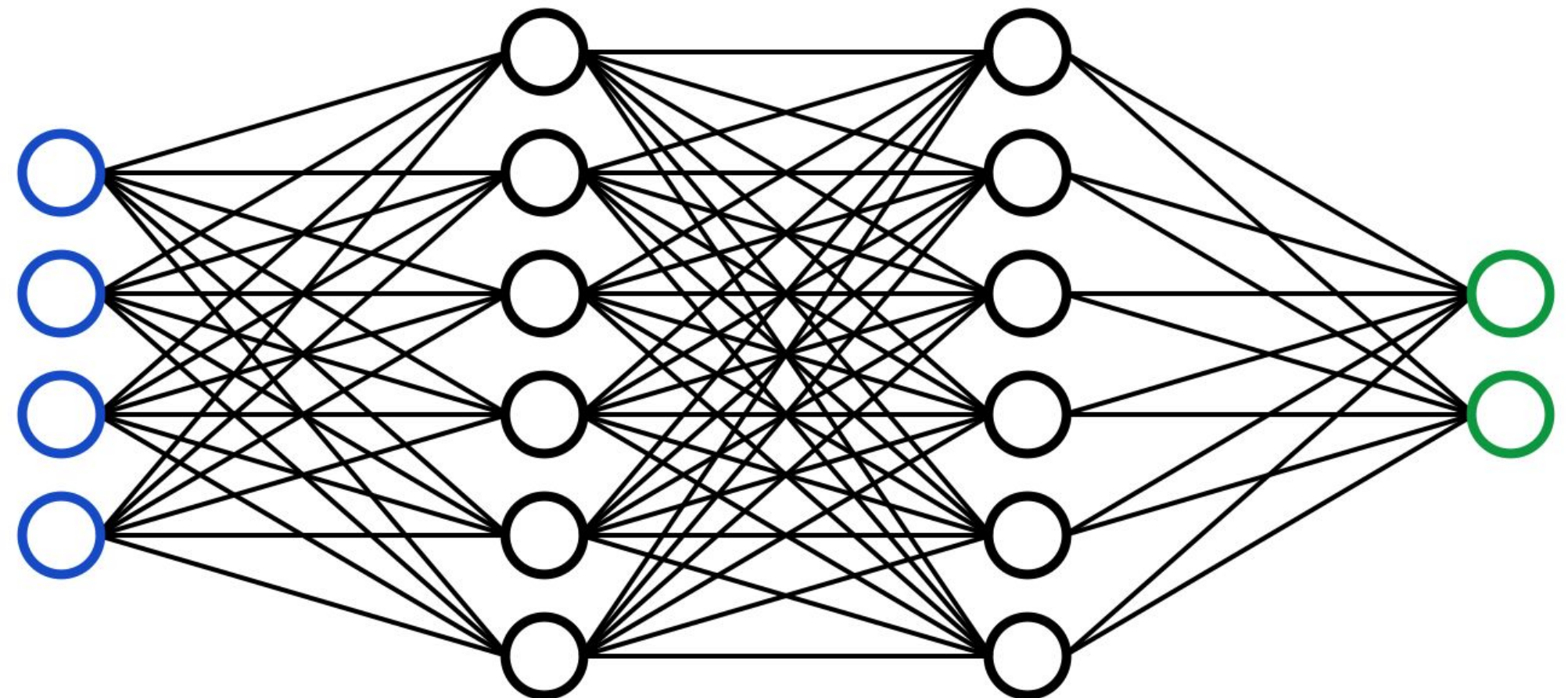


Período 3

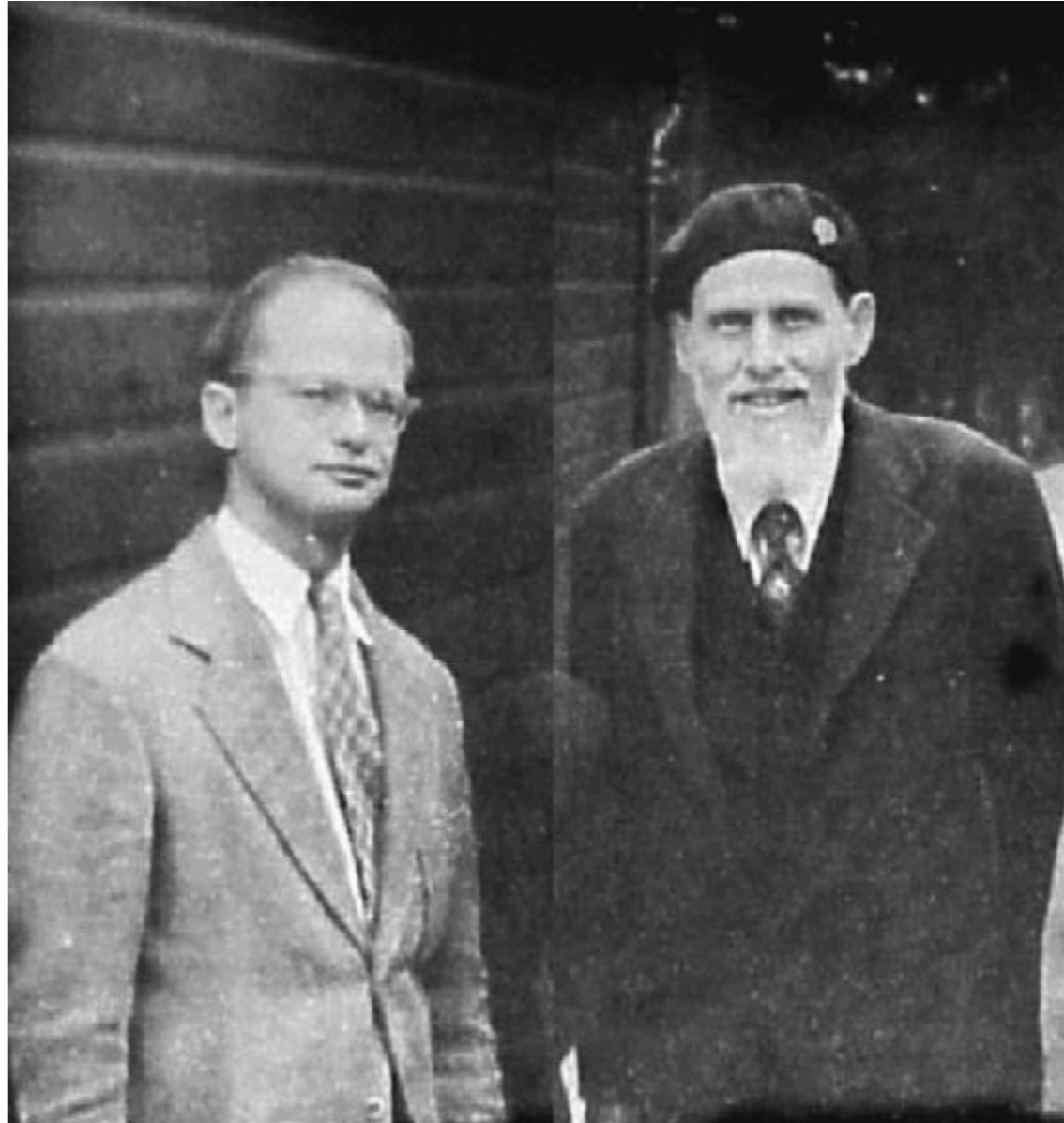
Redes Neurais

O que são Redes Neurais?

- Inspiradas (de forma simplificada) no cérebro humano.
- Sistemas que aprendem padrões a partir de dados.
- Usadas para: reconhecimento de imagem, tradução, previsão, etc.
- Não "pensam", mas são excelentes em encontrar relações complexas.



1943 - Primeira Idealização



A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH and WALTER H. PITTS

Because of the “all-or-none” character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

