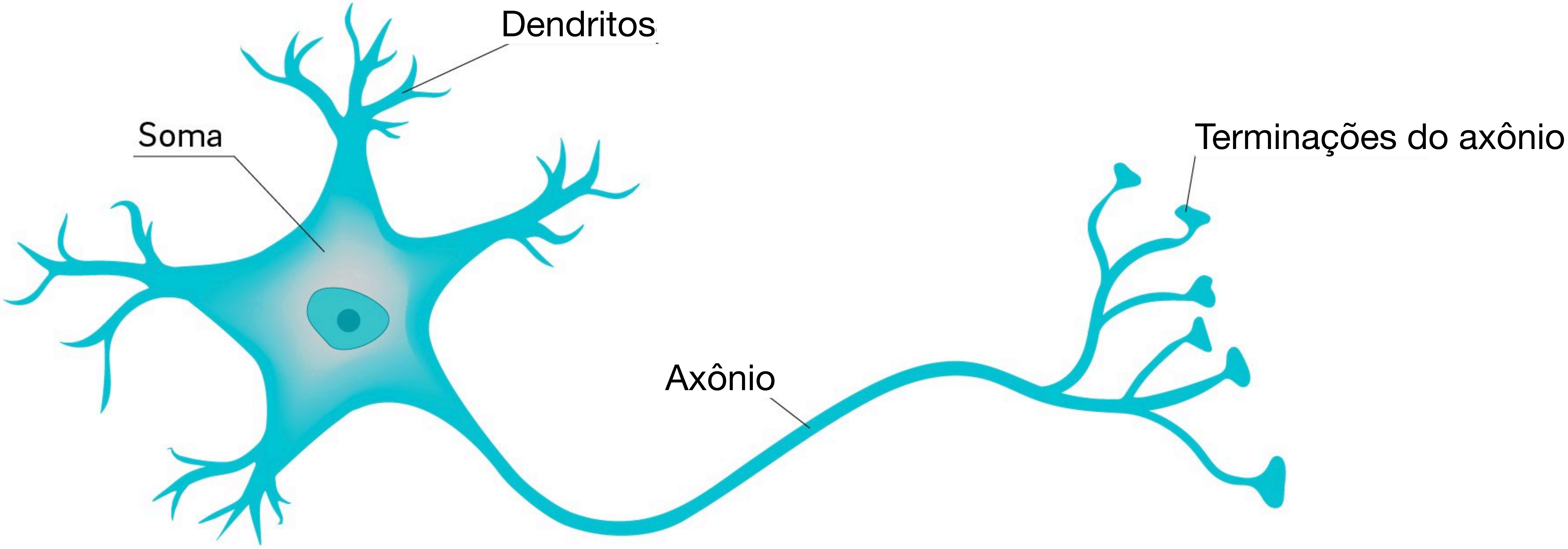
Introdução

- O cérebro é o principal órgão associado à inteligência e aprendizagem

- Na busca pela construção de máquinas inteligentes era natural que o cérebro surgisse como um modelo a ser seguido

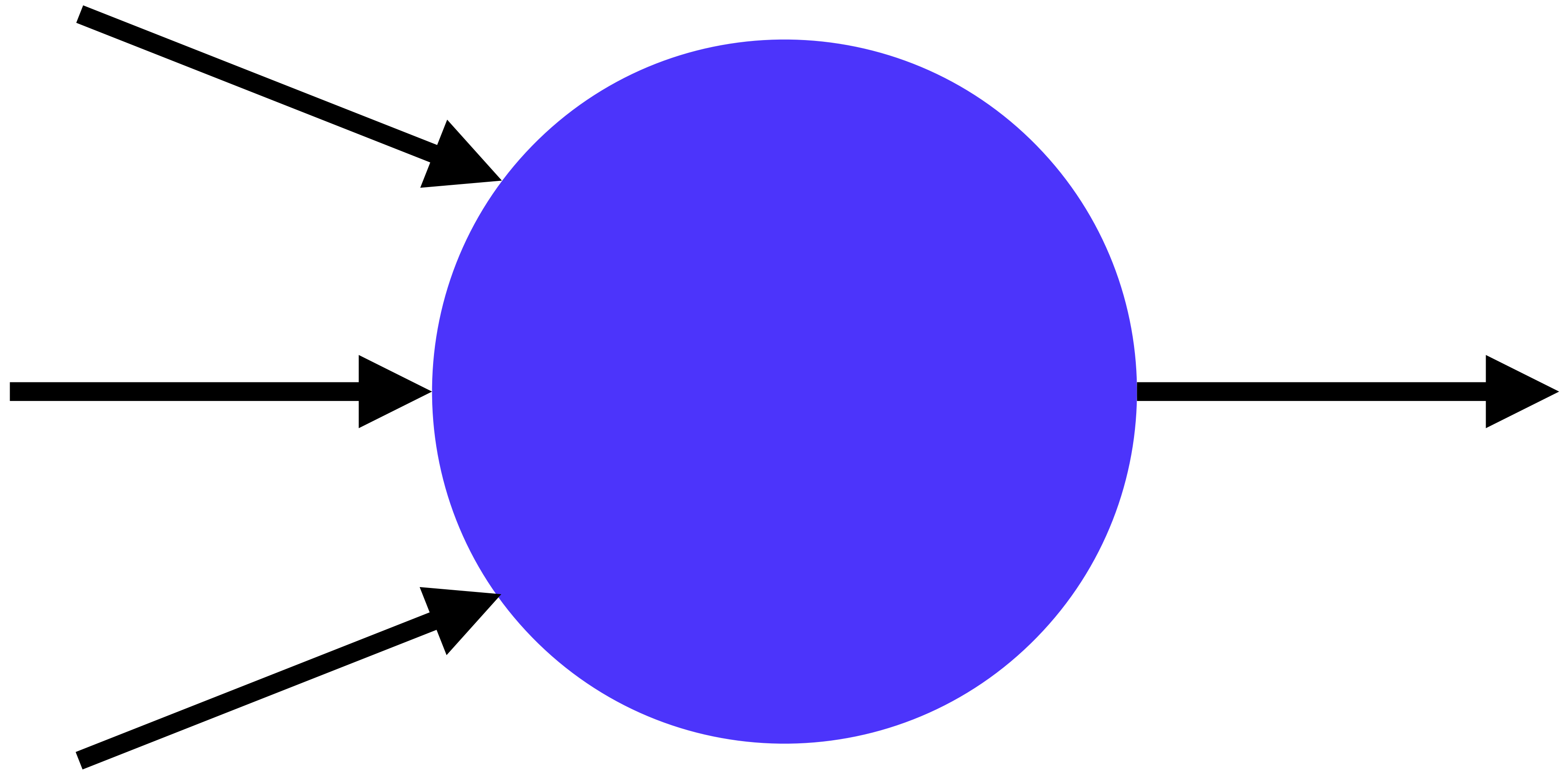
- O cérebro é composto por uma rede complexa de aproximadamente 100 bilhões de neurônios interconectados

Introdução

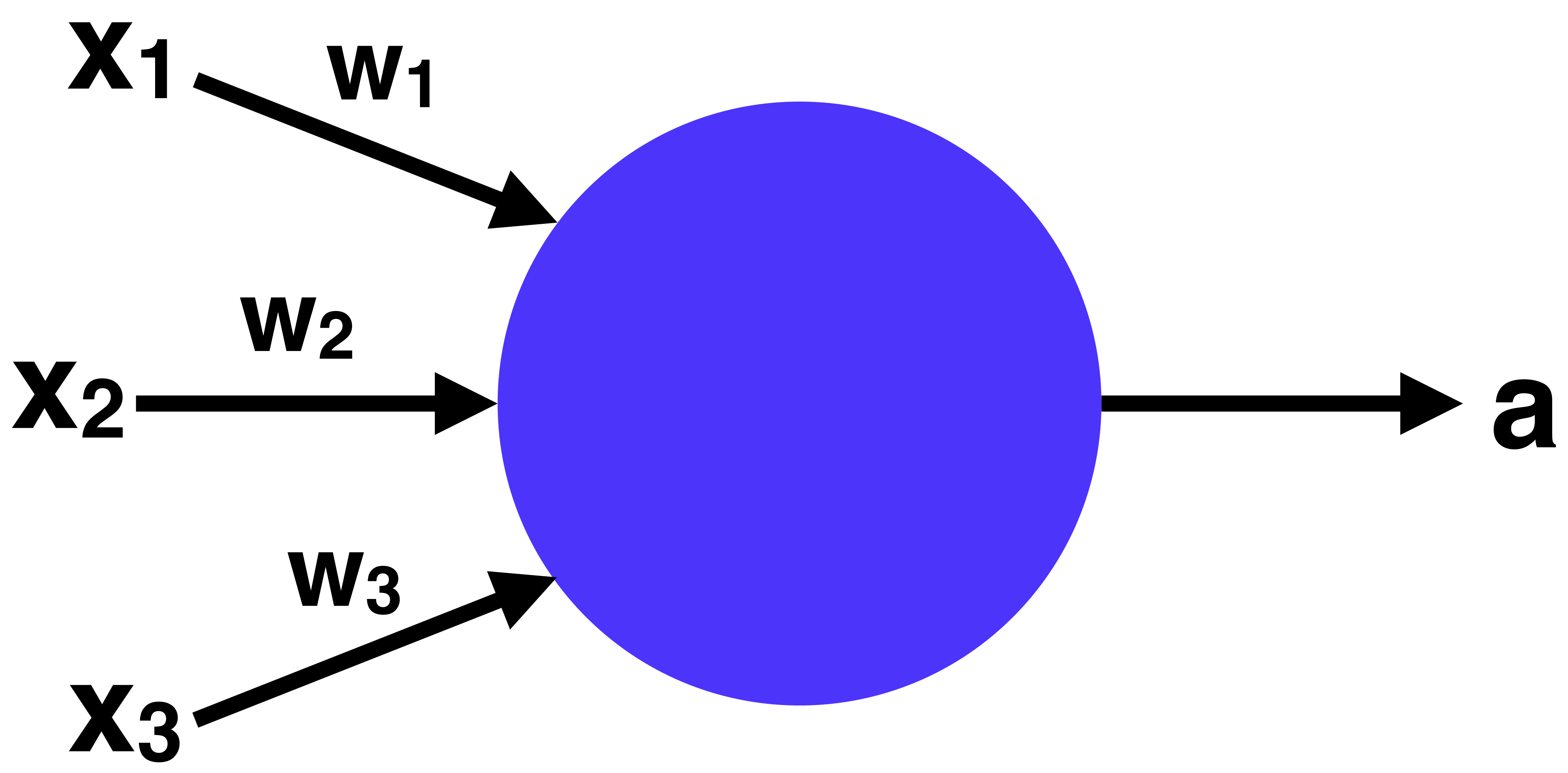


Perceptron

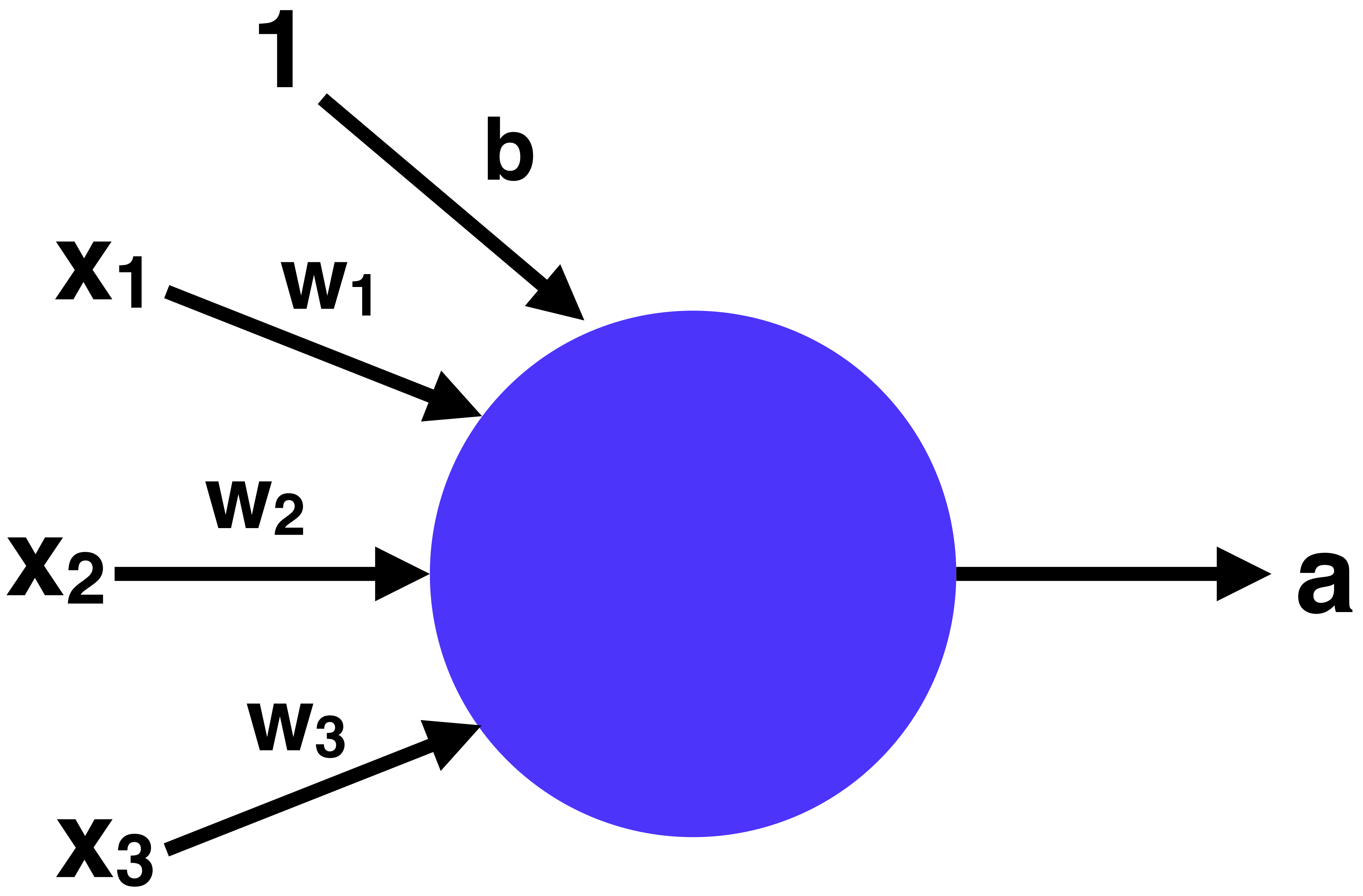
- Em 1943, McCullock e Pitts propuseram um modelo de neurônio artificial: **perceptron**



Perceptron



Perceptron

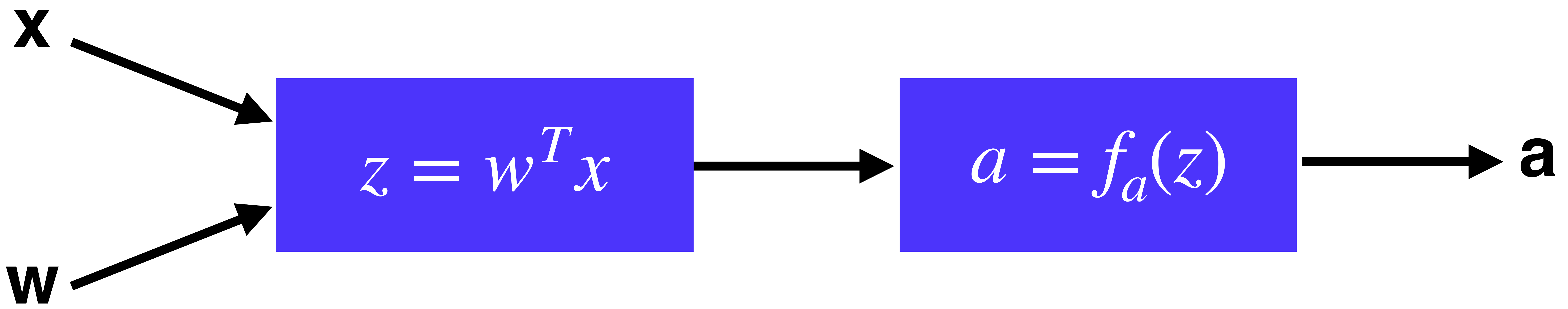


Perceptron

- Considerando que:

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad w = \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Perceptron

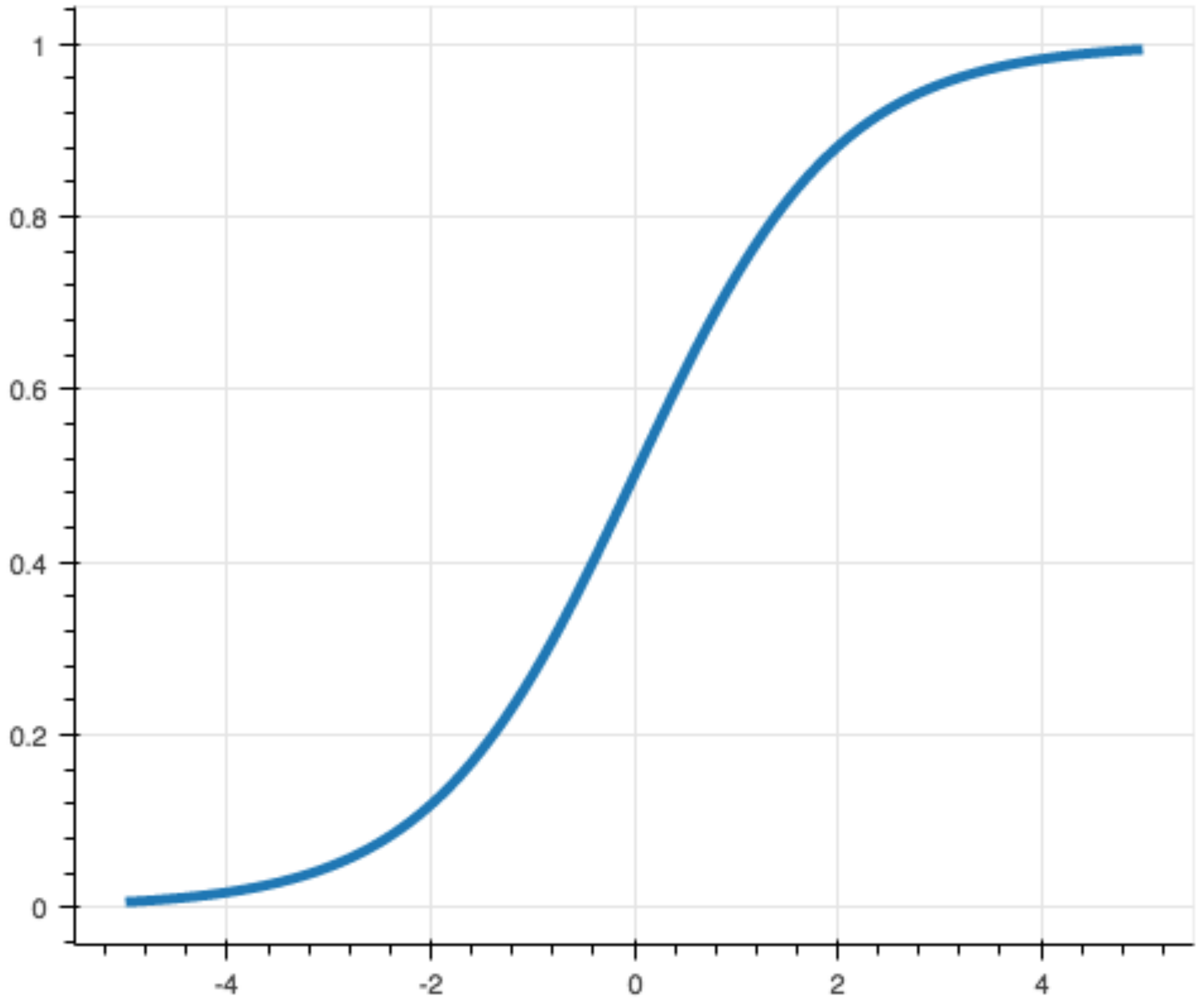


Função de Ativação

- Várias funções podem ser usadas como função de ativação (Fa)