Perceptron de Rosenblatt

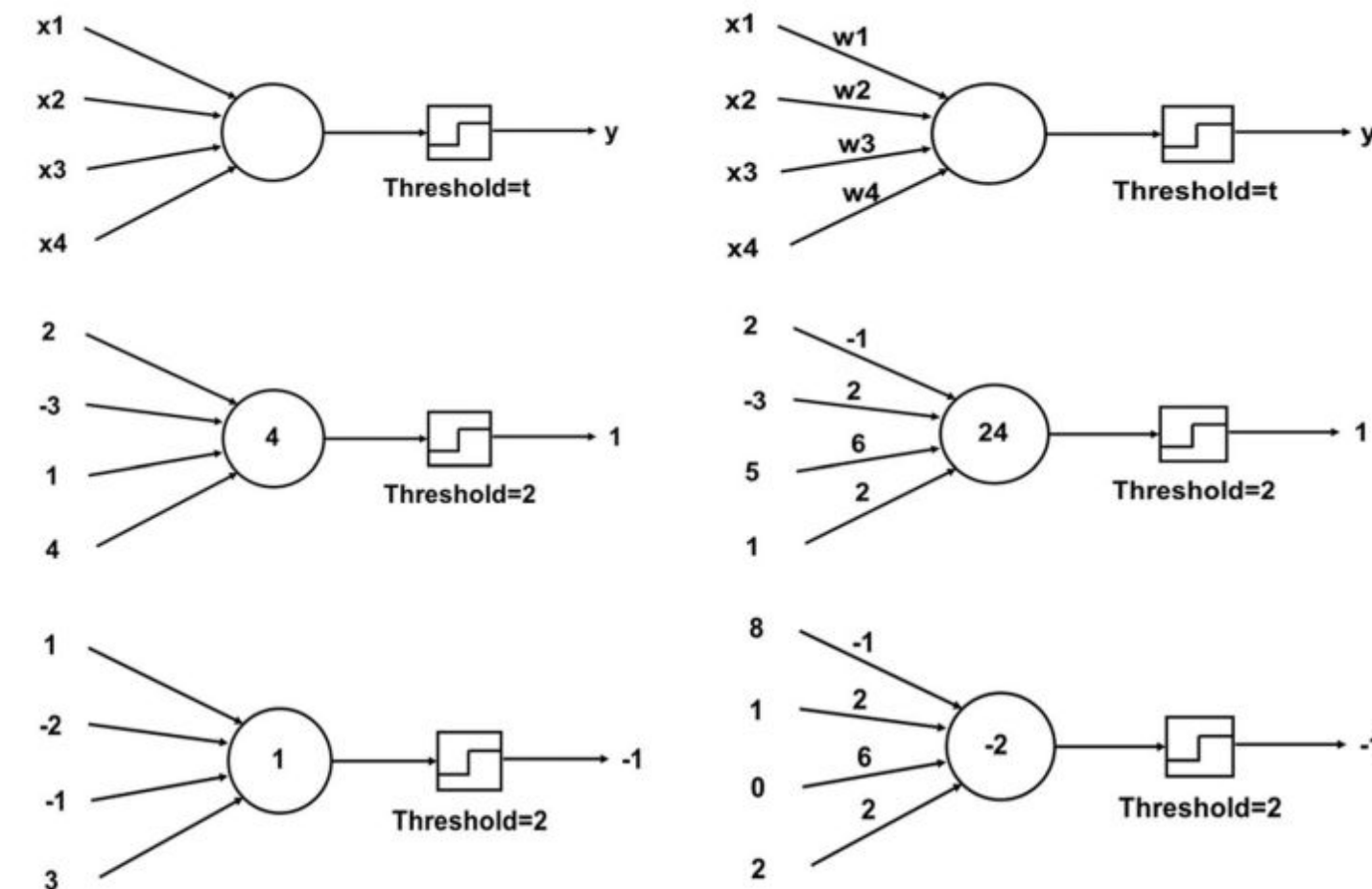
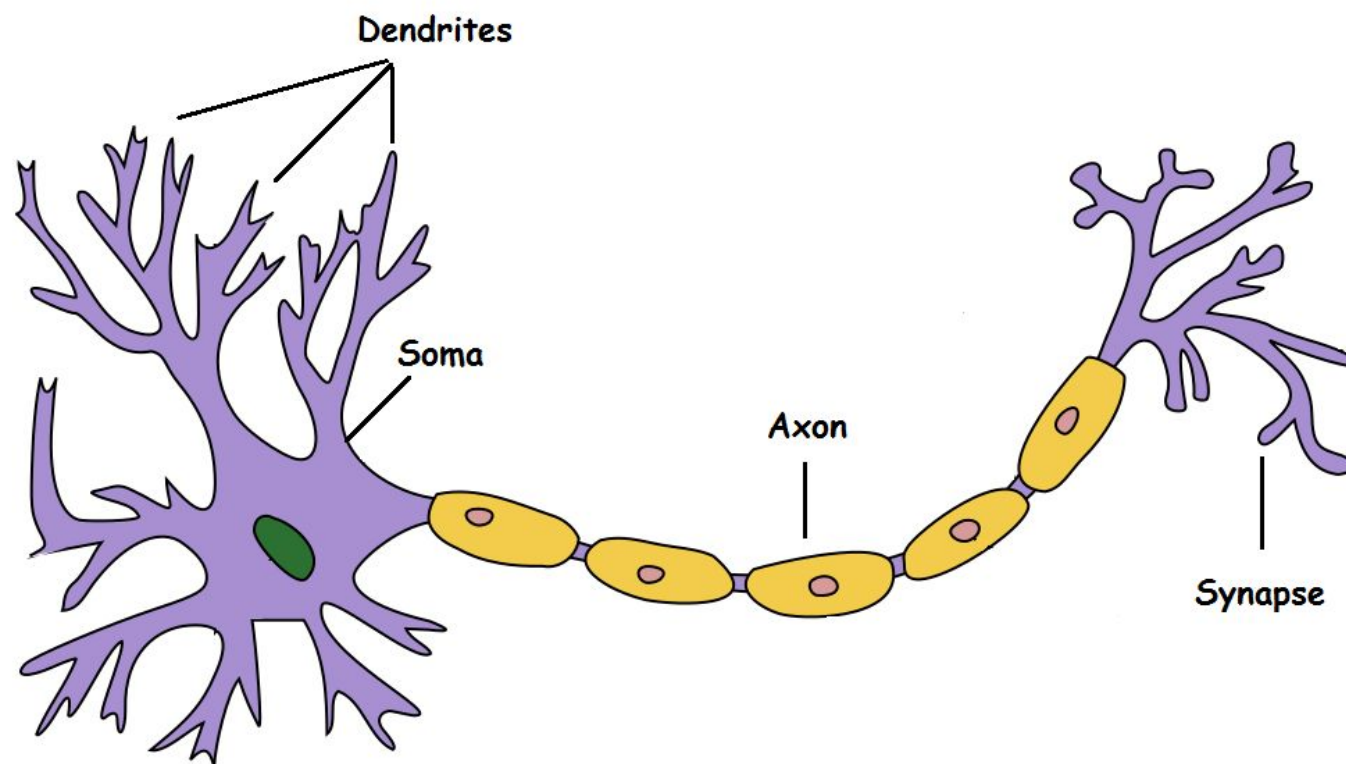
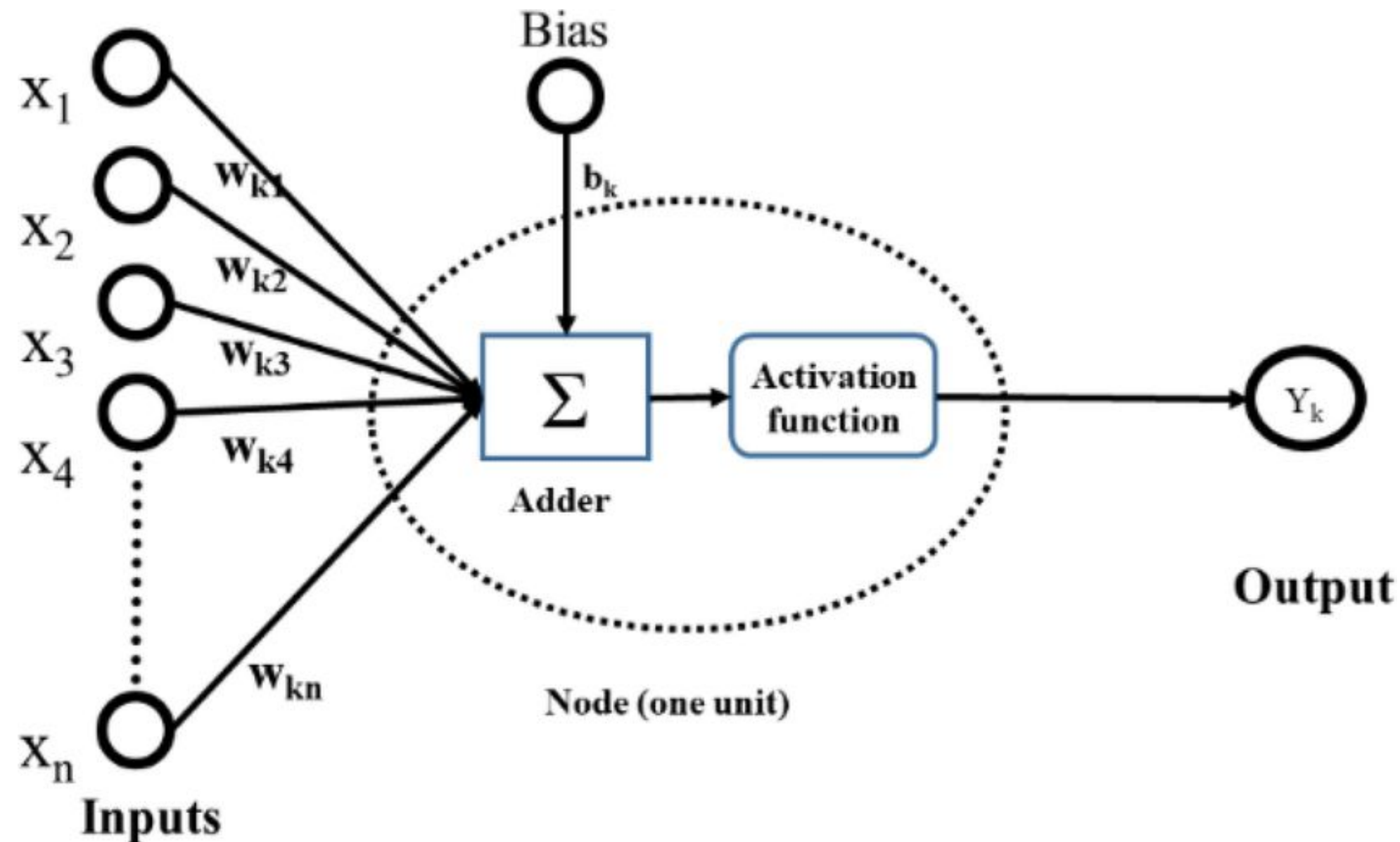
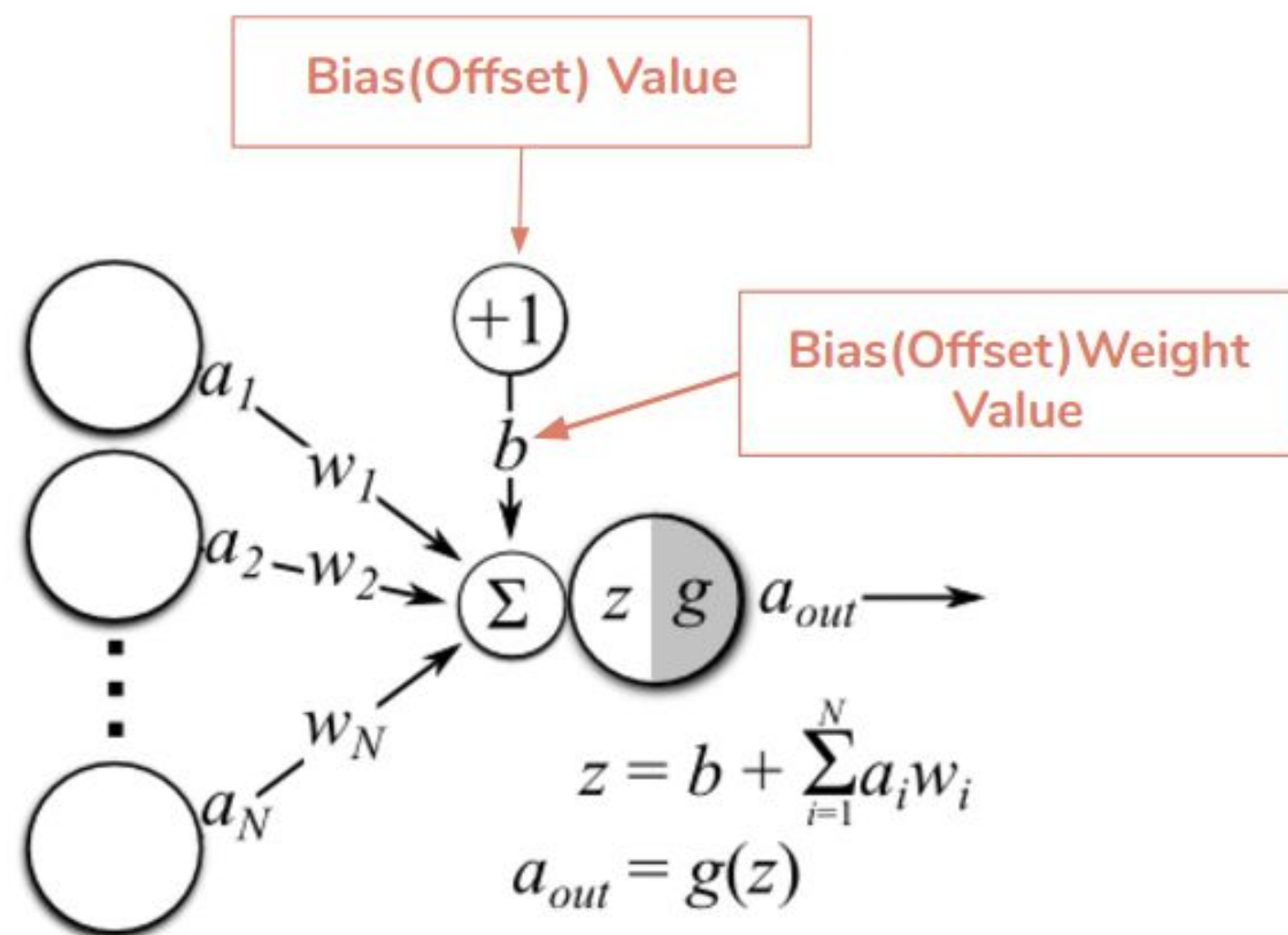


Figure 1. Graphical representation of the main concepts of a McCulloch–Pitts neuron (left column) and a Rosenblatt perceptron (right column), with examples. A McCulloch–Pitts neuron sums up all its inputs (represented by x_1, x_2, x_3, x_4 , etc.) and the result is compared with a predetermined threshold: if the result is higher than a predefined threshold, the neuron fires (outcome +1); if the result is lower than a predefined threshold, the neuron does not fire (outcome -1). Rosenblatt built on this concept to create the perceptron, a neuron where the inputs are multiplied by weights (represented by w_1, w_2, w_3, w_4 , etc.) before being summed up. The crucial insight in the perceptron is that weights can be adjusted (learned) to fit the desired neuronal behavior.

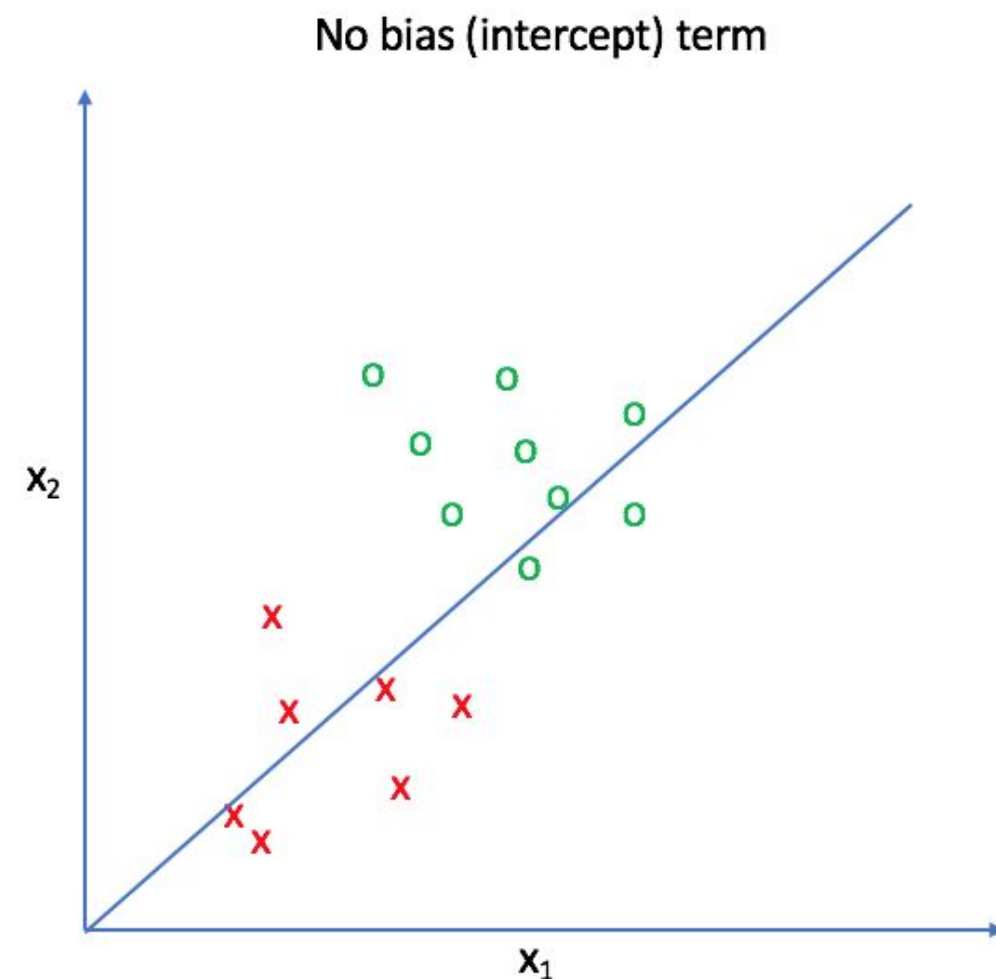
Componentes do Perceptron



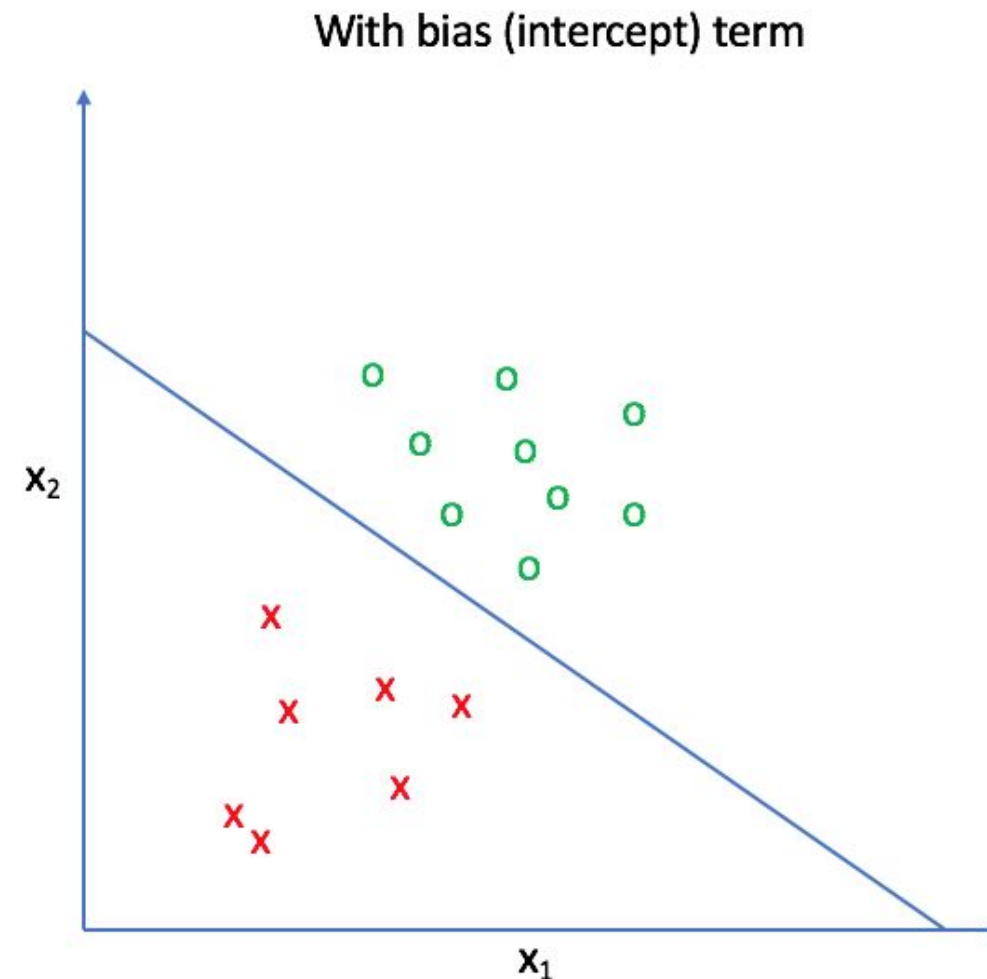
Componentes do Perceptron - Pesos e Bias