- Função Sigmoide:

$$f_a(z) = \frac{1}{1 + e^{-z}}$$

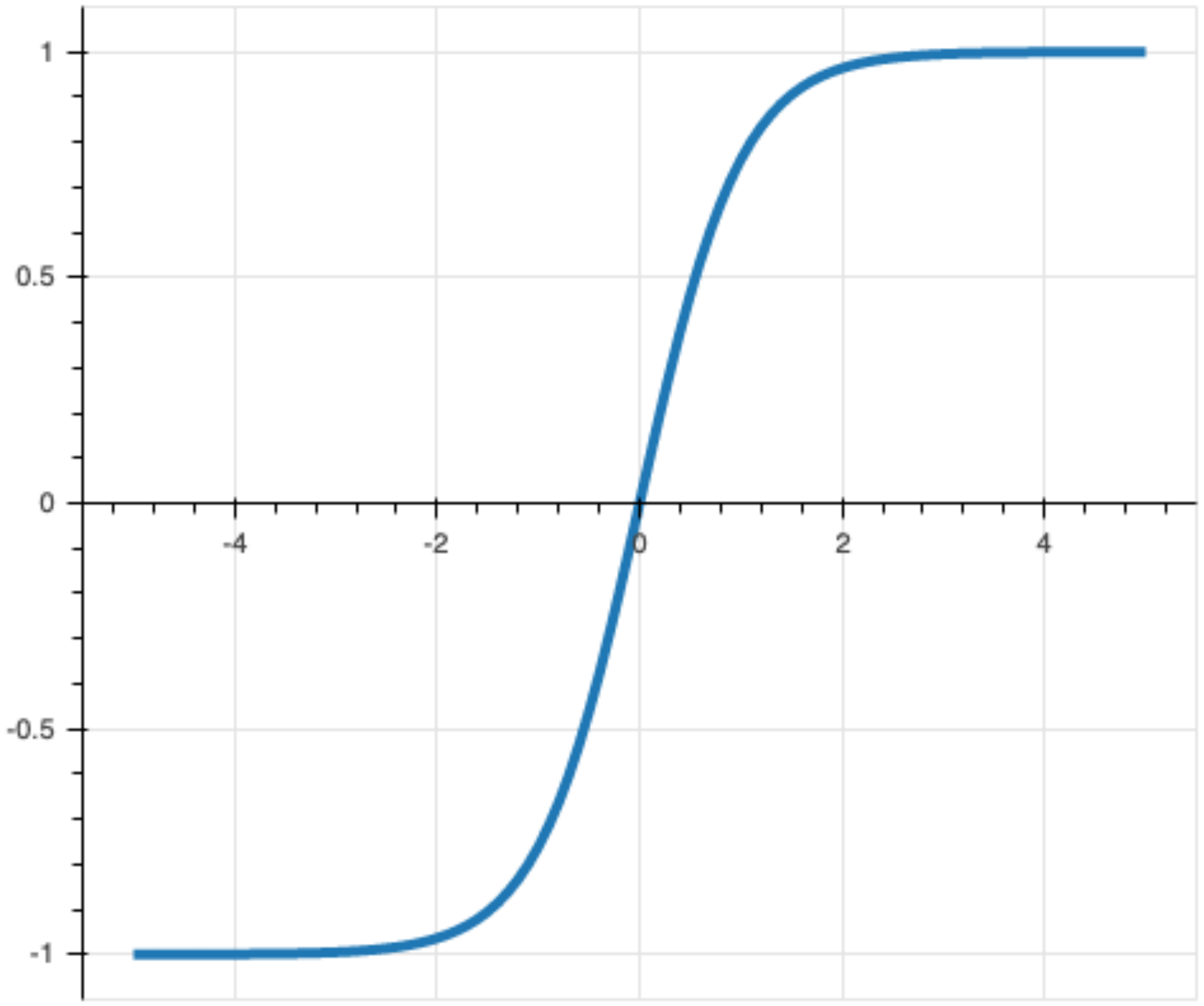


Função de Ativação

- Várias funções podem ser usadas como função de ativação (Fa)

- Função Tangente Hiperbólica:

$$f_a(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$

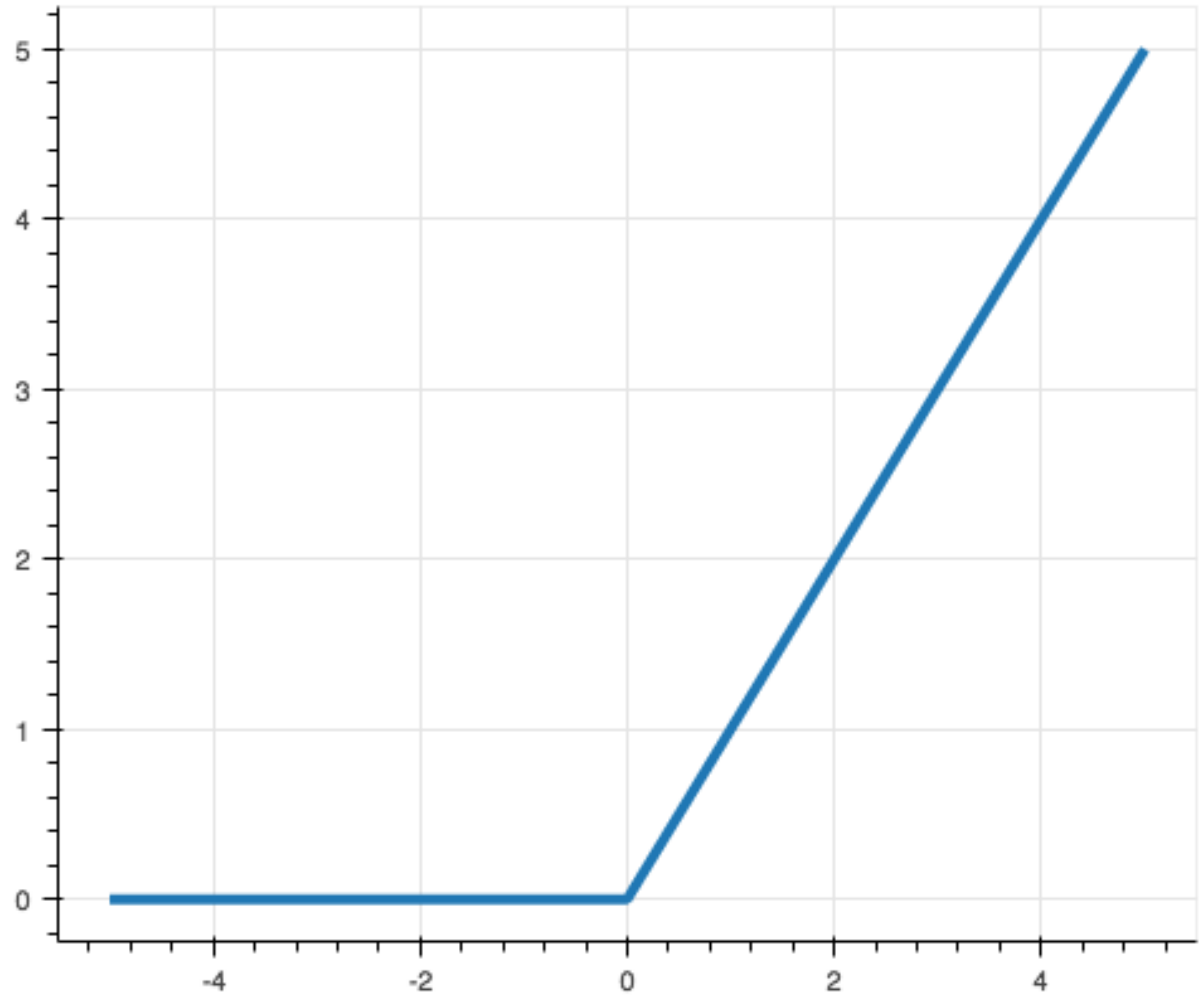


Função de Ativação

- Várias funções podem ser usadas como função de ativação (Fa)

- Função ReLU

$$f_a(z) = max(0,z)$$



Perceptron

- Dados:

$$x = \begin{bmatrix} 1 \\ 0.5 \\ -1 \\ 2 \end{bmatrix} \qquad w = \begin{bmatrix} 0.2 \\ 1 \\ 0.1 \\ 0.5 \end{bmatrix} \qquad f_a(z) = \frac{1}{1 + e^{-z}}$$

- Calcule a saída **a**

Perceptron

$$z = w^T x$$

$$z = b + w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$z = 0.2 \cdot 1 + 1 \cdot 0.5 + 0.1 \cdot (-1) + 0.5 \cdot 2$$

$$z = 1.6$$

Perceptron

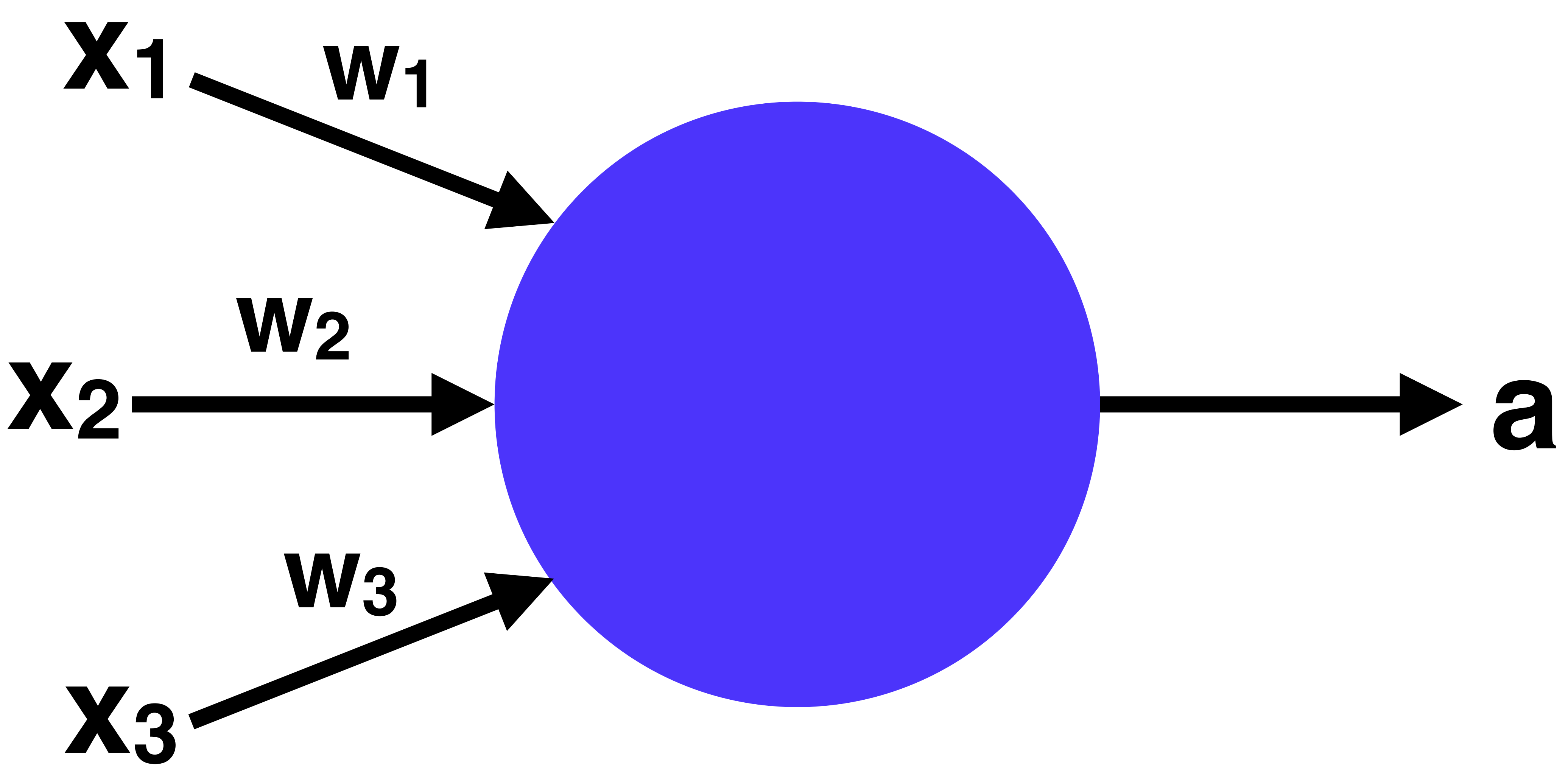
$$f_a(z) = \frac{1}{1 + e^{-z}}$$

$$f_a(1.6) = \frac{1}{1 + e^{-1.6}}$$

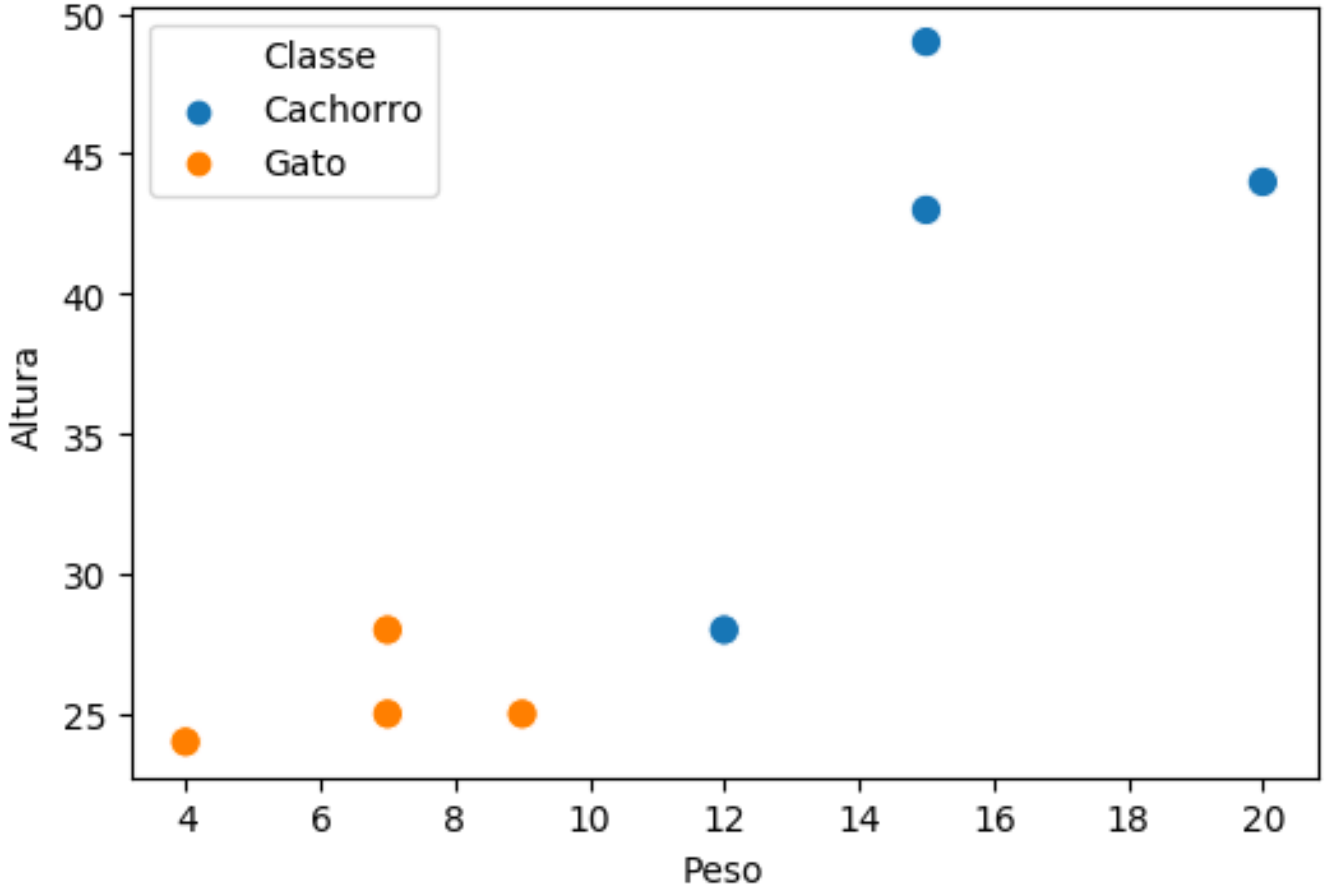
$$f_a(1.6) = 0.8320$$

Perceptron

- A rede perceptron pode ser usada como um classificador
- Apenas uma camada de neurônios
- Podemos usá-la na aprendizagem de máquina supervisionada