Componentes do Perceptron - Pesos e Bias

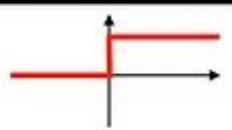
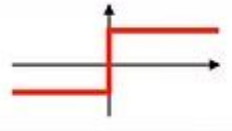
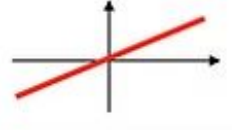


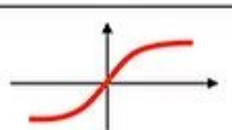

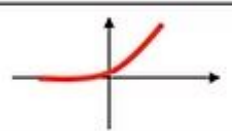


Our line is forced to pass through the origin.



Adding the intercept term allows for much better fit.

Componentes do Perceptron - Funções de Ativação

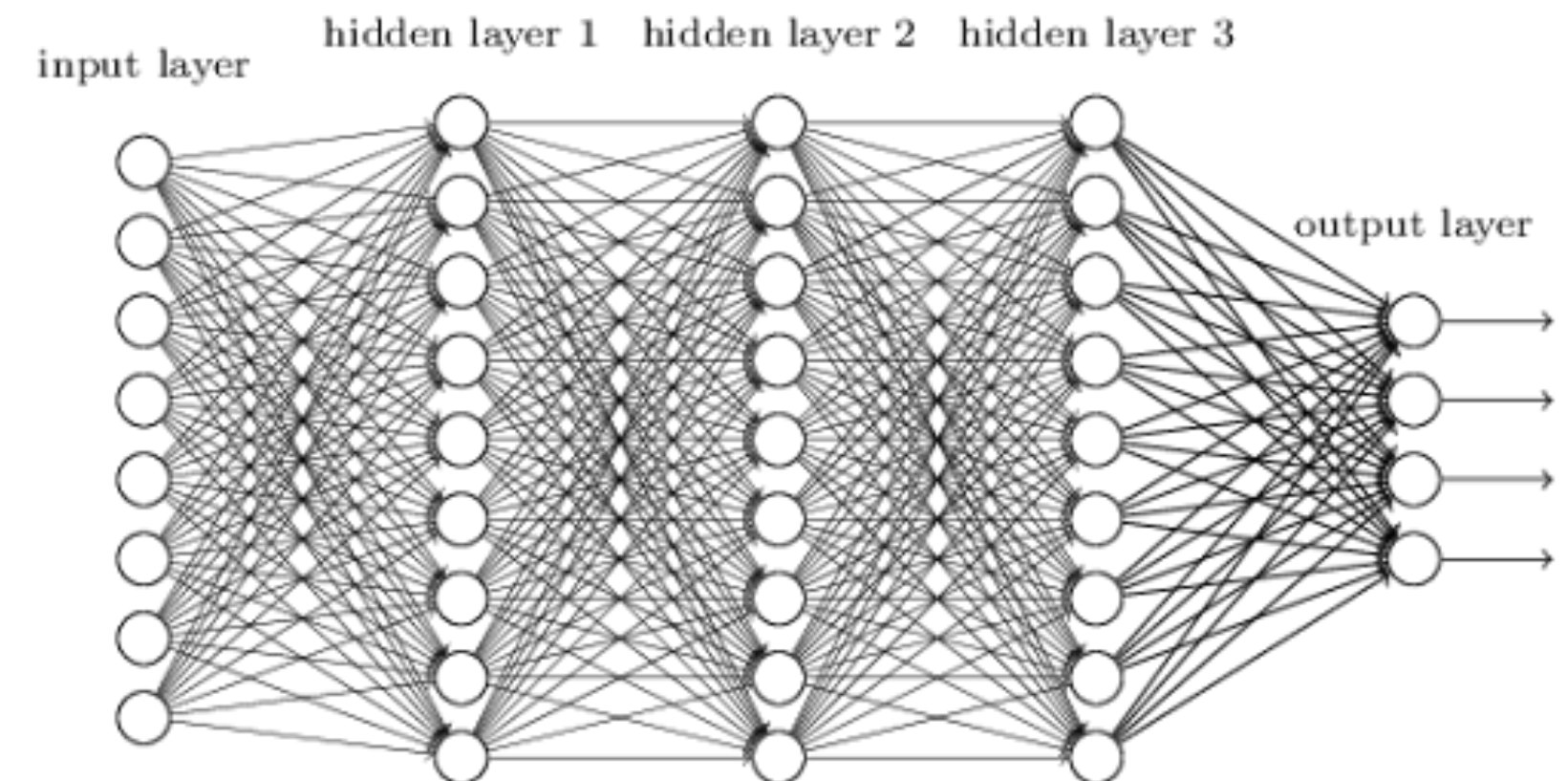
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016
(<http://sebastianraschka.com>)

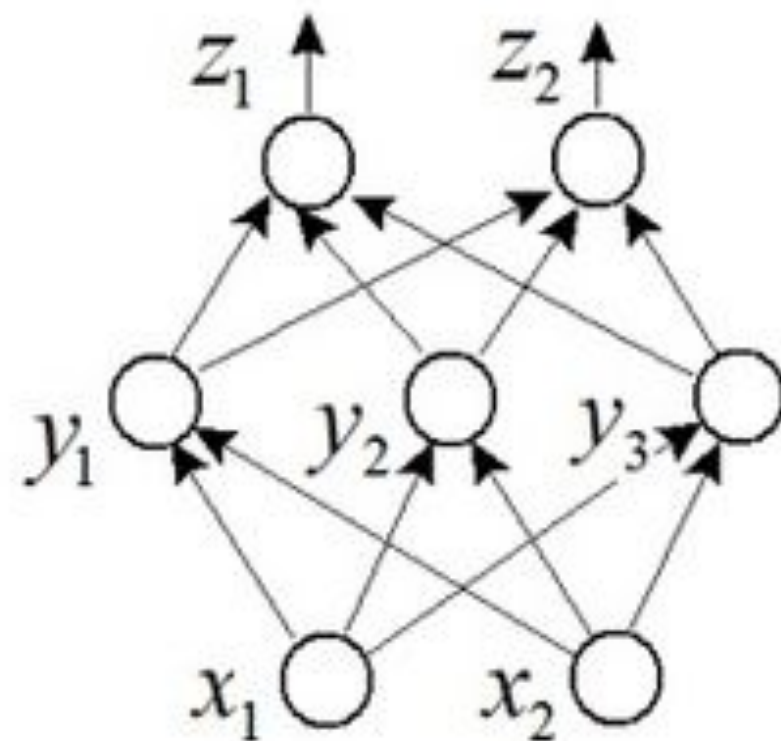
Servem para introduzir
não-linearidades

Construindo Redes Maiores: Largura vs. Profundidade

- Mais Neurônios (Largura):
 - Aprende mais características/padrões no mesmo nível de abstração.
 - Aumenta a capacidade de representação de uma camada específica.
- Mais Camadas (Profundidade):
 - Aprende uma hierarquia de características (simples para complexas).
 - Permite combinações mais complexas e não lineares dos dados.



Operações Vetoriais e Feed Forward



$$(1) \begin{bmatrix} \bar{y}_1 \\ \bar{y}_2 \\ \bar{y}_3 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & b_1 \\ w_{21} & w_{22} & b_2 \\ w_{31} & w_{32} & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ -1 \end{bmatrix}$$

$$(2) \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} f(\bar{y}_1) \\ f(\bar{y}_2) \\ f(\bar{y}_3) \end{bmatrix}$$

$$(3) \begin{bmatrix} \bar{z}_1 \\ \bar{z}_2 \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & v_{13} & a_1 \\ v_{21} & v_{22} & v_{23} & a_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ -1 \end{bmatrix}$$

$$(4) \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} f(\bar{z}_1) \\ f(\bar{z}_2) \end{bmatrix}$$

Medindo o Erro: Função de Custo

- Como sabemos se a rede está acertando ou errando?
- A Função de Custo (ou Loss Function) quantifica a diferença entre a previsão da rede e o valor real.
- Objetivo do treinamento: Minimizar essa função!

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|$$

$$Huber Loss = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

$$|y^{(i)} - \hat{y}^{(i)}| \leq \delta$$

$$\frac{1}{n} \sum_{i=1}^n \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta)$$

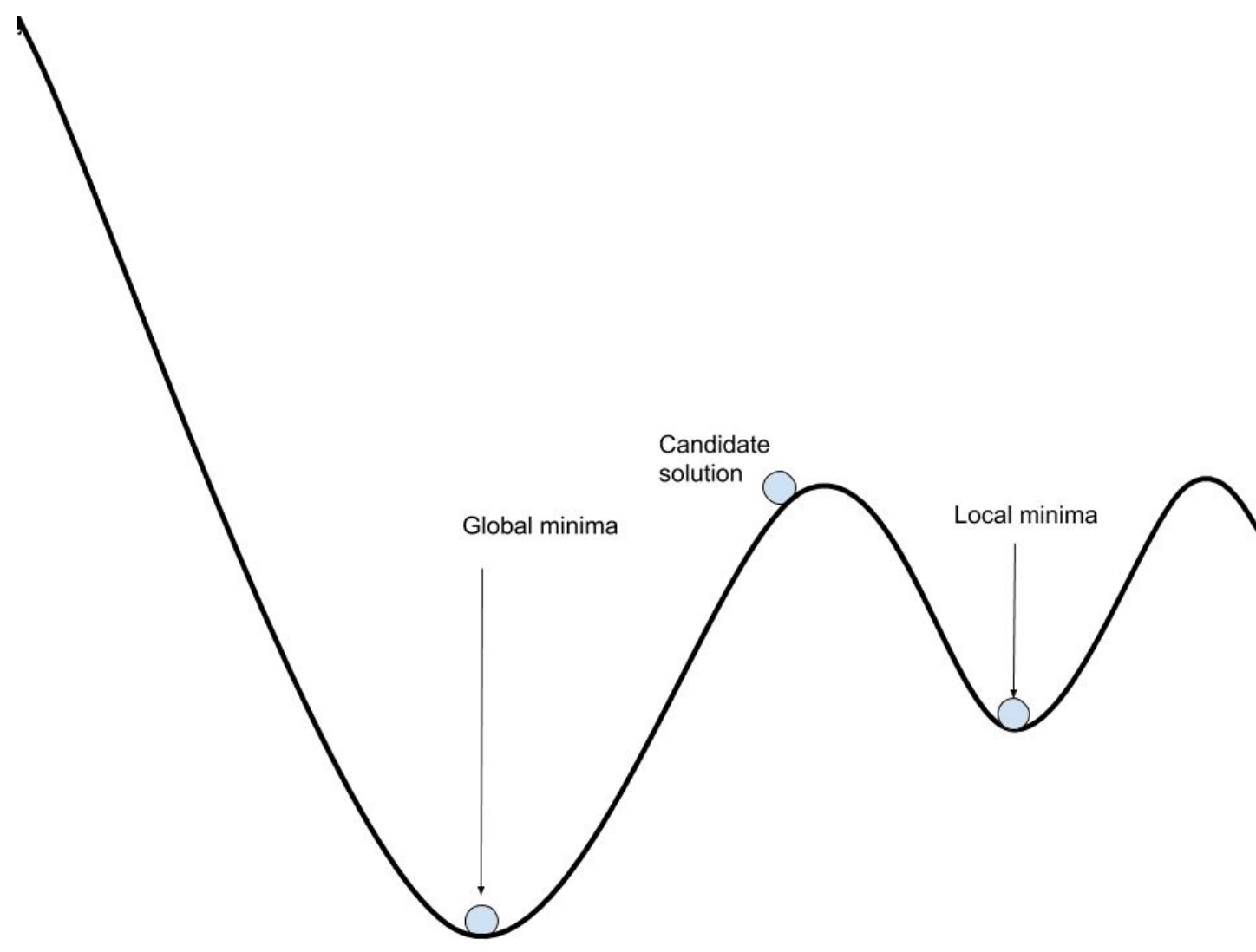
$$|y^{(i)} - \hat{y}^{(i)}| > \delta$$

$$CE Loss = \frac{1}{n} \sum_{i=1}^N - (y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i))$$

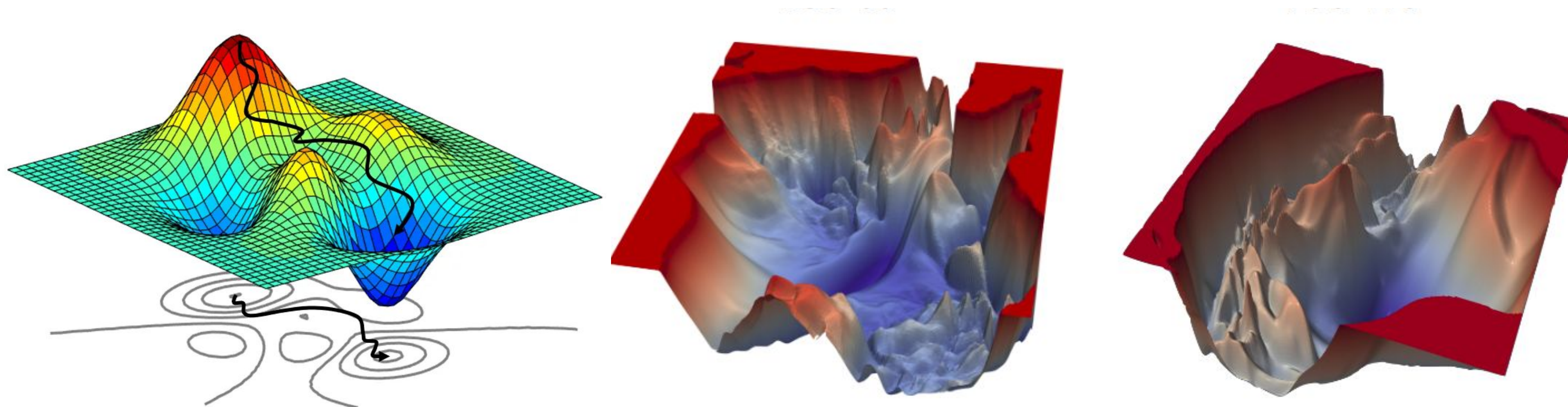
$$CE Loss = -\frac{1}{n} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \cdot \log(p_{ij})$$

<https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9/>

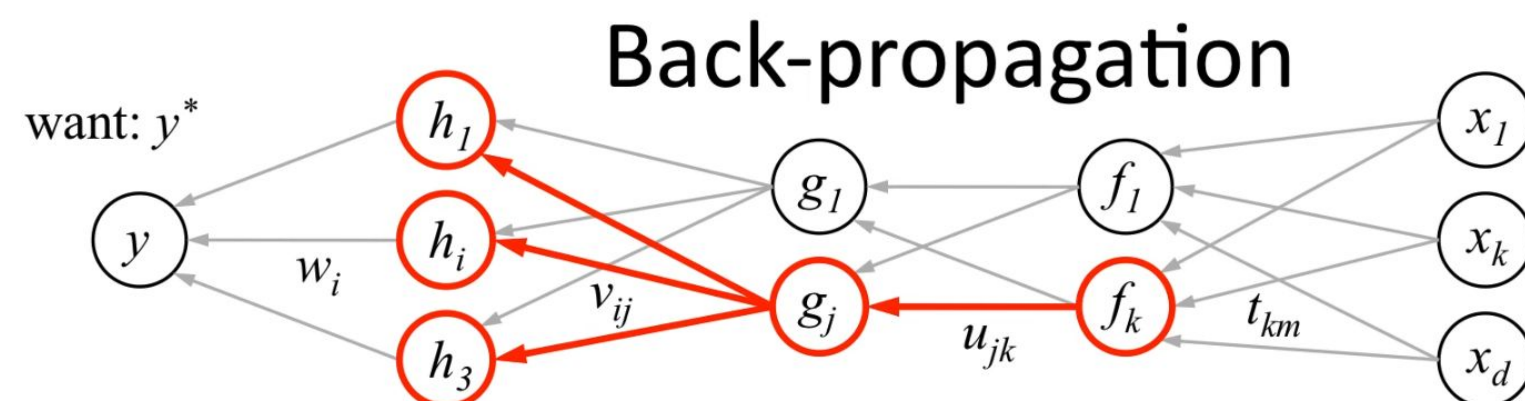
Mínimos Locais e Globais



E para o caso da Função de Custo? Gradiente Descentente



Backpropagation



1. receive new observation $\mathbf{x} = [x_1 \dots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer $1 \dots L$
compute g_j based on units f_k from previous layer: $g_j = \sigma \left(u_{j0} + \sum_k u_{jk} f_k \right)$
3. get prediction y and error $(y - y^*)$
4. **back-propagate error:** for each unit g_j in each layer $L \dots 1$

(a) compute error on g_j

$$\frac{\partial E}{\partial g_j} = \sum_i \underbrace{\sigma'(h_i)}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{v_{ij}}_{\text{was } h_i \text{ too high or too low?}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{should } g_j \text{ be higher or lower?}}$$

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \underbrace{\frac{\partial E}{\partial g_j}}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{\sigma'(g_j) f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