Exemplo



	Peso	Altura	Classe
0	20	44	Cachorro
1	15	43	Cachorro
2	12	28	Cachorro
3	15	49	Cachorro
4	7	28	Gato
5	7	25	Gato
6	9	25	Gato
7	4	24	Gato

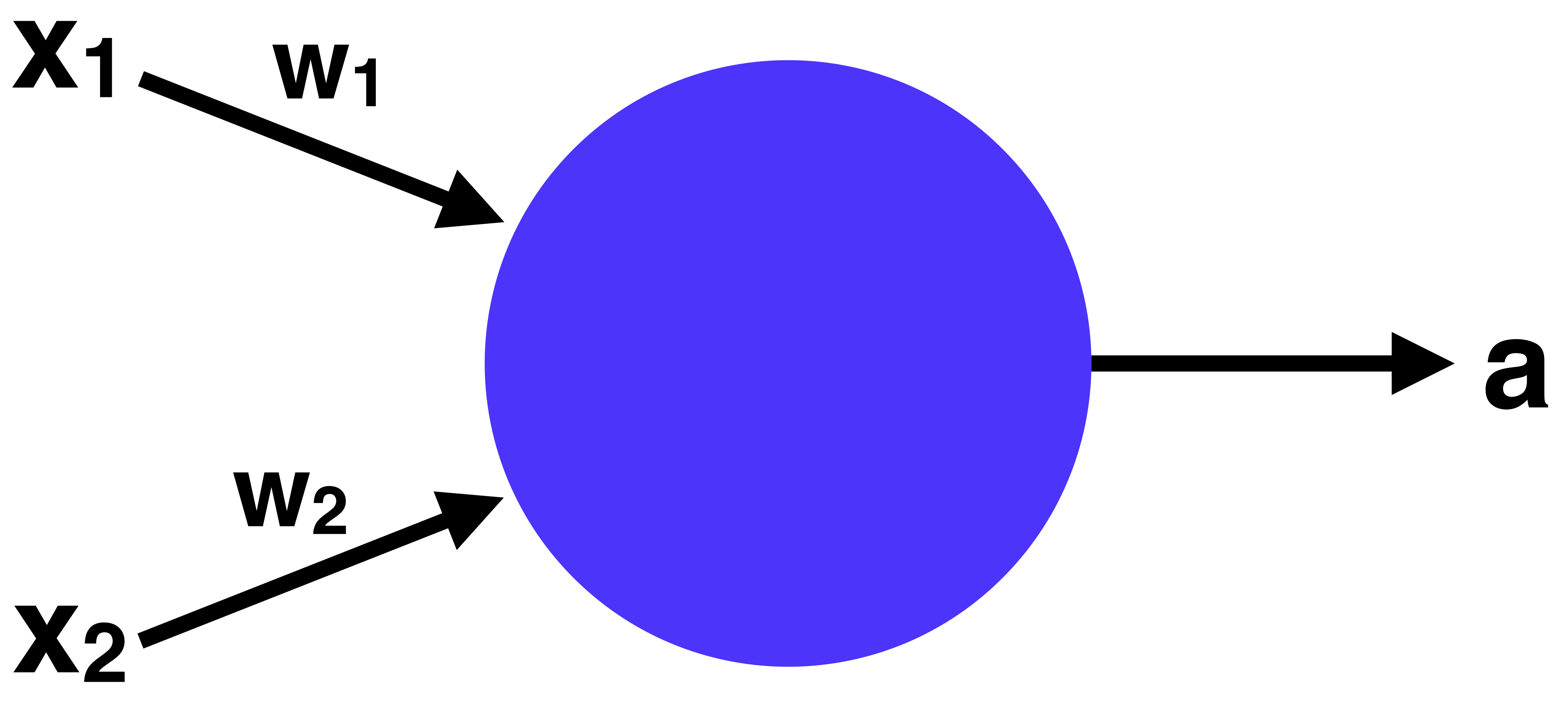
Exemplo

- Dada a altura e o peso, vamos fazer o neurônio ter:

- Saída **0** se for um gato
- Saída **1** se for um cachorro

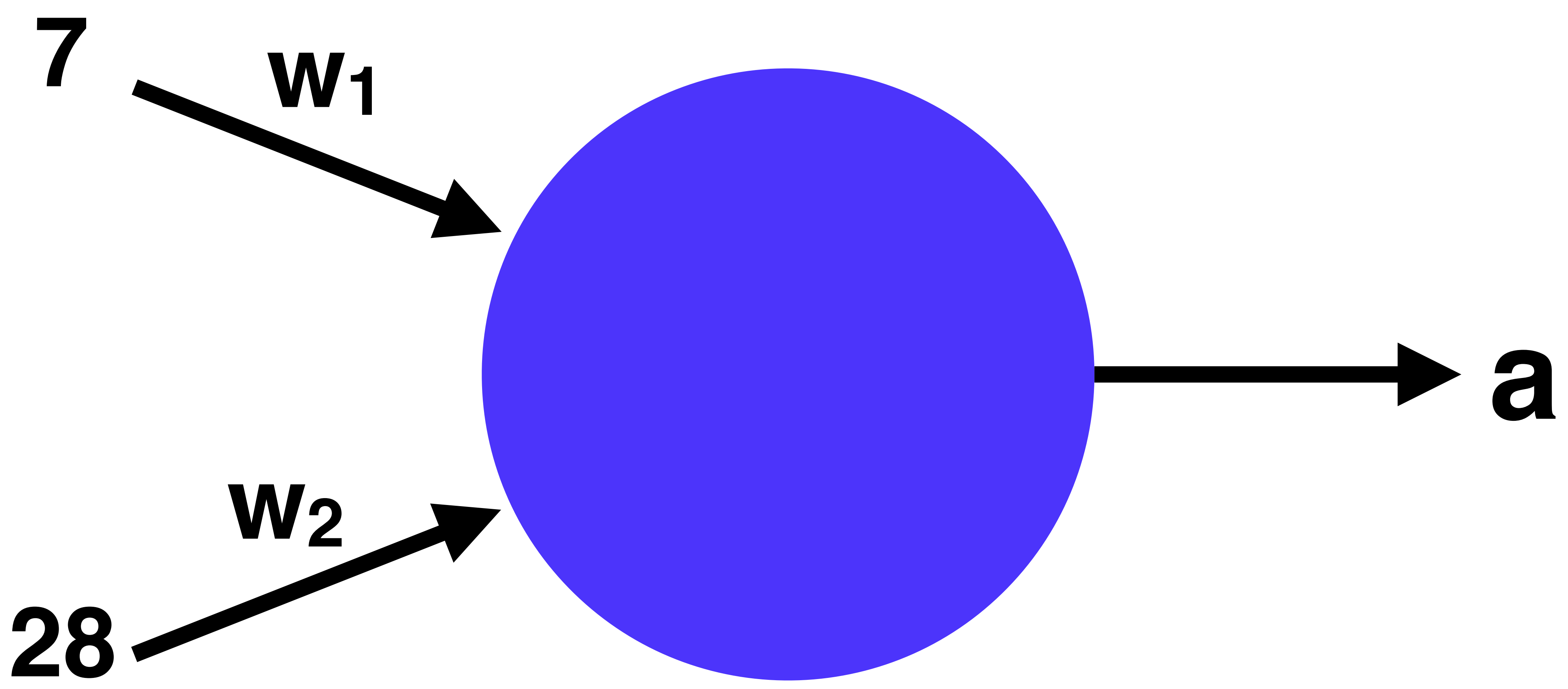
Exemplo

- x_1 - peso
- x_2 - altura



Exemplo

- x_1 - peso
- x_2 - altura



$$z = w_1 \cdot 7 + w_2 \cdot 28$$

Exemplo

- Quais são os valores de **w**?

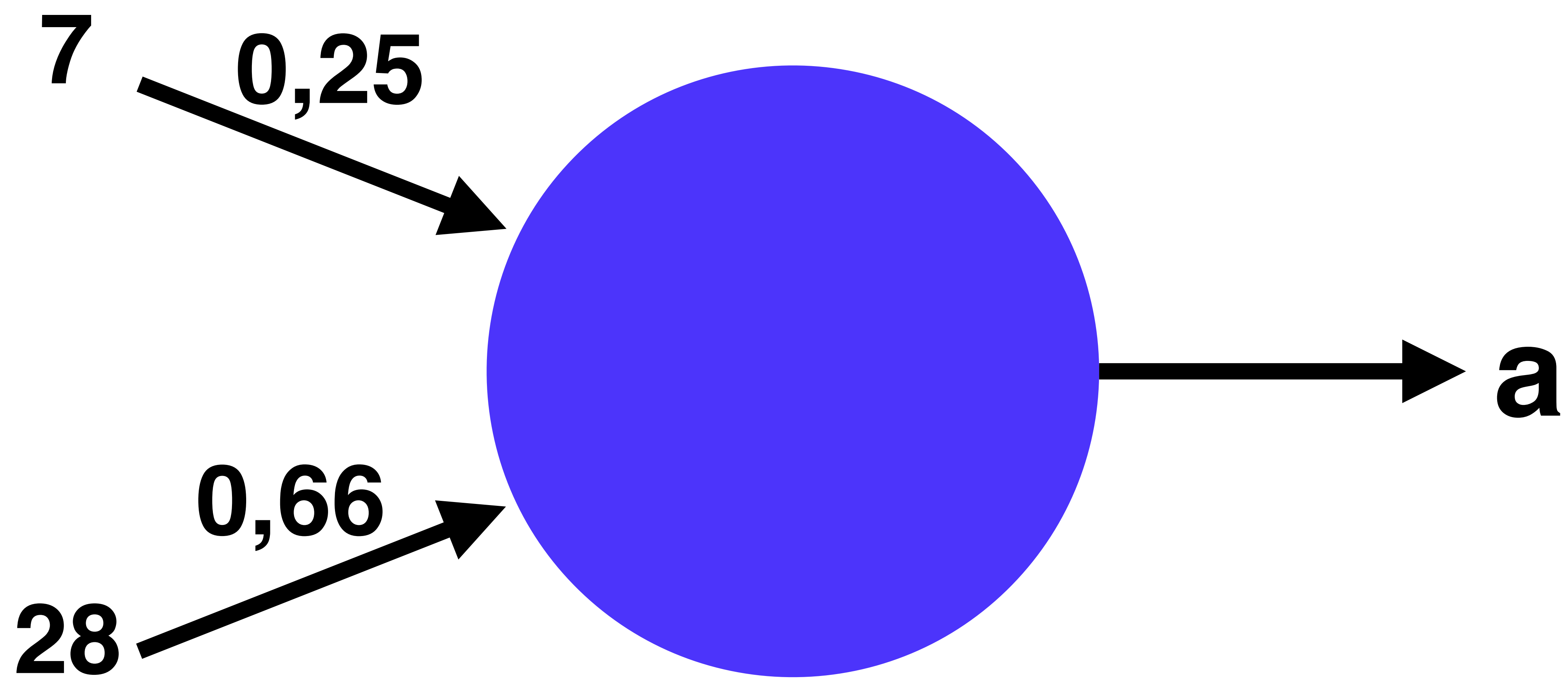
- Podemos tentar encontrá-los manualmente, porém este é um processo complexo e muitas vezes inviável

Treinando uma Rede Neural

- Como não sabemos os pesos iniciais, vamos iniciá-los aleatoriamente
  - $w_1 = 0,25$
  - $w_2 = 0,66$

- Em seguida, vamos calcular a saída, usando a função de ativação sigmoide, para
  - $x_1 = 7$
  - $x_2 = 28$

Treinando uma Rede Neural



$$z = 0{,}25 \cdot 7 + 0{,}66 \cdot 28 = 20{,}23$$

$$a = f(20{,}23) = 0{,}99$$

Treinando uma Rede Neural

- A saída correta para a classe gato é 0, mas a rede retornou algo muito próximo de 1

- Precisamos ajustar o peso para que esse erro não aconteça ou pelo menos que ele diminua

Treinando uma Rede Neural

- Nas redes neurais, durante o treinamento, os pesos **w** serão atualizados para que a rede consiga acertar o máximo possível a classificação para os dados de treinamento
- Para encontrar os melhores valores de **w**, vamos utilizar o algoritmo de descida de gradiente

Treinando uma Rede Neural

- Na descida de gradiente é necessário saber se o algoritmo está retornando bons resultados para um determinado **w**
- Um bom valor é um valor que faz com que o erro seja baixo
- Como calcular o erro de classificação de uma rede neural?

Treinando uma Rede Neural

- Usando a função sigmoide os valores retornados poderão variar em todo intervalo entre 0 e 1
- Podemos interpretar que quanto mais perto do valor de uma das classes, mais confiante a rede é para aquela classificação
- Assim, podemos calcular o erro da rede como:

$$J(w) = \frac{1}{2}(a - a_{esperado})^2$$

Treinando uma Rede Neural

- Na descida de gradiente, precisamos saber se devemos aumentar ou diminuir o valor de **w** para que o erro diminua
- Para isso precisamos computar o gradiente, que indica como o erro muda de acordo com um valor de **w**:

$$\frac{\partial J}{\partial w_i}$$

Treinando uma Rede Neural

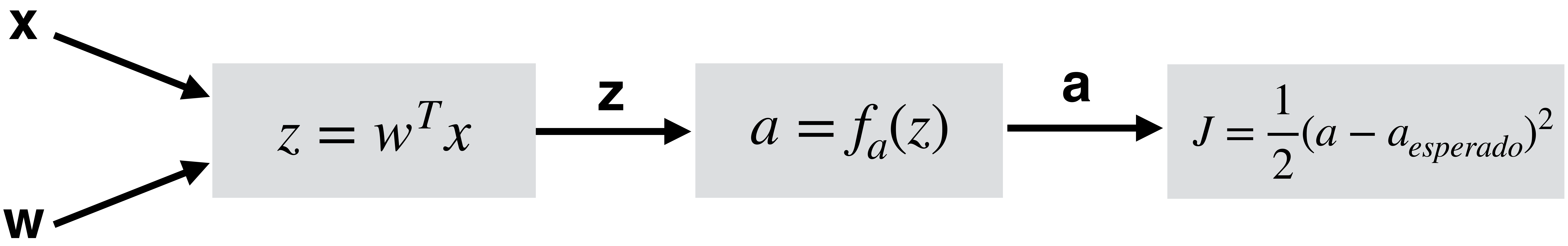
- Para cada exemplo dos dados de treinamento, atualizamos os valores de **w_i** da seguinte forma:

$$w_i := w_i - \alpha \frac{\partial J}{\partial w_i}$$

- Onde α é a taxa de aprendizagem

Treinando uma Rede Neural

- Como calcular $\dfrac{\partial J}{\partial w_i}$?



Treinando uma Rede Neural

- Como calcular $\frac{\partial J}{\partial w_i}$?

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_i}$$

Treinando uma Rede Neural

$$\frac{\partial J}{\partial a} = a - a_{esperado}$$

$$\frac{\partial a}{\partial z} = a(1 - a)$$

$$\frac{\partial z}{\partial w_i} = x_i$$

Treinando uma Rede Neural

$$\frac{\partial J}{\partial w_i} = (a - a_{esperado}) \cdot a(1 - a) \cdot x_i$$

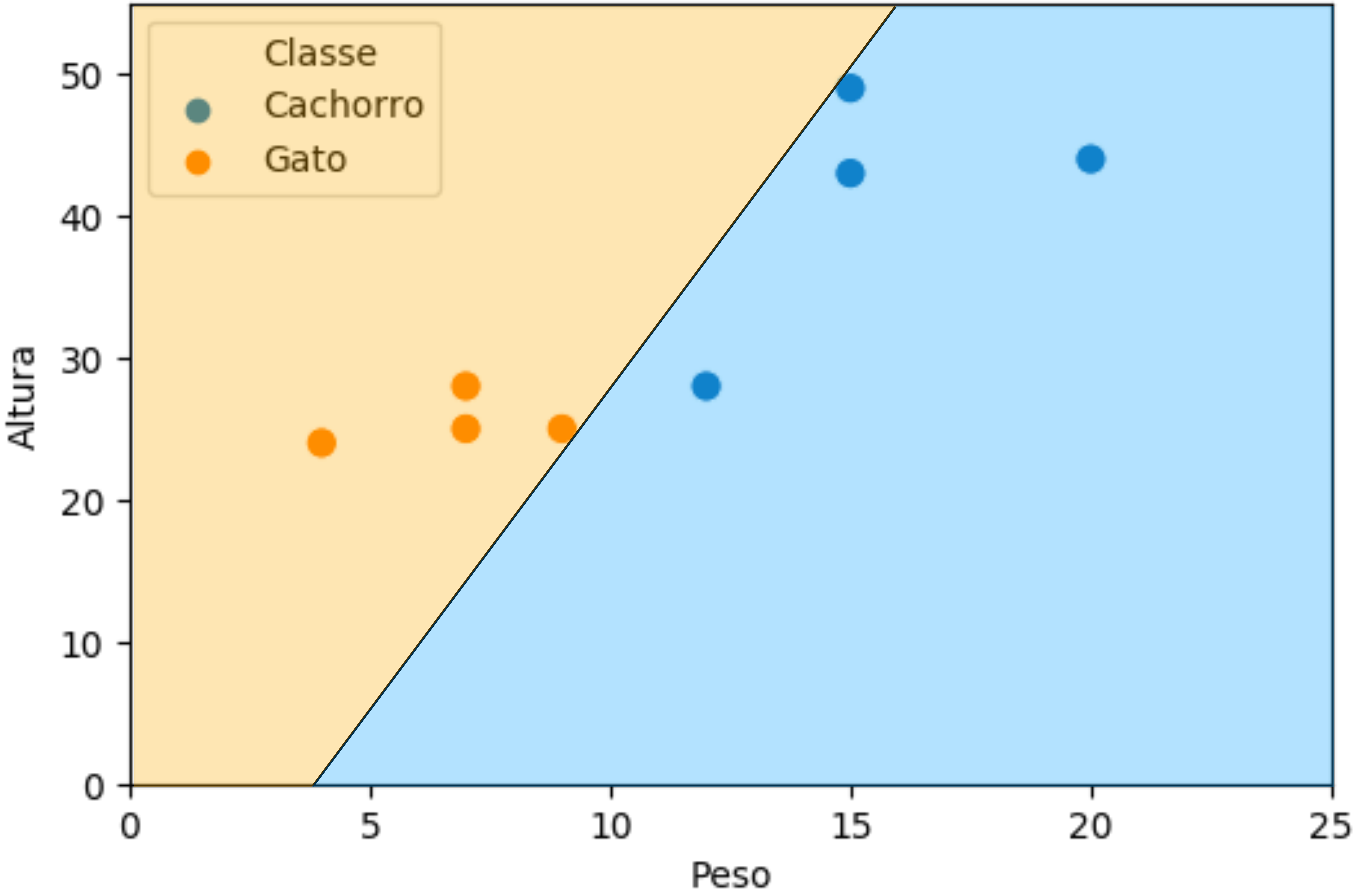
Treinando uma Rede Neural

- Em muitos casos, fazer uma atualização para cada exemplo pode não ser suficiente