(ii) update the weight

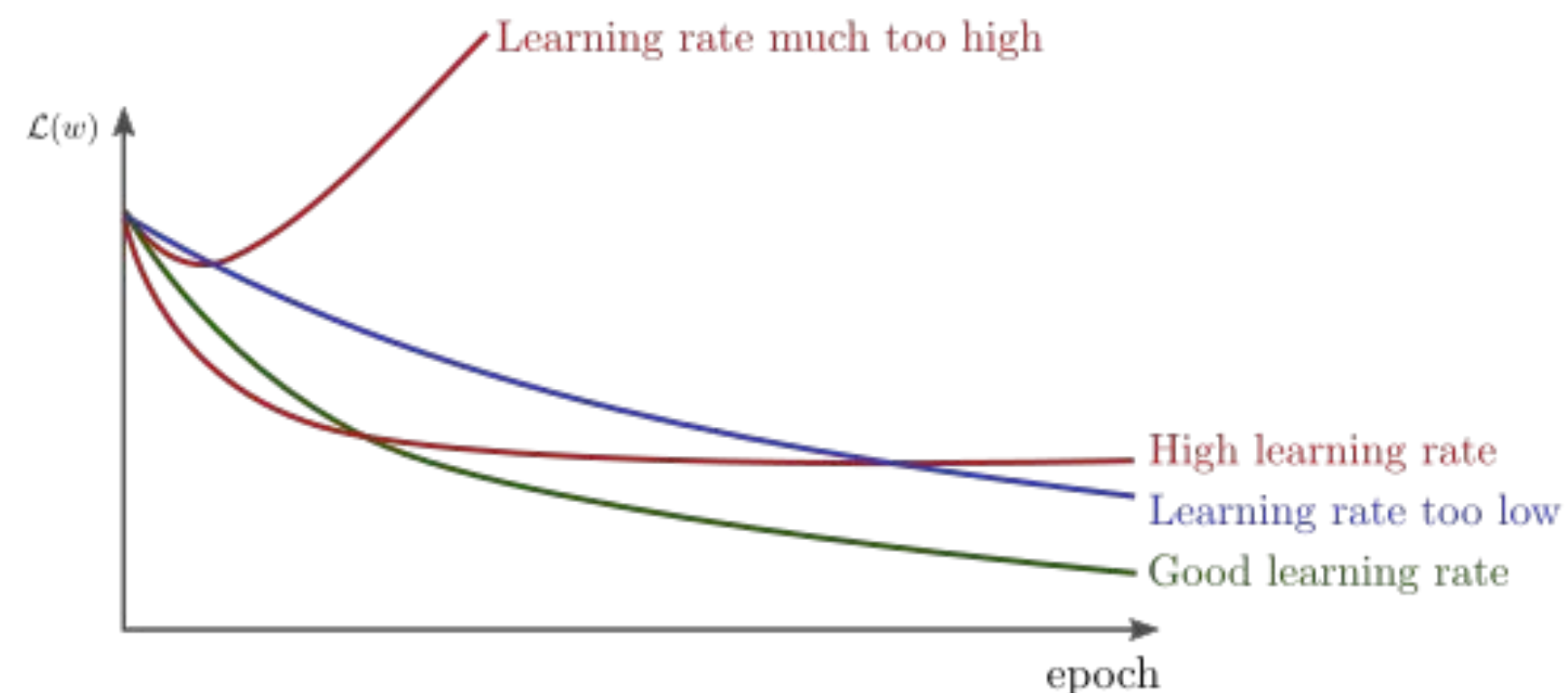
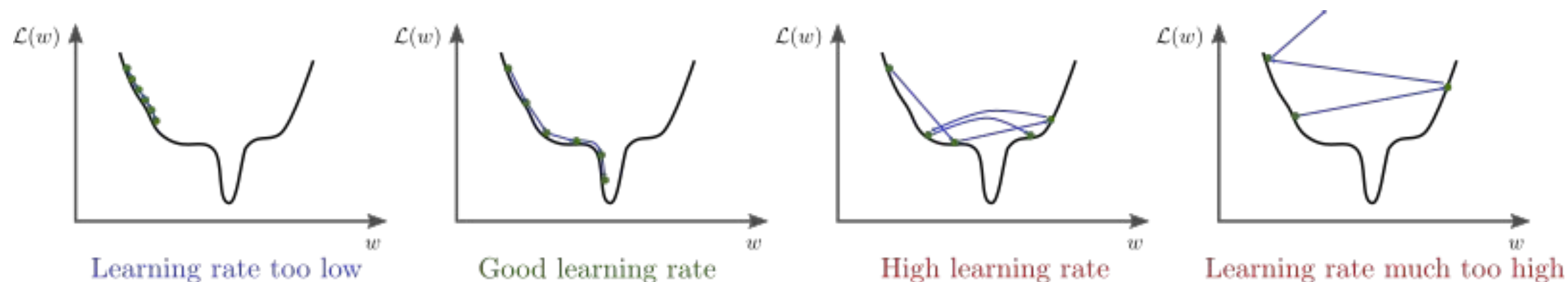
$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

Copyright © 2014 Victor Lavrenko

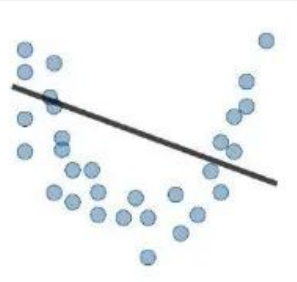

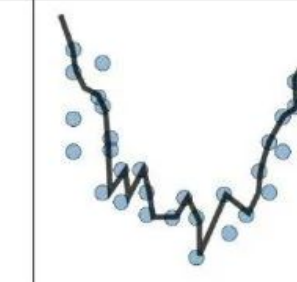
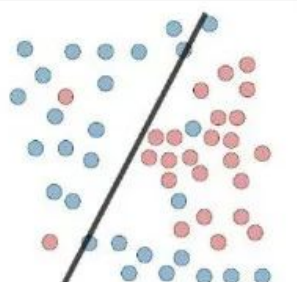
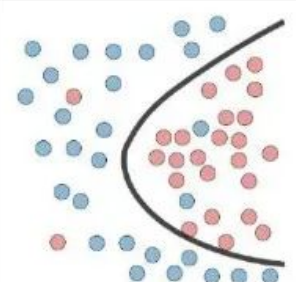
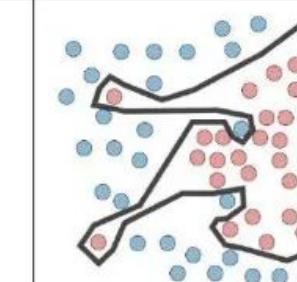

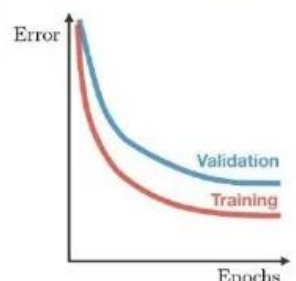
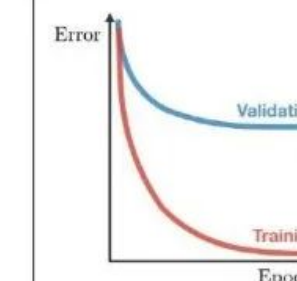
Mini-Batches

- Idealmente, calcularíamos o gradiente usando todos os dados de treinamento para cada atualização de peso, isso nos daria uma
 - Convergência estável para o mínimo;
 - Muito custoso computacionalmente;
- No outro extremo, apenas um exemplo de cada vez: Stochastic Gradient Descent (SGD).
 - Prós: Atualizações rápidas e frequentes; pode escapar de mínimos locais rasos.
 - Contras: Trajetória de convergência muito ruidosa (oscilante).
- Solução: Dividimos o dataset de treinamento em pequenos subconjuntos chamados batches (ex: 32, 64, 128 amostras). Calculamos o gradiente e atualizamos os pesos da rede para cada mini-batch.
 - Mais eficiente computacionalmente que o fazer tudo de uma vez (aproveita paralelização de hardware com GPUs e TPUs).
 - Convergência mais estável que o SGD.
 - Atualizações mais frequentes que o Batch GD, levando a uma convergência geralmente mais rápida.

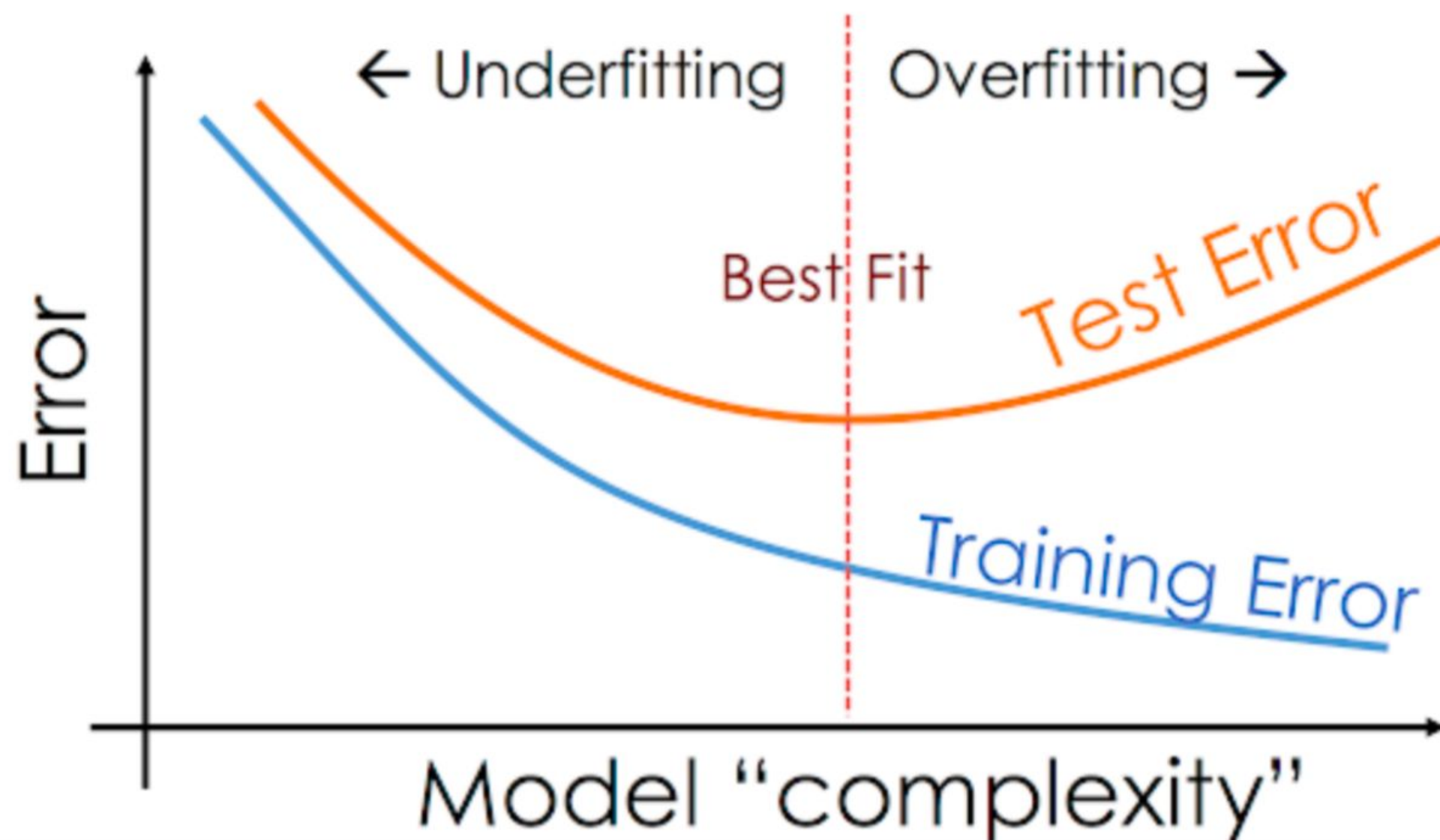
O Tamanho do Passo: Learning Rate



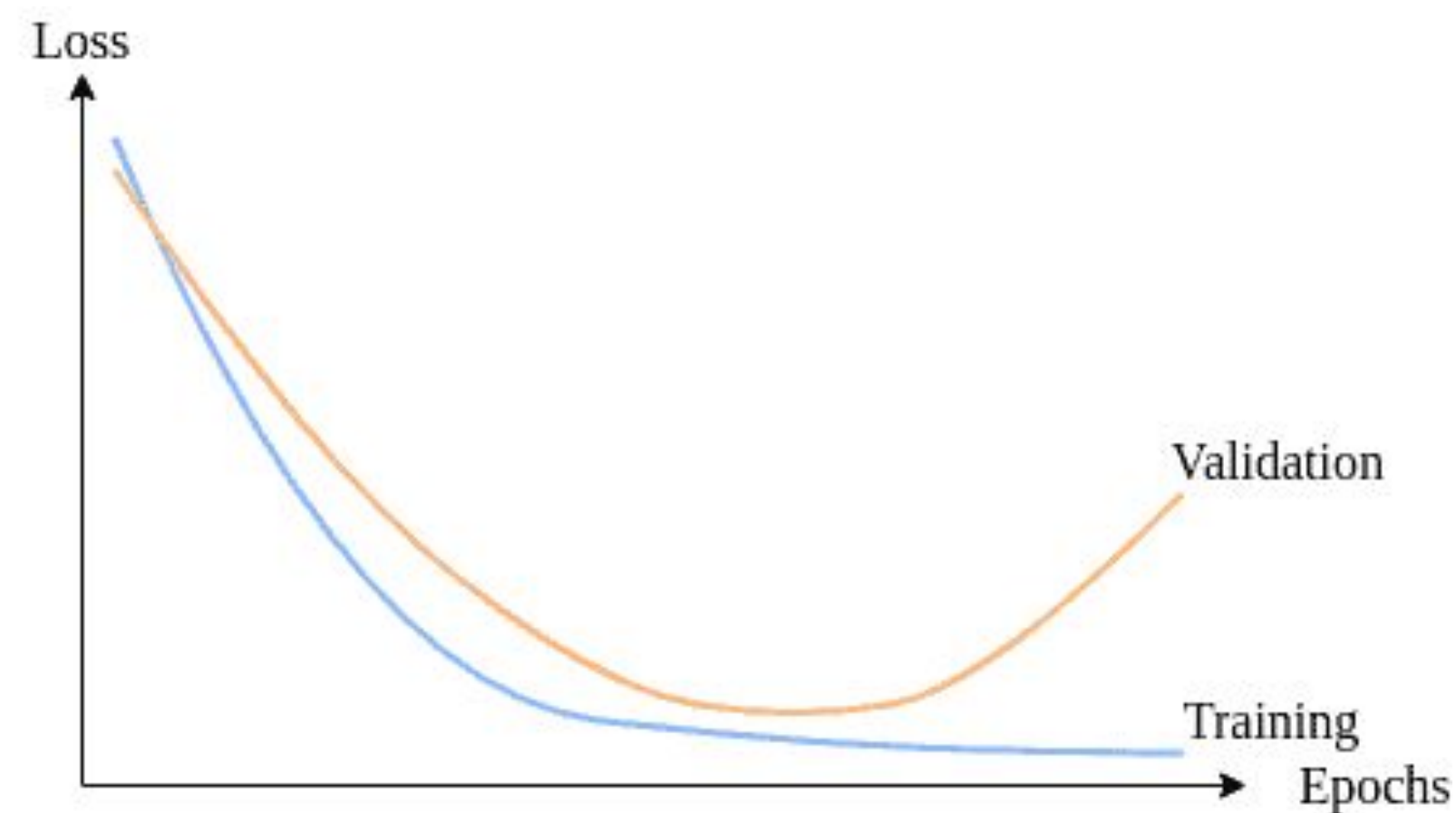
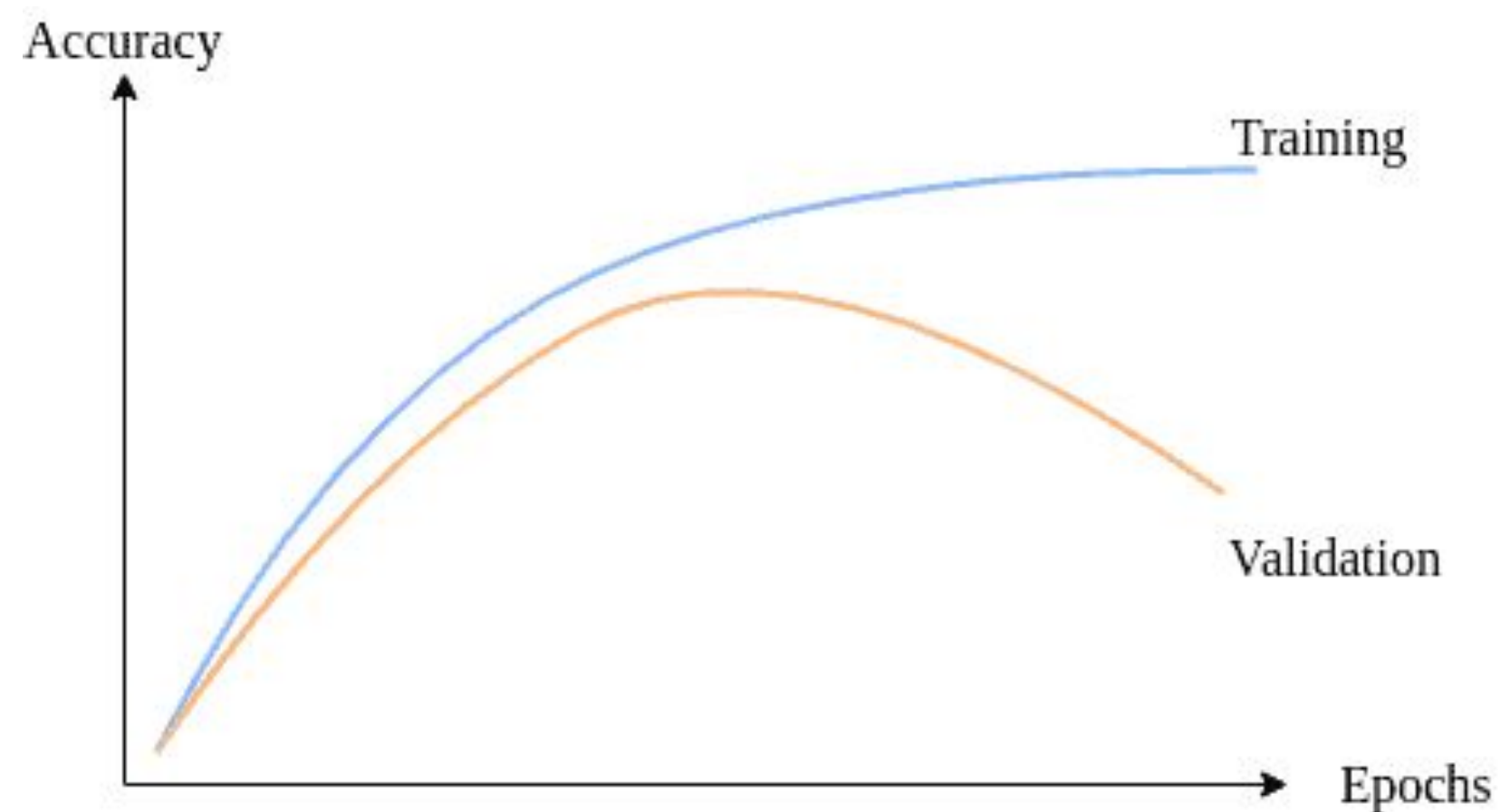
Underfitting e Overfitting

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"> - High training error - Training error close to test error - High bias 	<ul style="list-style-type: none"> - Training error slightly lower than test error 	<ul style="list-style-type: none"> - Low training error - Training error much lower than test error - High variance
Regression			
Classification			
Deep learning			
Remedies	<ul style="list-style-type: none"> - Complexify model - Add more features - Train longer 		<ul style="list-style-type: none"> - Regularize - Get more data