- Por isso, costumamos atualizar para cada exemplo e em seguida refazer o processo várias vezes

- Esse número de vezes é chamado de **épocas**

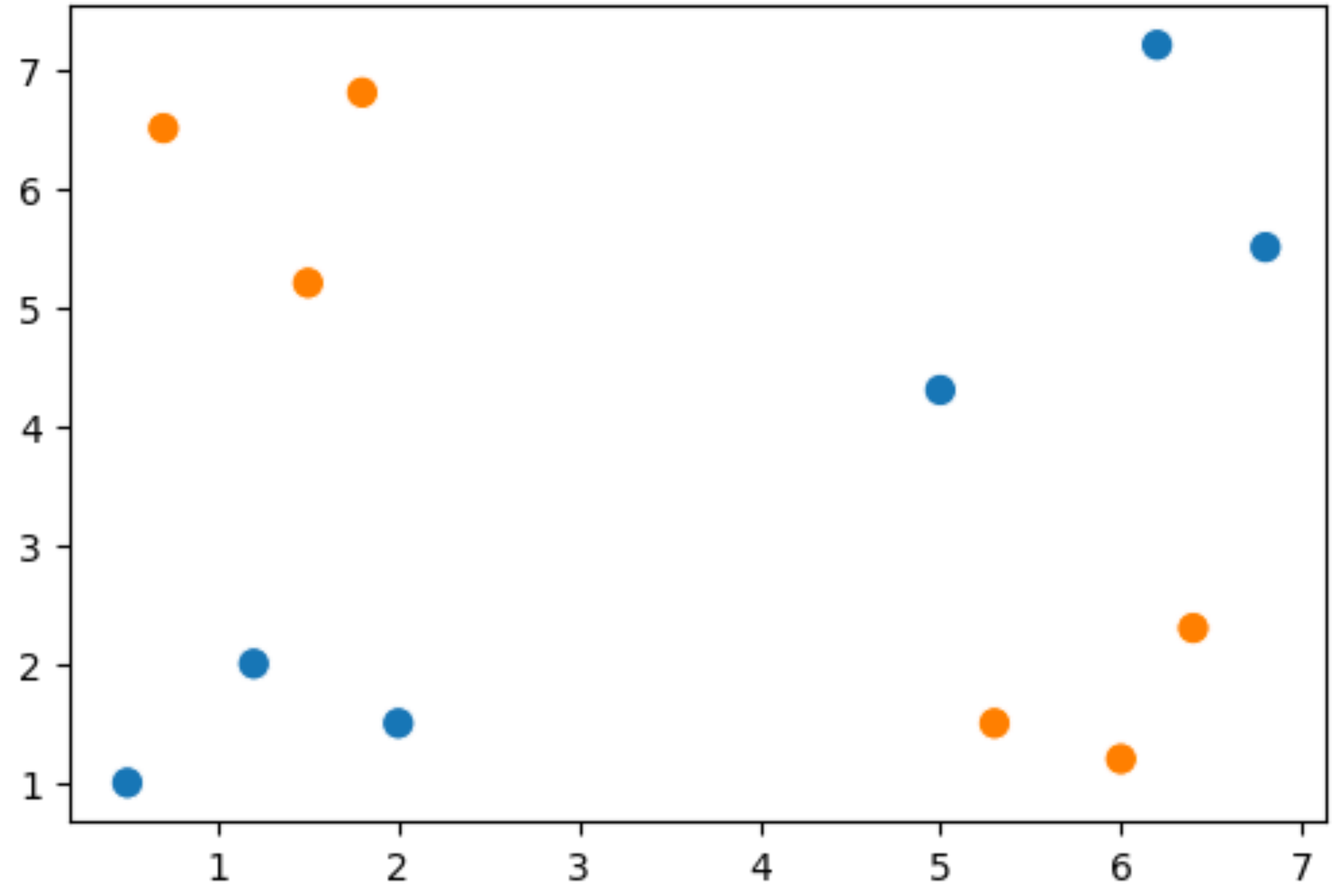
Treinando uma Rede Neural



Peceptron

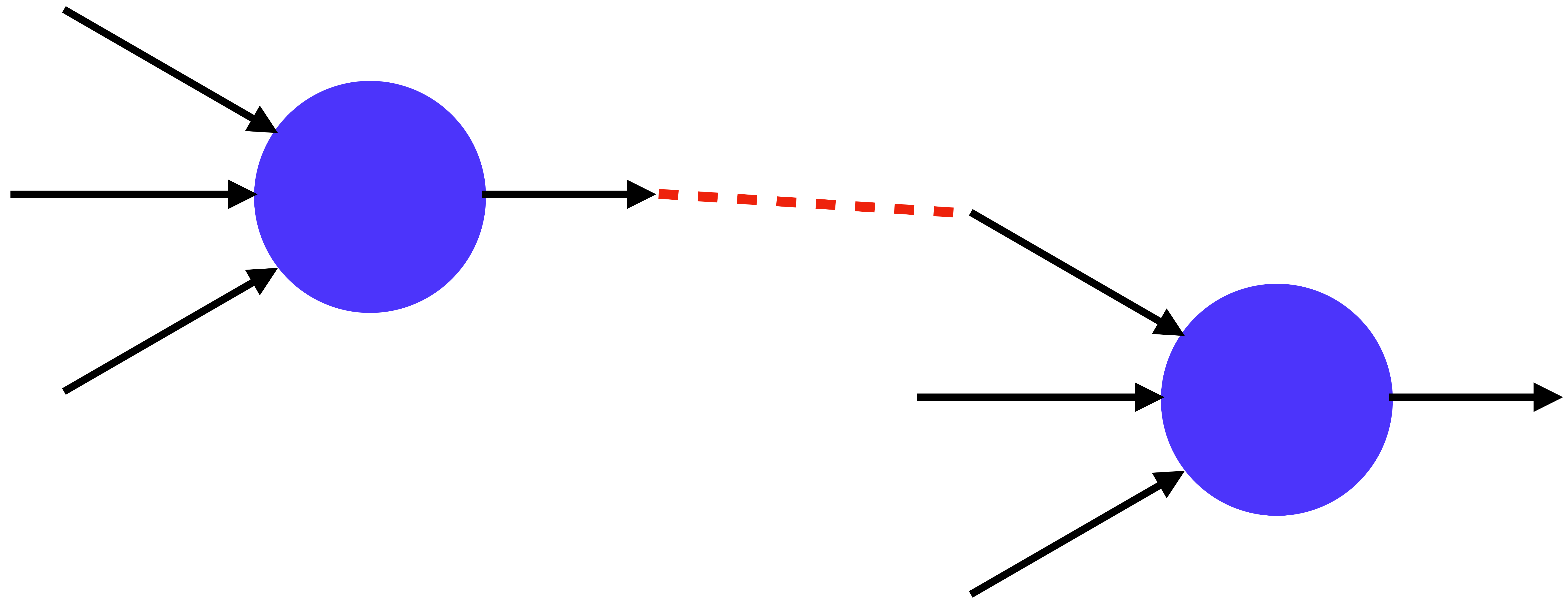
- Como vimos, o perceptron possui a limitação de criar apenas fronteiras lineares

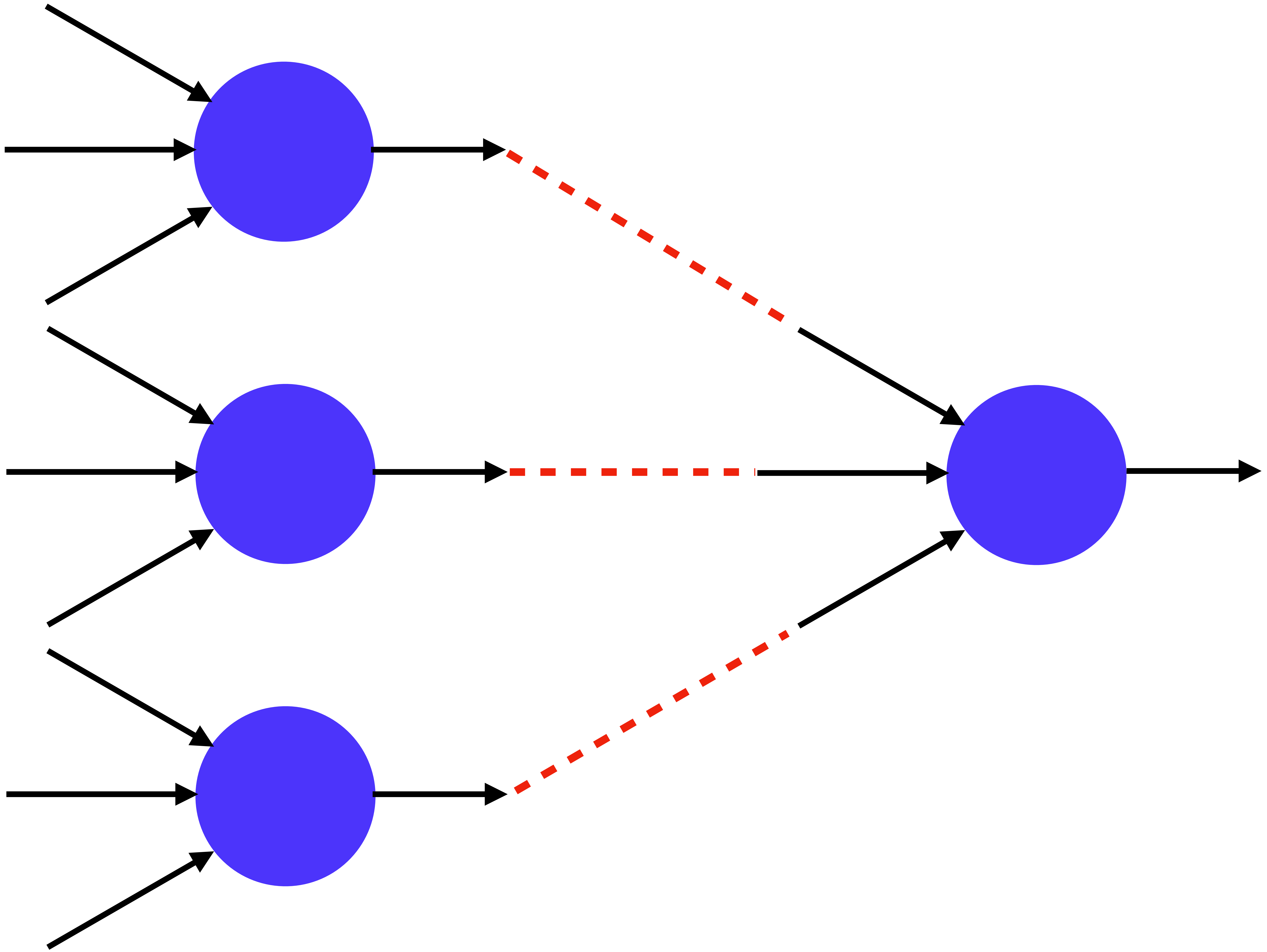
- Ele não consegue classificar bem um conjunto de dados dessa forma:

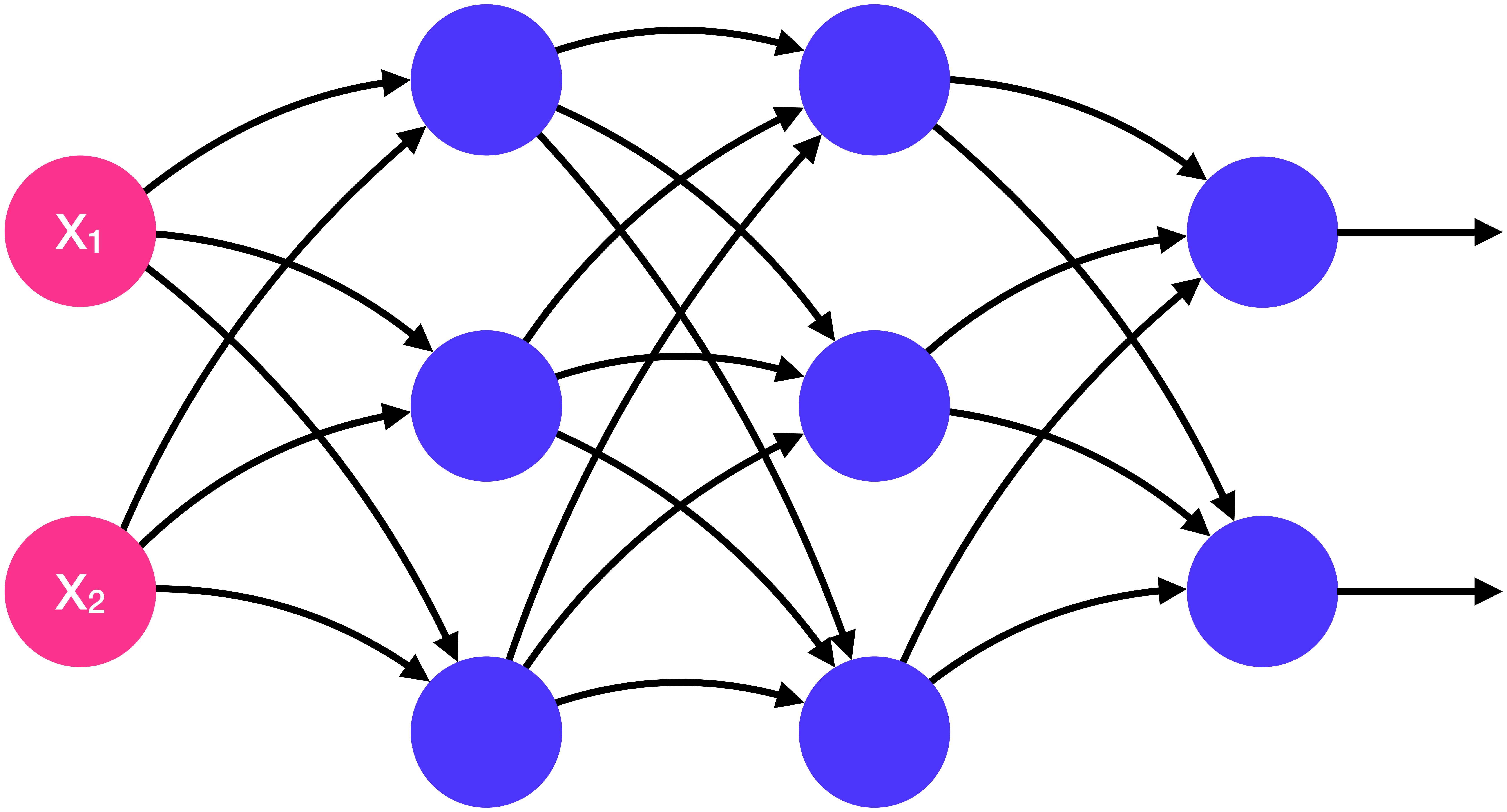


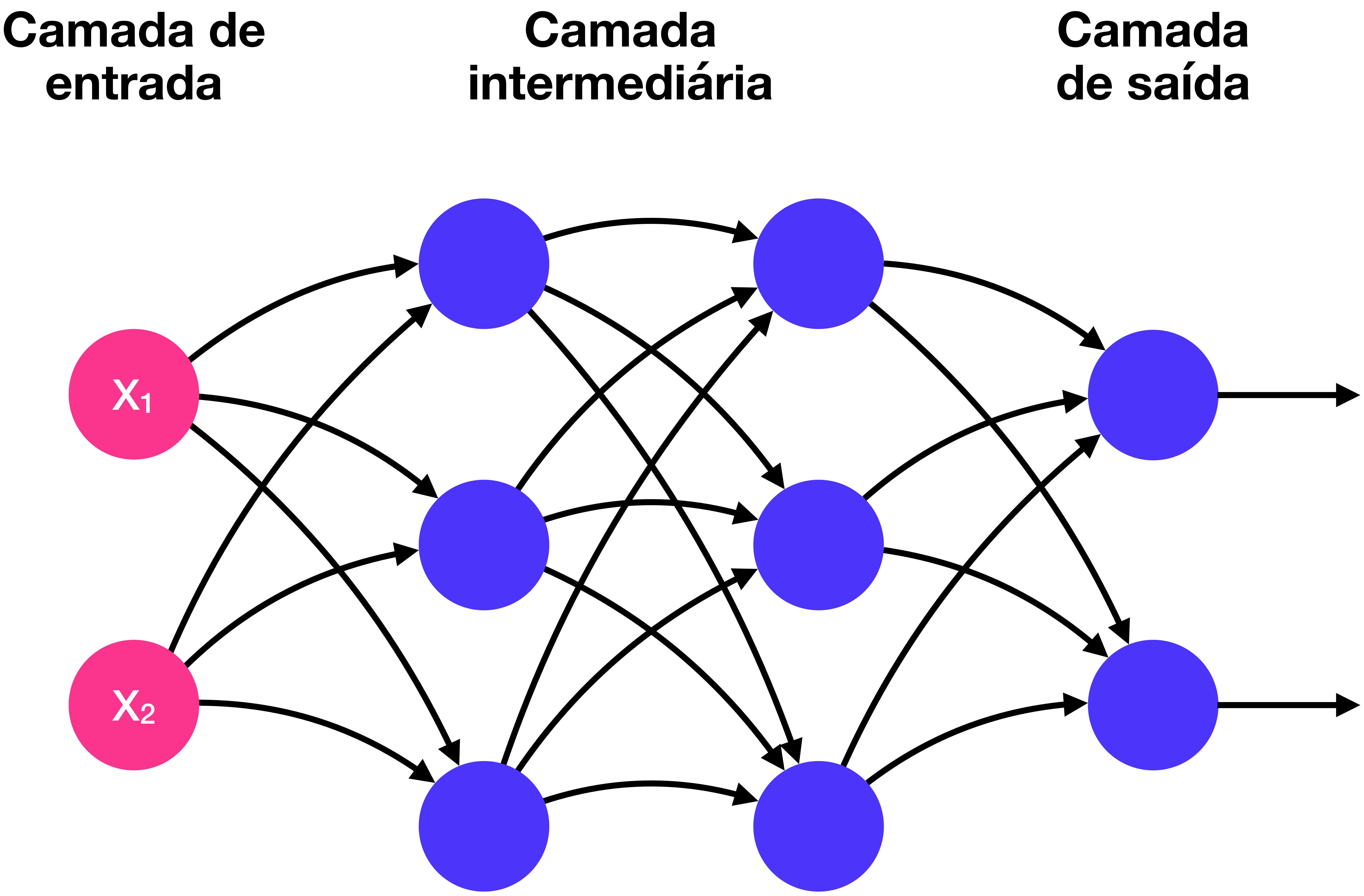
Redes Neurais

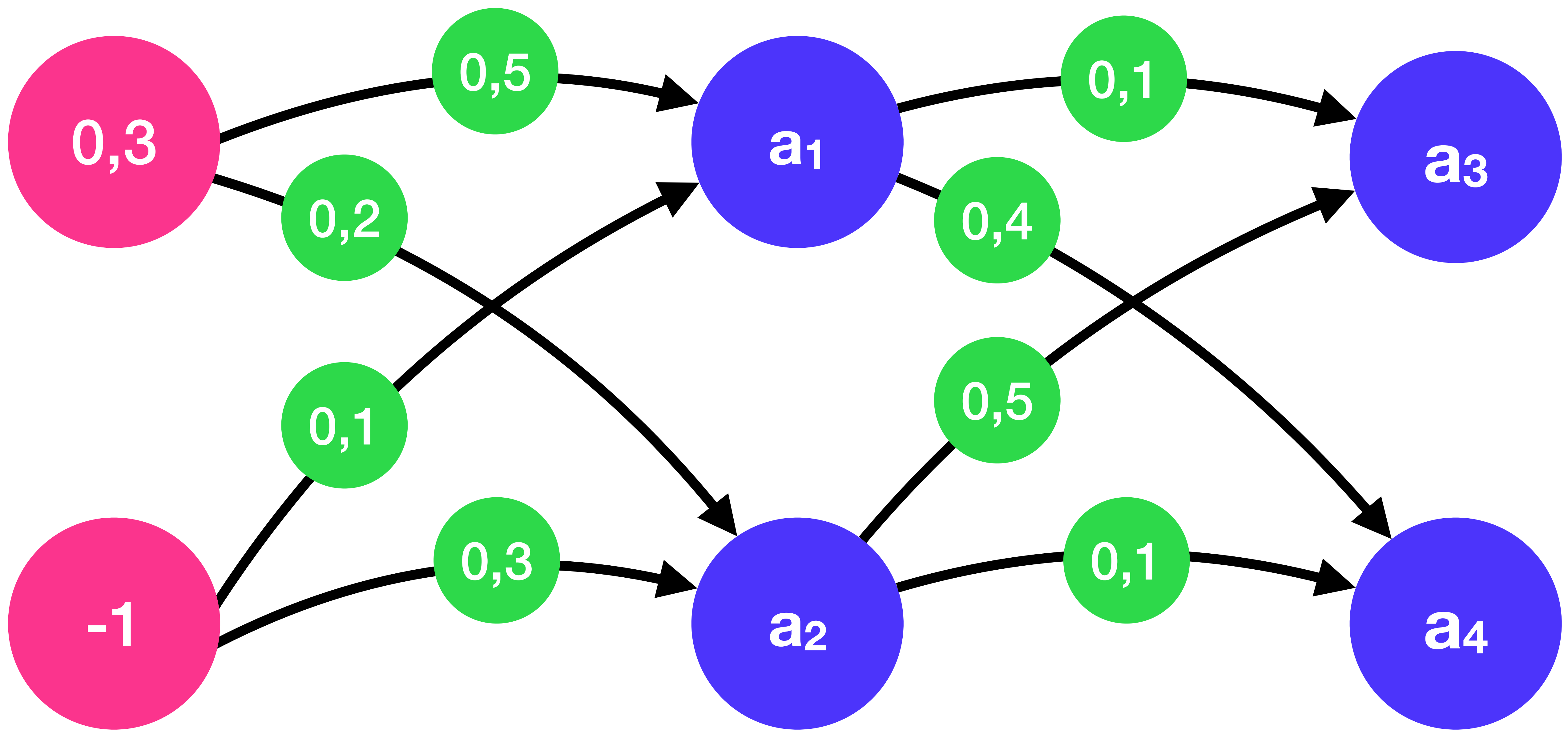
- Para resolver isso, podemos agrupar os neurônios

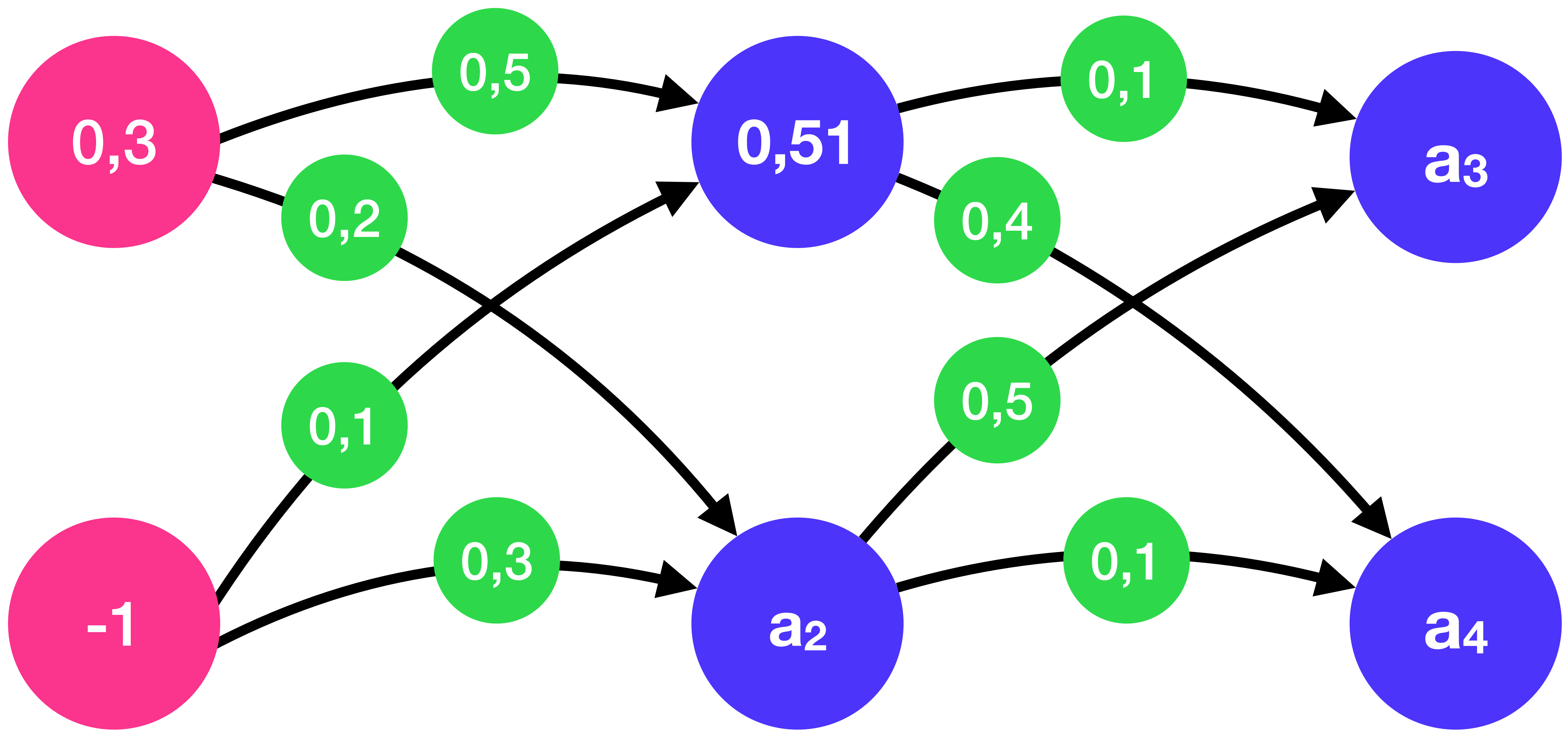





X_1
X_2

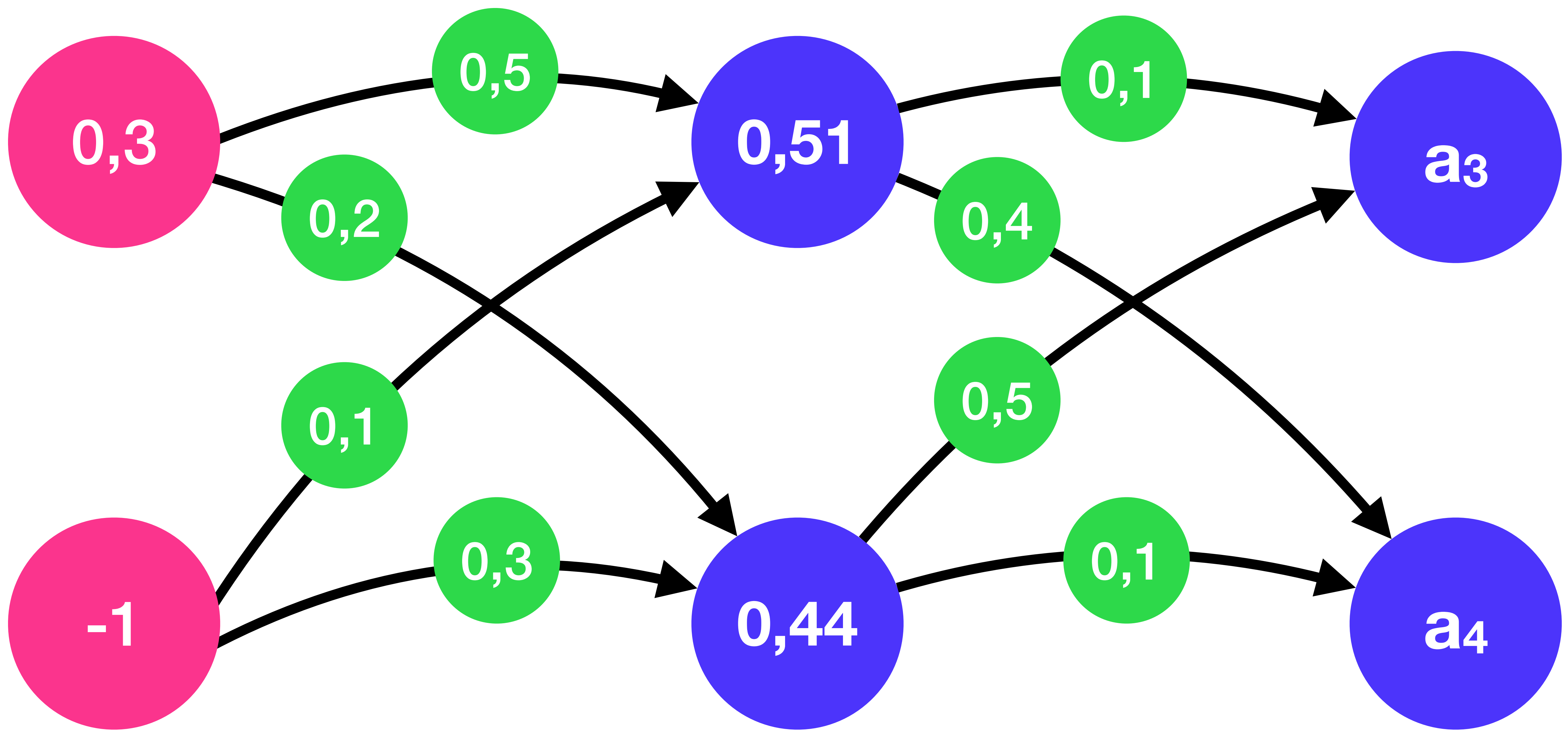

Camada de entrada
Camada intermediária
Camada de saída
X1
X2


0,3
-1
0,5
0,2
0,1
0,3
a1
a2
0,1
0,4
0,5
0,1
a3
a4



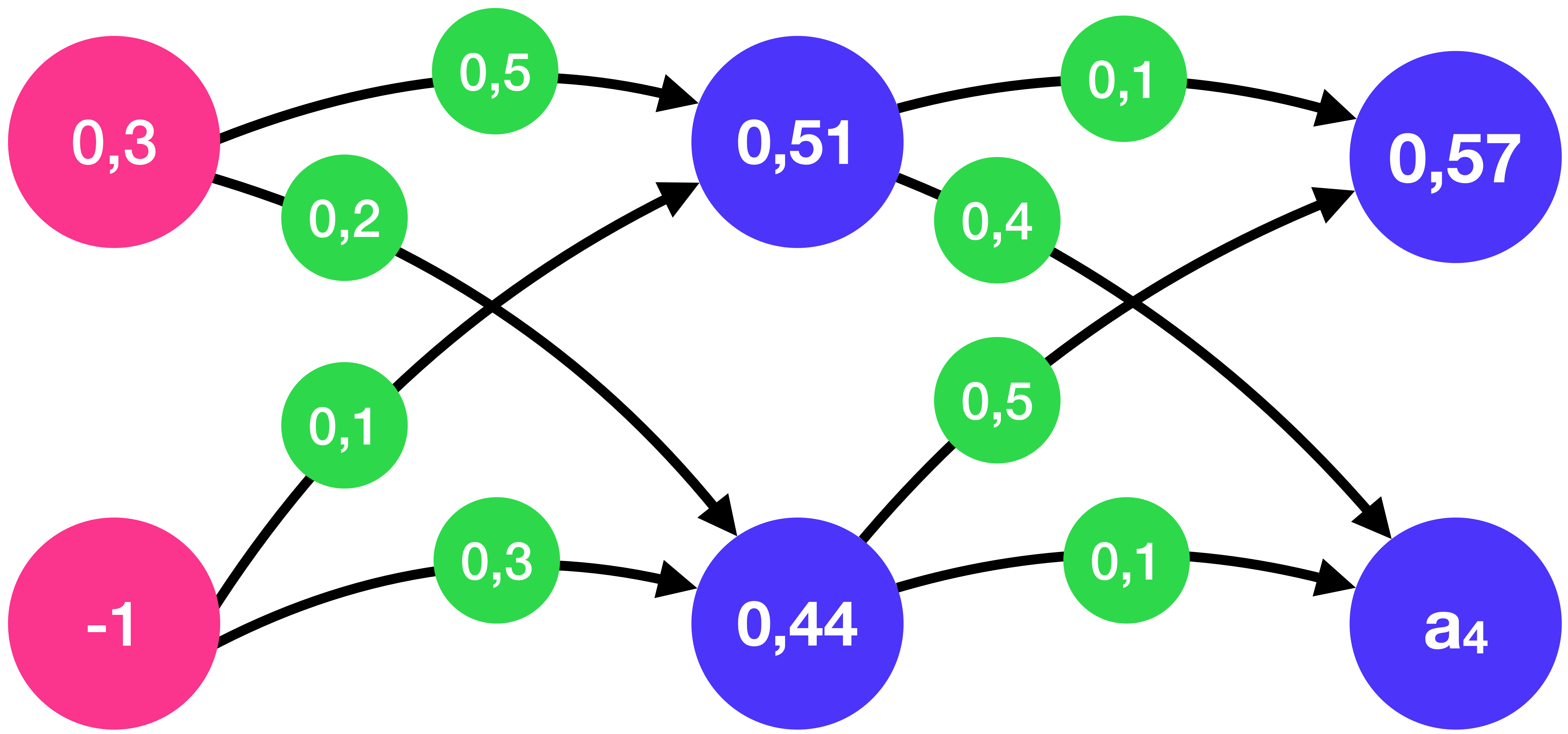
$$z_1 = 0{,}5 \cdot 0{,}3 + 0{,}1 \cdot -1 = 0{,}05$$

$$a_1 = f(0{,}05) = 0{,}51$$



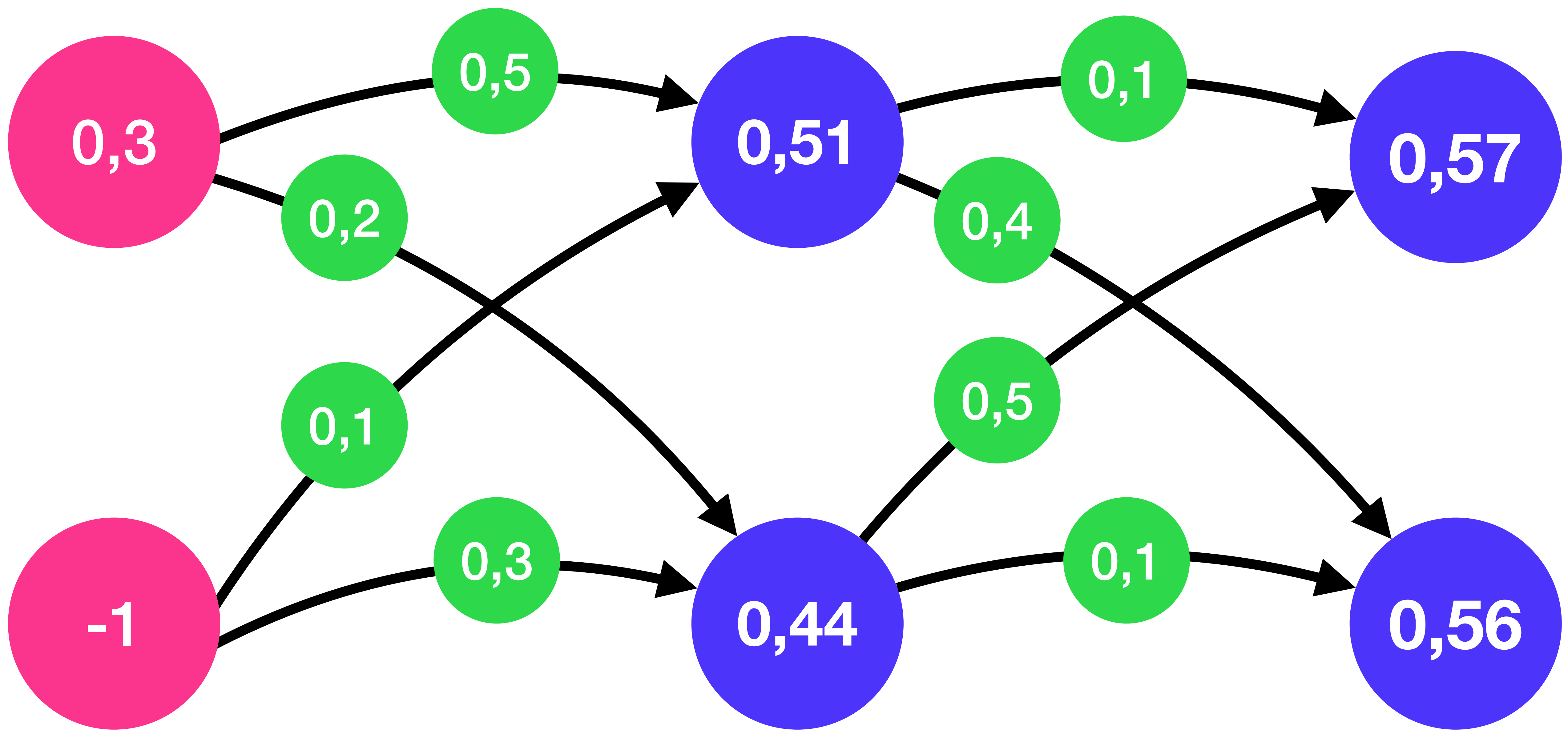
$$z_2 = 0{,}2 \cdot 0{,}3 + 0{,}3 \cdot -1 = -0{,}24$$

$$a_2 = f(-0{,}24) = 0{,}44$$



$$z_3 = 0{,}1 \cdot 0{,}51 + 0{,}5 \cdot 0{,}44 = 0{,}271$$

$$a_3 = f(0{,}271) = 0{,}57$$



$$z_4 = 0{,}4 \cdot 0{,}51 + 0{,}1 \cdot 0{,}44 = 0{,}248$$

$$a_4 = f(0{,}248) = 0{,}56$$

Redes Neurais

- Para treinar uma rede com multicamadas, vamos usar o algoritmo chamado **backpropagation**
- O backpropagation expande a ideia do treinamento da rede perceptron
- O algoritmo tem esse nome, pois ele começa pela camada de saída e vai propagando o erro para as camadas anteriores da rede