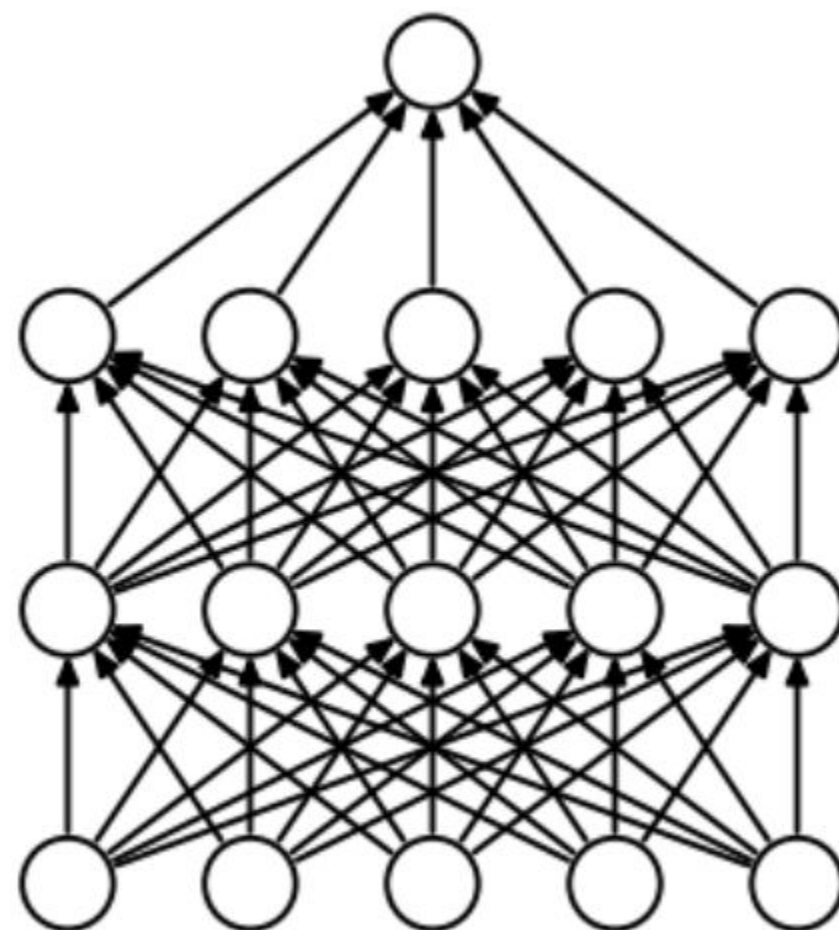
Underfitting e Overfitting - Arquitetura do Modelo



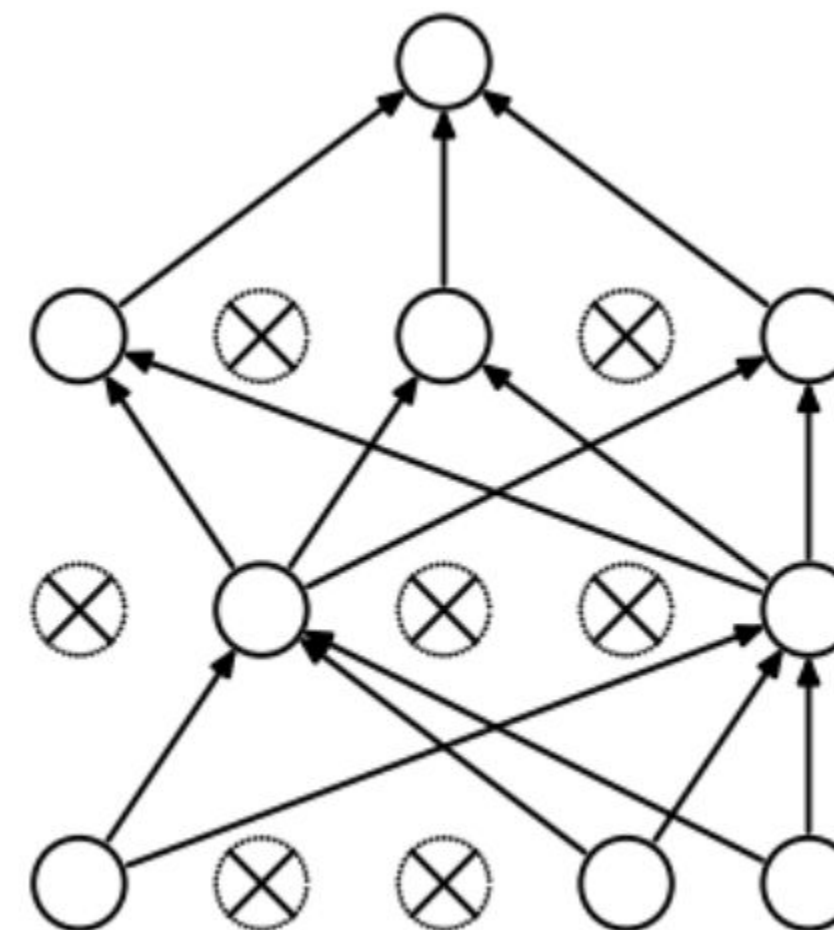
Underfitting e Overfitting - Treinamento



Dropout

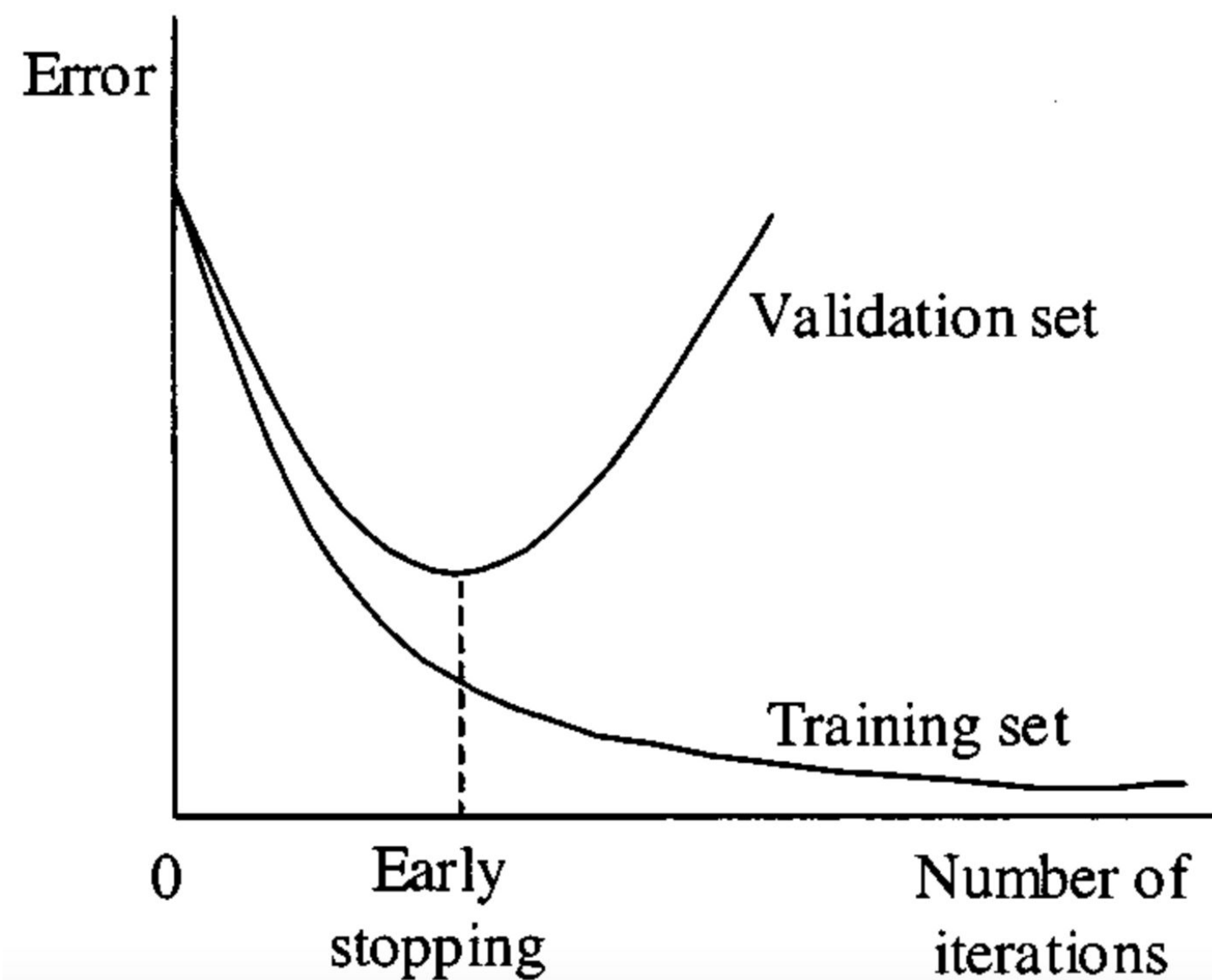


(a) Standard Neural Net



(b) After applying dropout.

Early Stopping



Regularização L1 e L2

- Adicionar penalidade aos pesos na função de custo para mantê-los pequenos.
- L1 (Lasso): Tende a encolher alguns pesos para exatamente zero. Isso efetivamente realiza uma "seleção de features", tornando o modelo mais esparsos (usa menos variáveis de entrada).
 - Útil quando se suspeita que muitas features são irrelevantes.
- L2 (Ridge): Tende a encolher todos os pesos, distribuindo a importância entre eles. Os pesos se aproximam de zero, mas raramente se tornam exatamente zero.
 - Geralmente preferida quando se acredita que todas as features contribuem de alguma forma.

L1 Regularization

$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2}_{\text{Loss function}} + \lambda \underbrace{\sum_{j=0}^M W_j^2}_{\text{Regularization Term}}$$

*Lambda é uma constante que controla a "intensidade" da regularização

Principais Frameworks