Redes Neurais

- Mais uma vez vamos calcular o gradiente em relação aos pesos **w** da rede para descobrir em que sentido precisamos ajustá-los
- Sabendo o gradiente, aplicamos a atualização do valor de **w** como anteriormente:

$$w_i := w_i - \alpha \frac{\partial J}{\partial w_i}$$

Redes Neurais

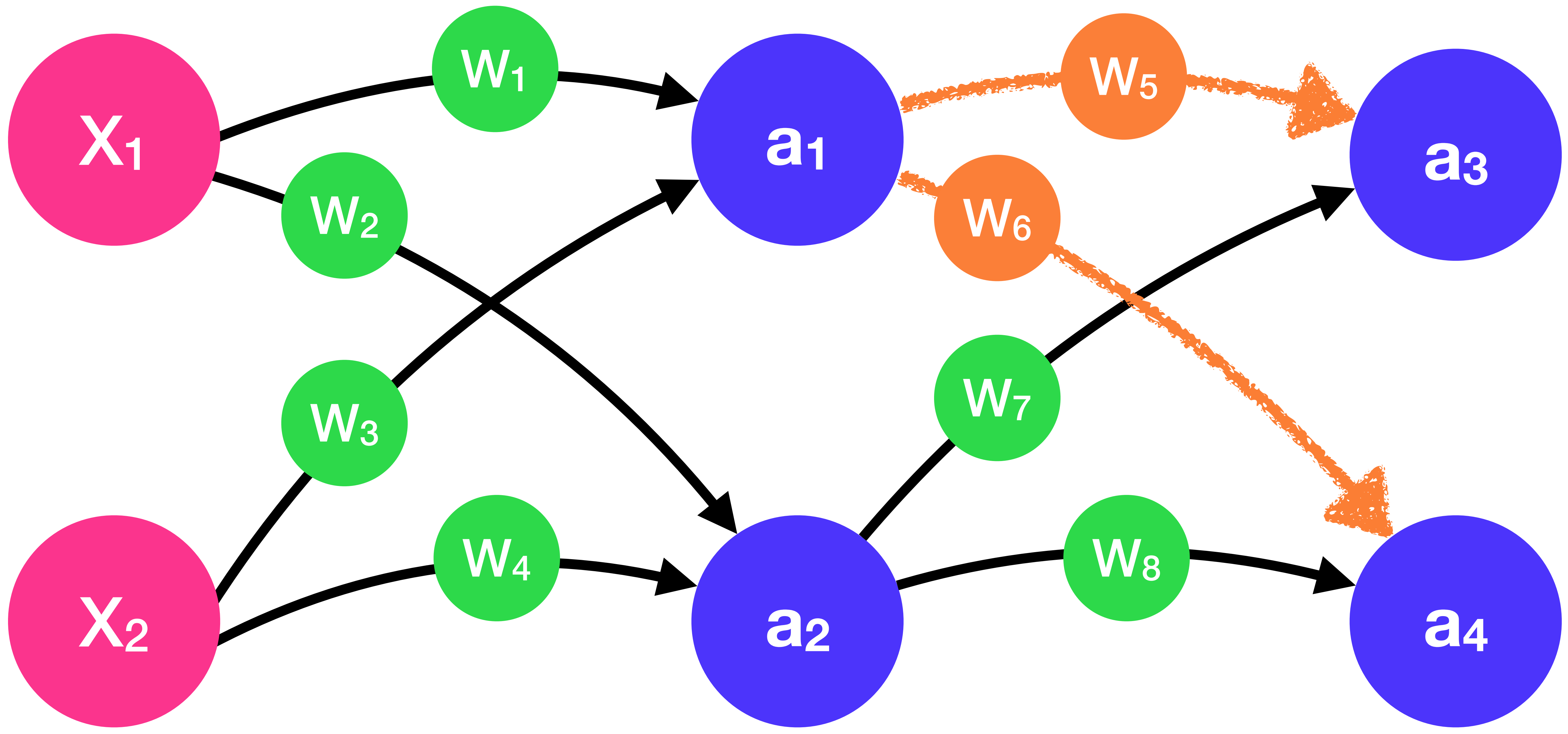
- Lembrando que:

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial a_j} \cdot \frac{\partial a_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_i}$$

Redes Neurais

- Para a camada de saída, os cálculos são os mesmos para a rede perceptron
- Para as camadas intermediárias temos que levar em consideração que a saída de um neurônio pode influenciar a saída de mais de um neurônio na camada posterior

Redes Neurais



Redes Neurais

- Com isso:

$$\frac{\partial J}{\partial a_1} = \frac{\partial J_1}{\partial a_1} + \frac{\partial J_2}{\partial a_1}$$

- Onde J_1 é o erro do 1º neurônio da camada de saída e J_2 é o erro do 2º neurônio da camada de saída

Redes Neurais

$$\frac{\partial J_1}{\partial a_1} = \frac{\partial J_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_1}$$

$$\frac{\partial J_1}{\partial z_3} = \boxed{\frac{\partial J_1}{\partial a_3}} \cdot \boxed{\frac{\partial a_3}{\partial z_3}}$$

$$\frac{\partial J_1}{\partial a_3} = a_3 - a_{esperado}$$

$$\frac{\partial a_3}{\partial z_3} = a_3(1 - a_3)$$

Redes Neurais

$$\frac{\partial J_1}{\partial a_1} = \frac{\partial J_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_1}$$

$$z_3 = b_1 + w_5 a_1 + w_7 a_2$$

$$\frac{\partial z_3}{\partial a_1} = w_5$$

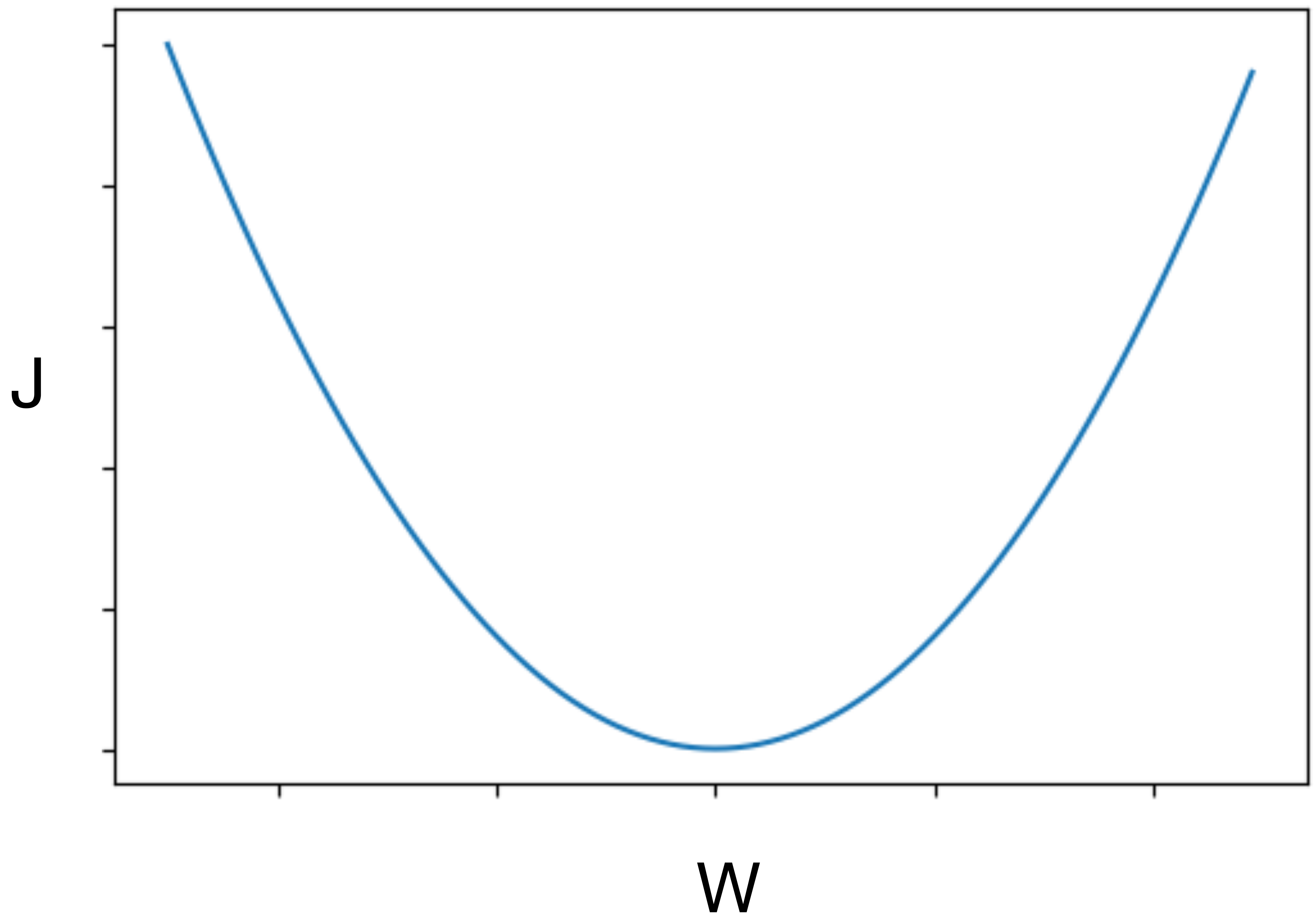
Redes Neurais

- Seguindo o mesmo processo temos que:

$$\frac{\partial J_2}{\partial a_1} = \frac{\partial J_2}{\partial z_4} \cdot \frac{\partial z_4}{\partial a_1}$$

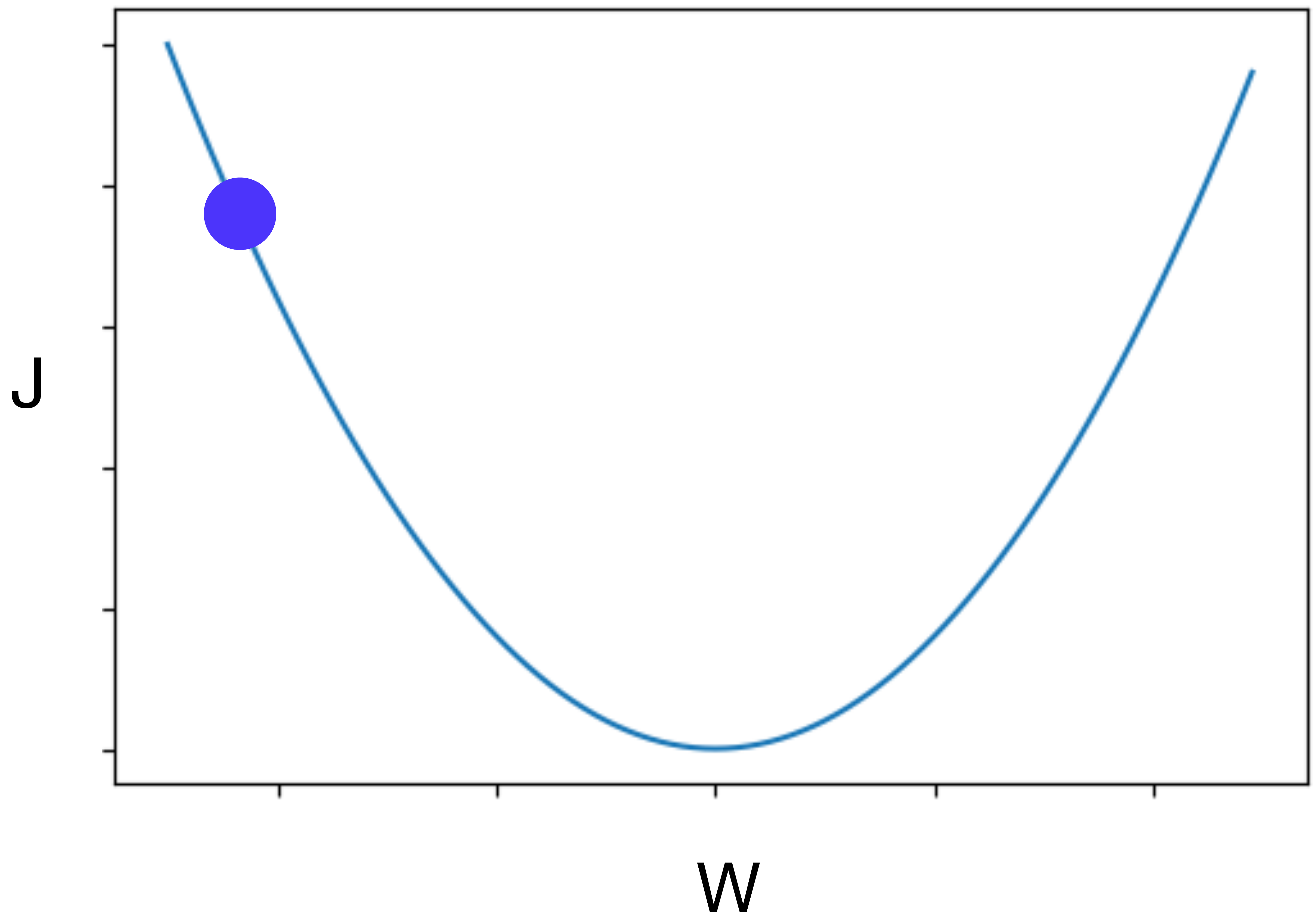
Treinando uma Rede Neural

- A função de erro tem uma forma como essa:



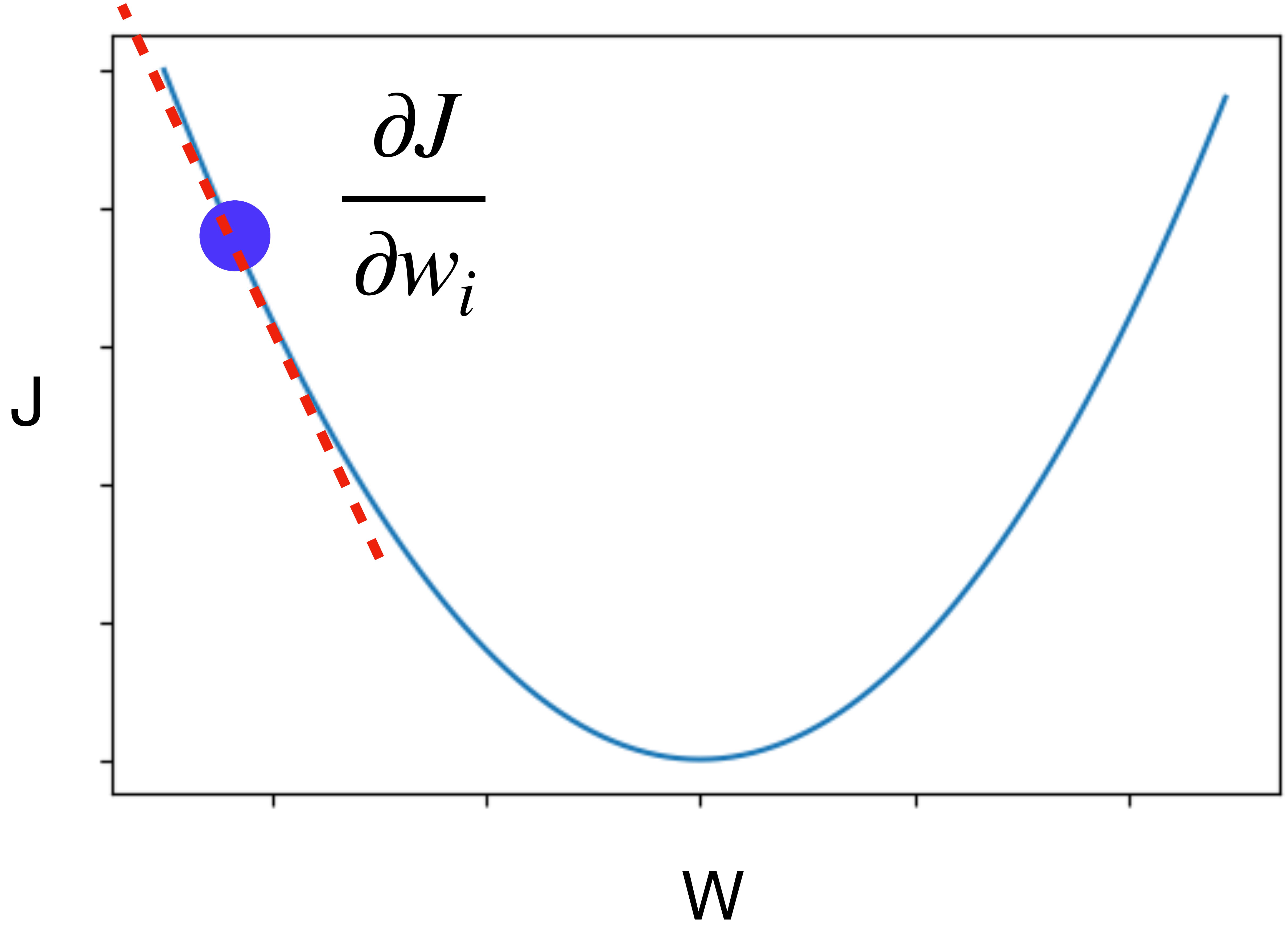
Treinando uma Rede Neural

- Inicializamos o valor de W aleatoriamente



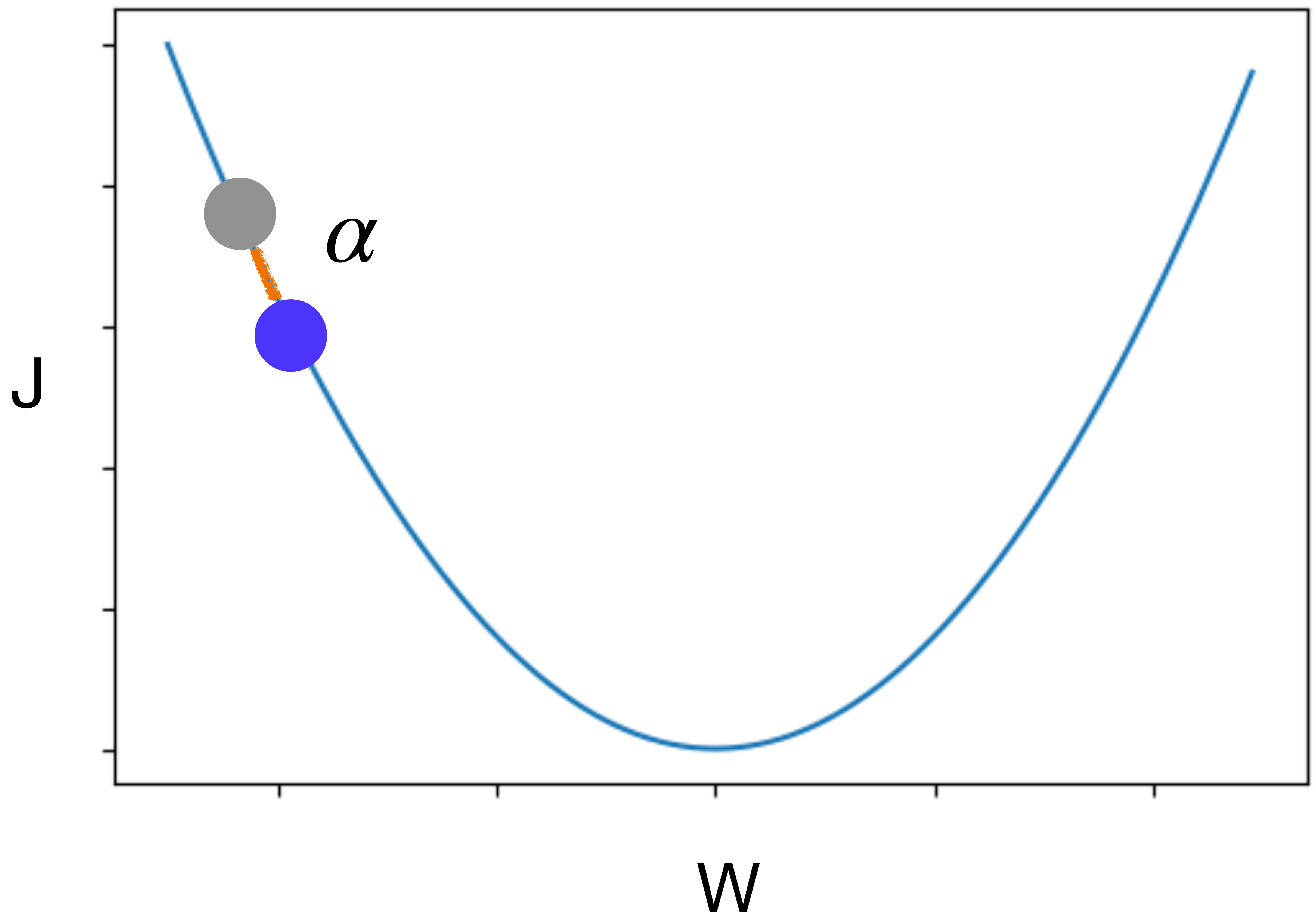
Treinando uma Rede Neural

- A derivada nos fornece a inclinação do ponto



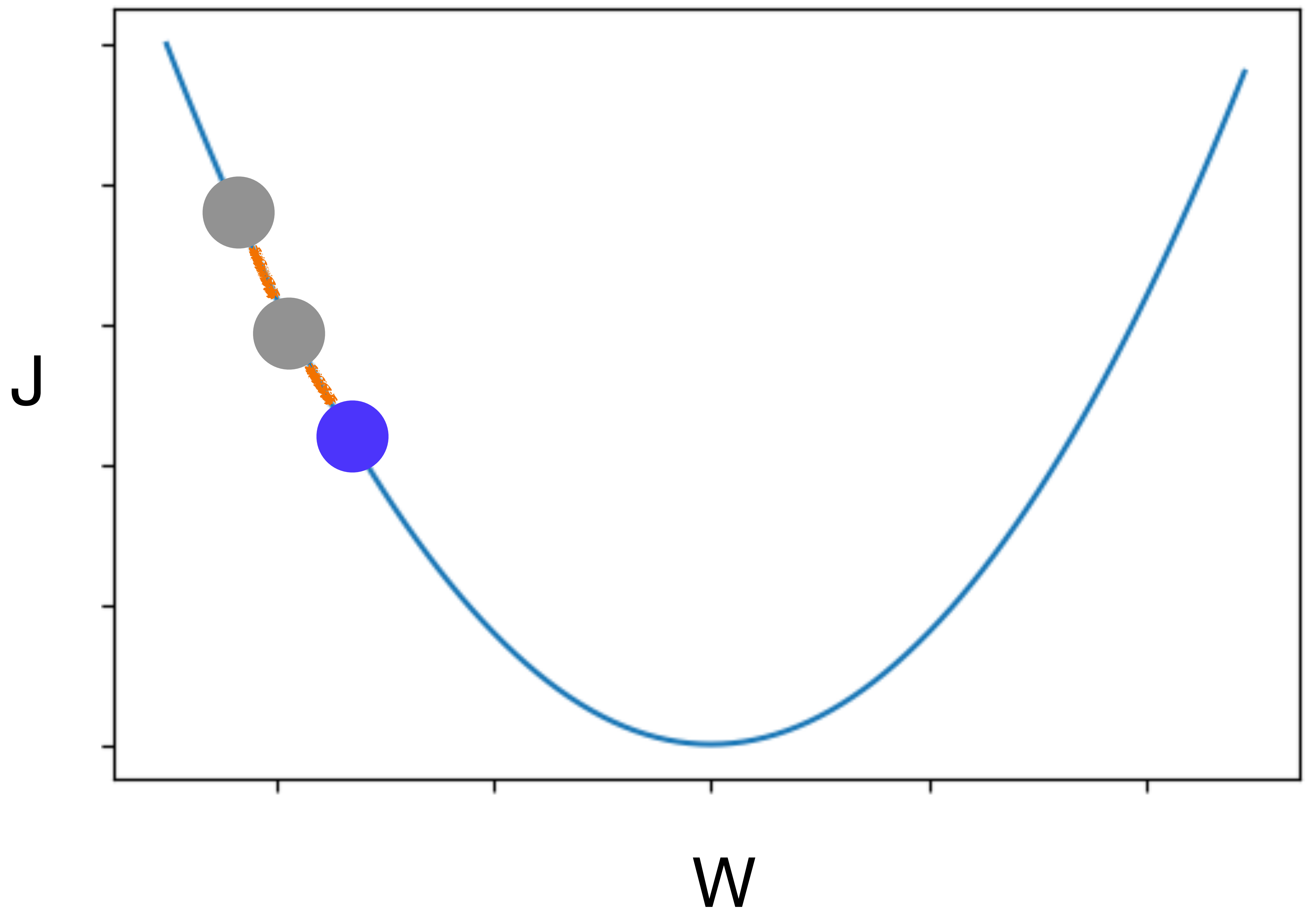
Treinando uma Rede Neural

- A taxa de aprendizagem determina o tamanho do passo de atualização



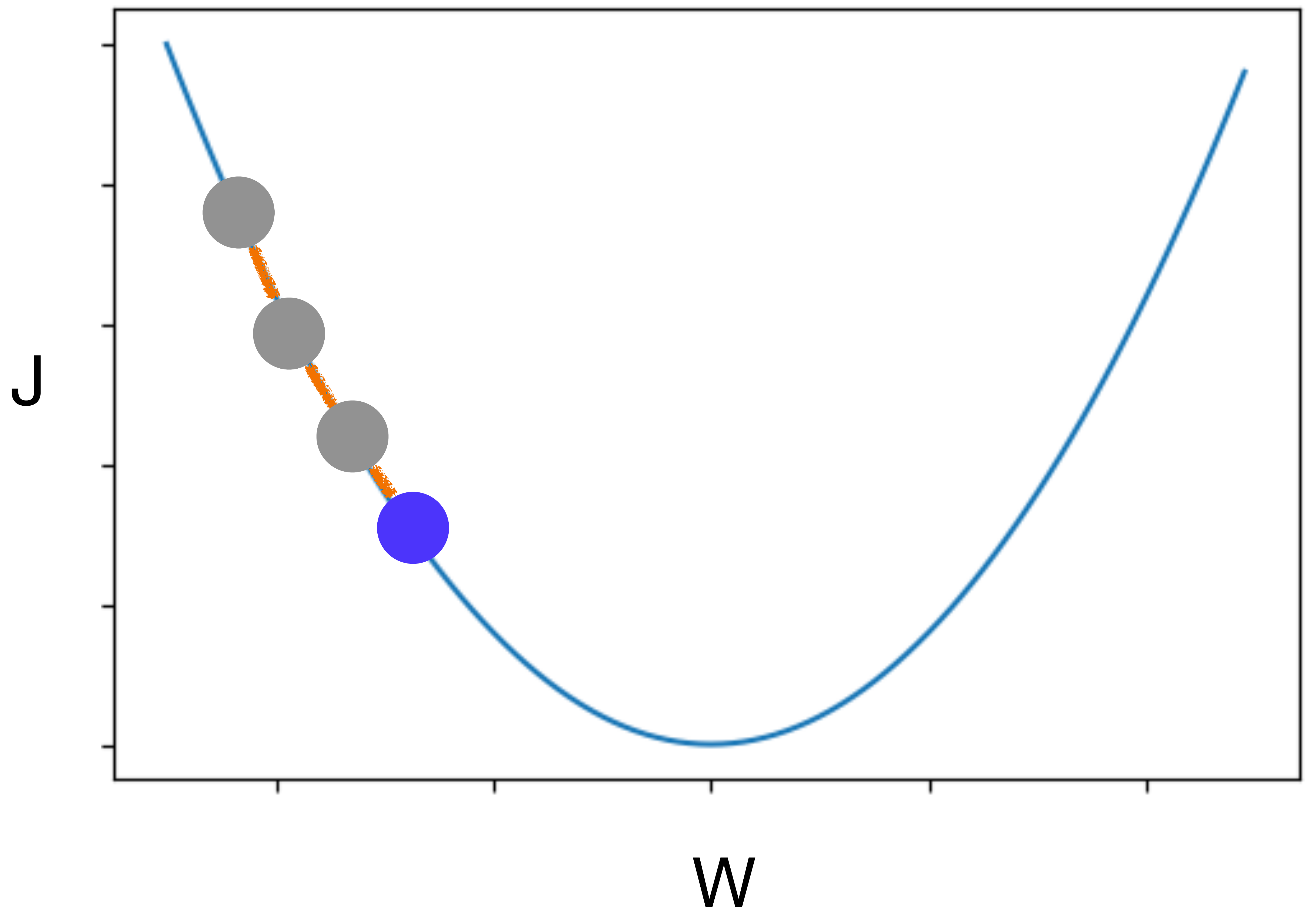
Treinando uma Rede Neural

- Repetimos atualização de **w** múltiplas vezes



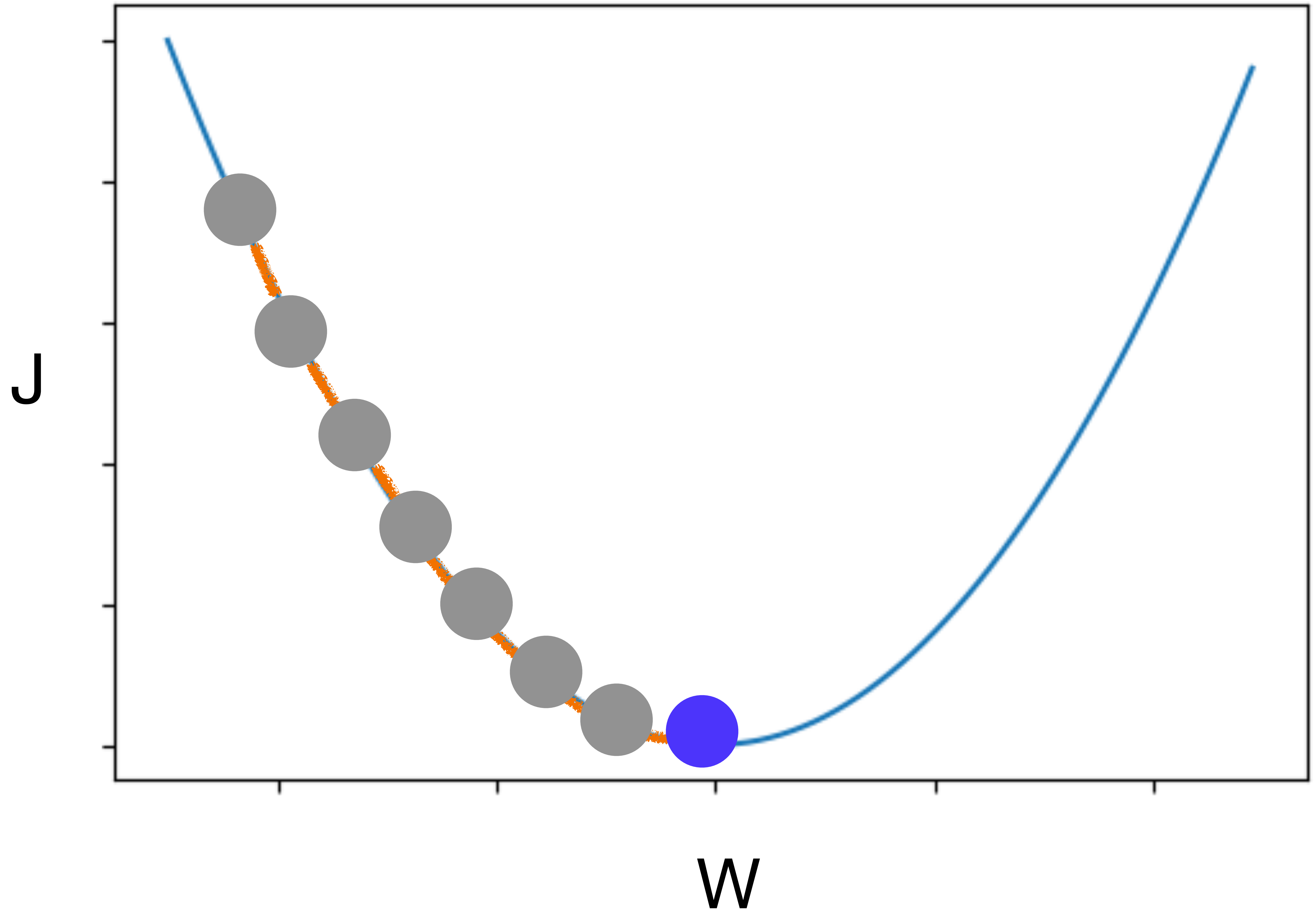
Treinando uma Rede Neural

- Repetimos atualização de **w** múltiplas vezes



Treinando uma Rede Neural

- Repetimos atualização de **w** múltiplas vezes



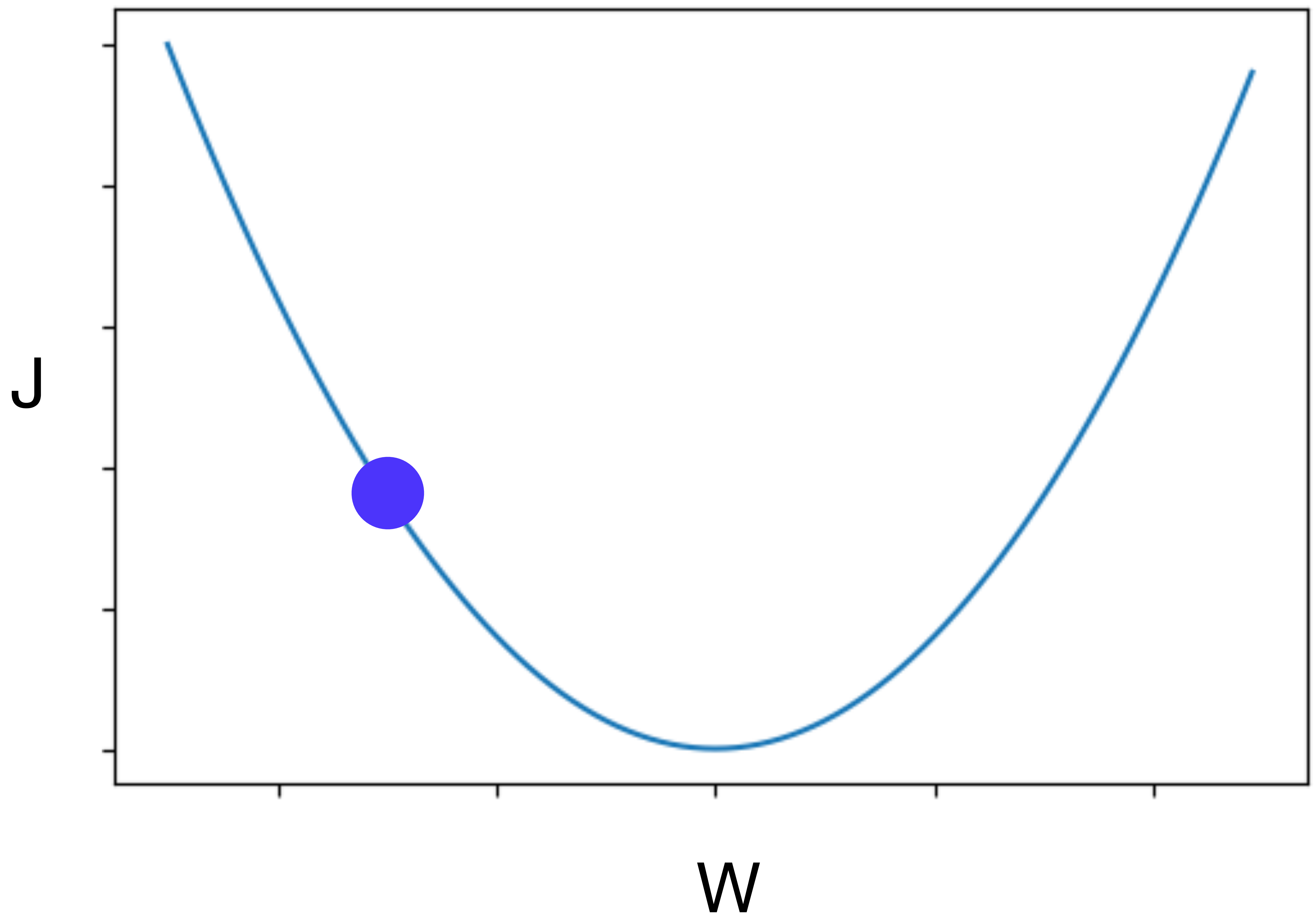
Treinando uma Rede Neural

- Valores pequenos para a taxa de aprendizagem podem fazer o treinamento demorar muito a convergir

- Valores muito grandes podem fazer o treinamento divergir

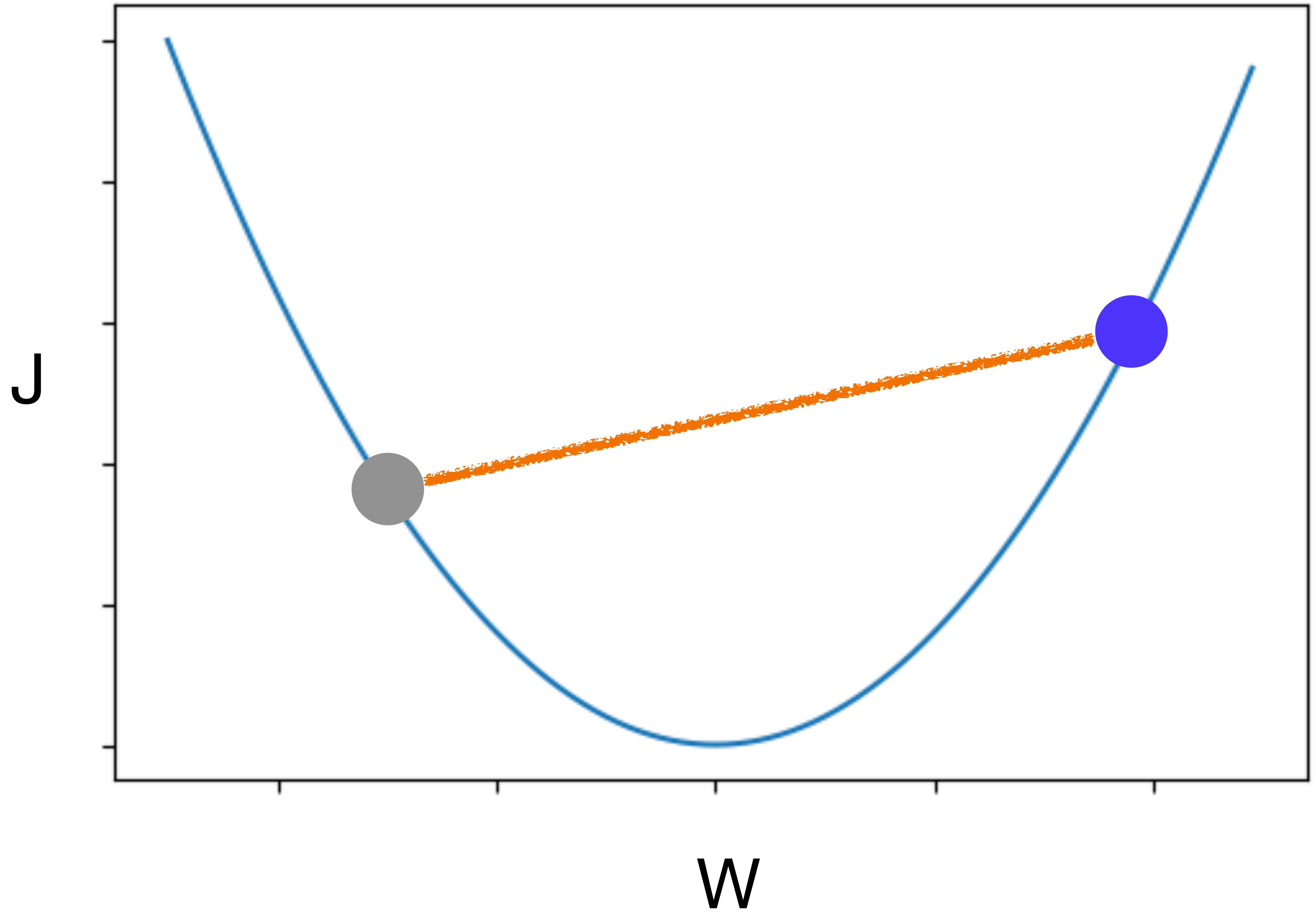
Treinando uma Rede Neural

- Inicializamos o valor de **w** aleatoriamente



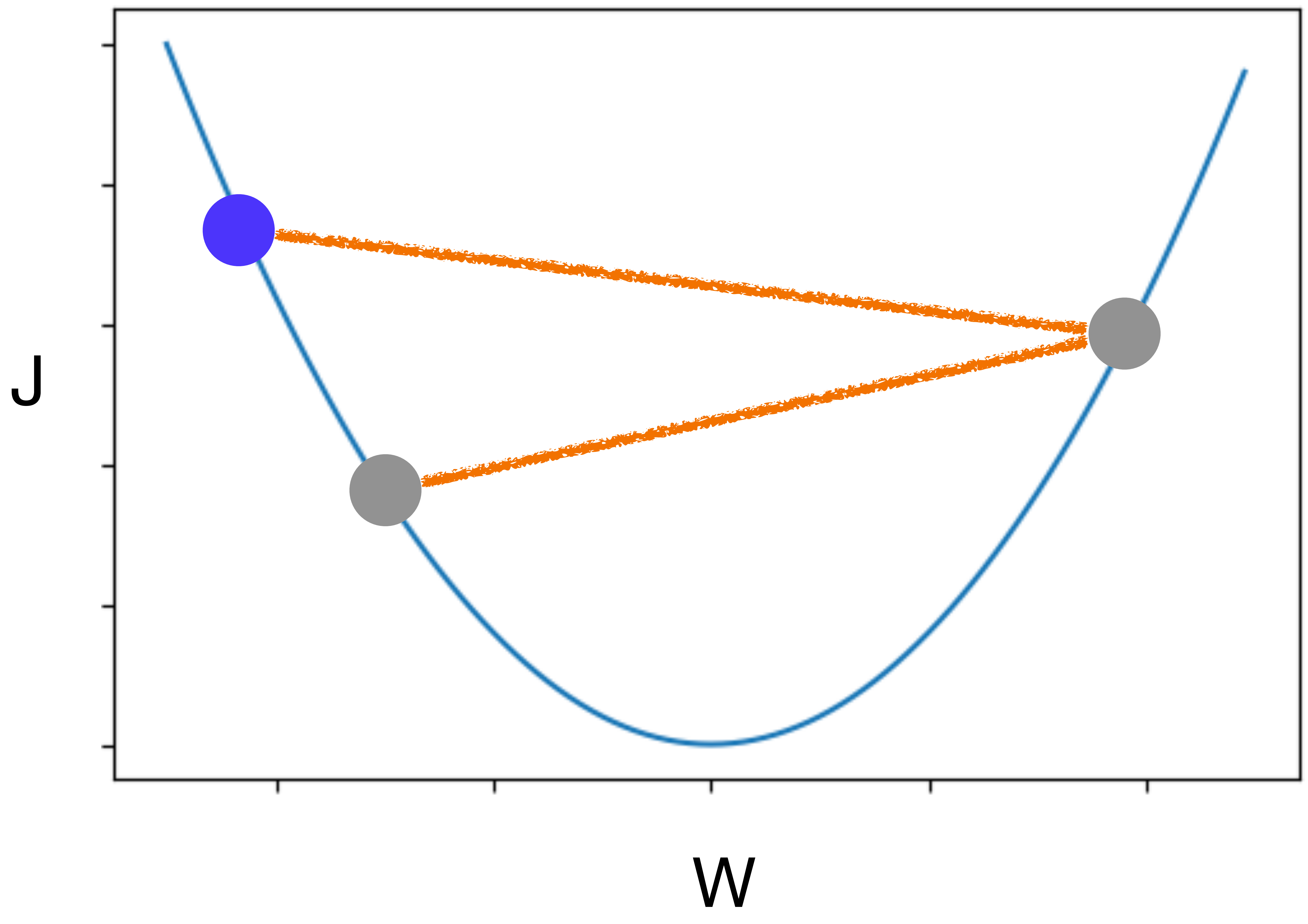
Treinando uma Rede Neural

- O passo de atualização foi tão grande que o erro aumentou



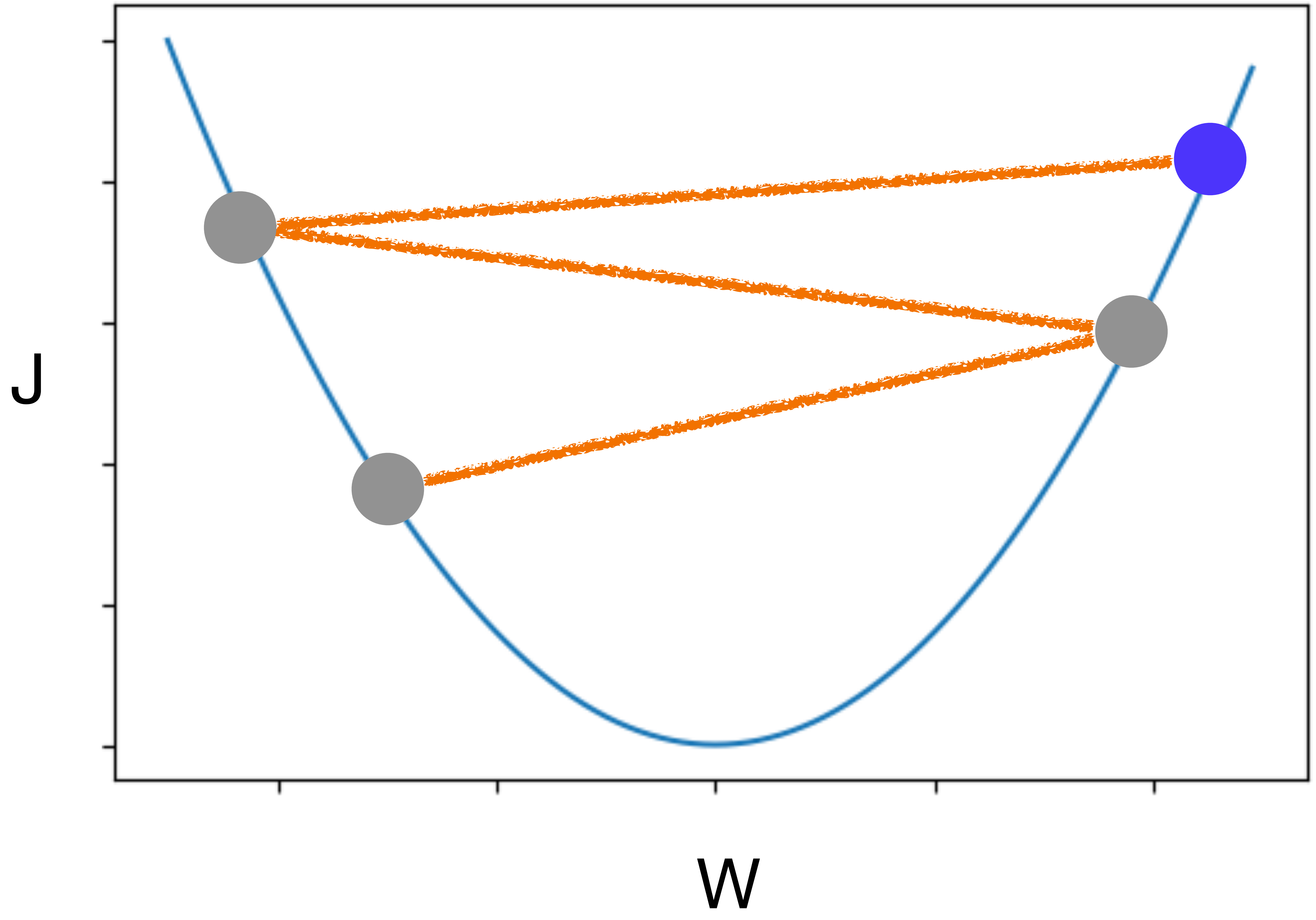
Treinando uma Rede Neural

- O erro continua aumentando

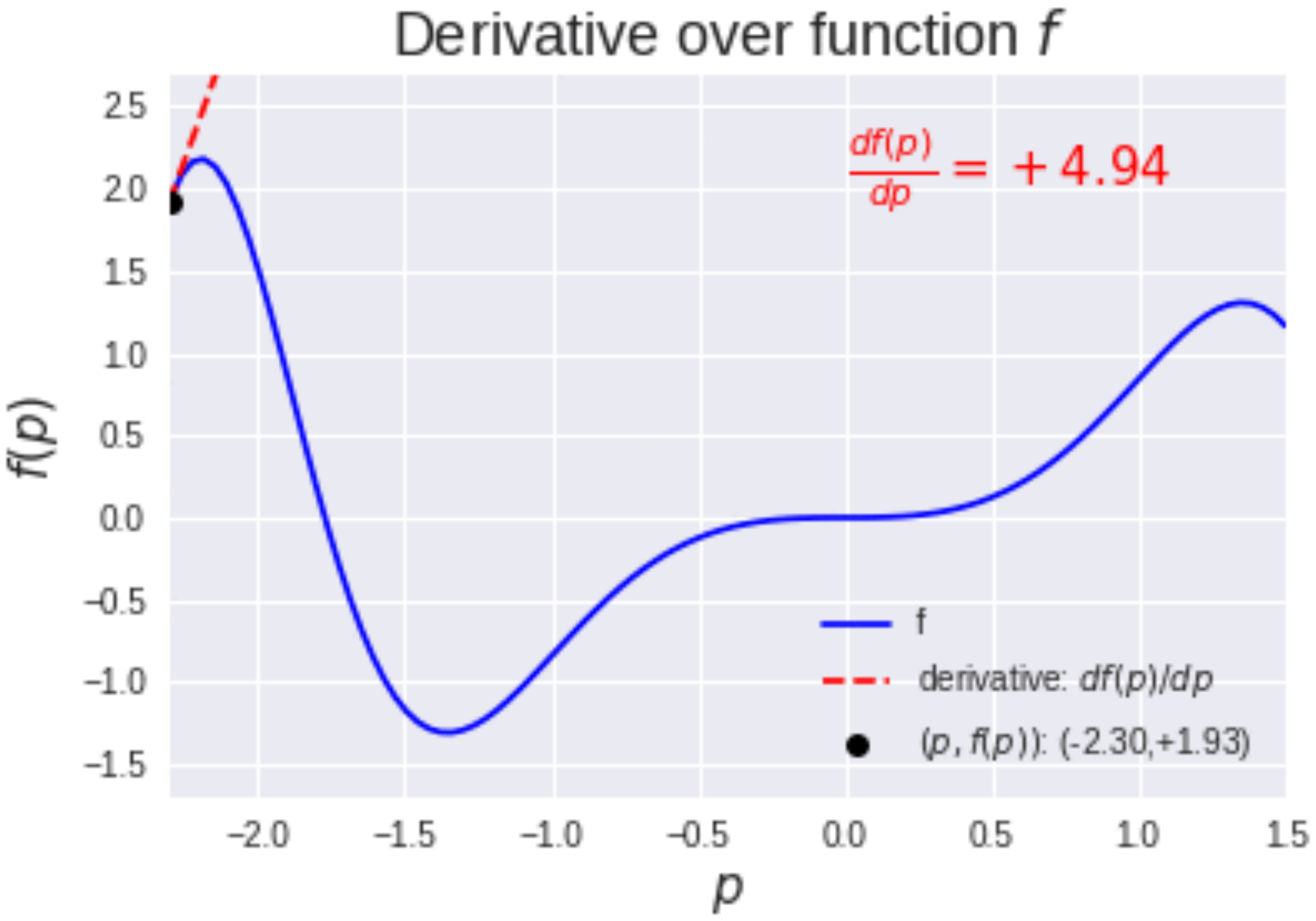


Treinando uma Rede Neural

- O erro continua aumentando



Descida de Gradiente



Exemplo prático

- Jupyter Notebook…