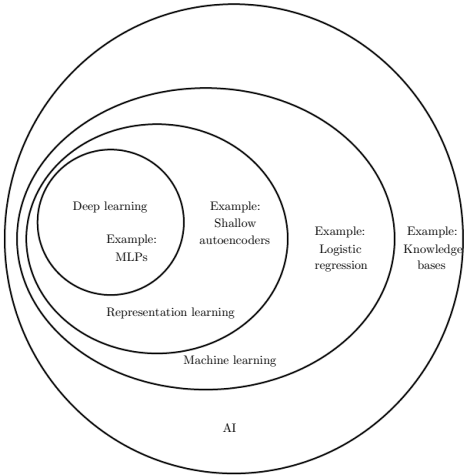
Machine Learning para Inteligencia Artificial

Redes Neuronales

Universidad ORT Uruguay

4 de Junio, 2025

Inteligencia Artificial (AI) - Machine learning (ML)



I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press, 2016.

Representación

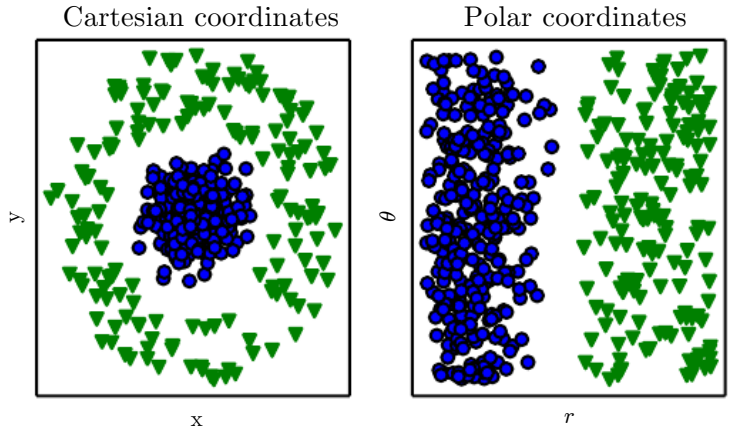
- Datos
- Hipótesis

Memorización

Inferencia/generalización

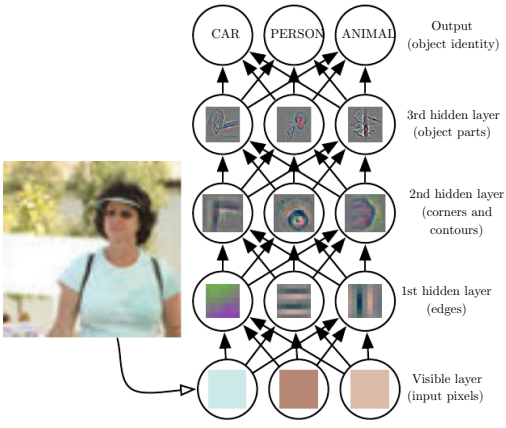
Representación

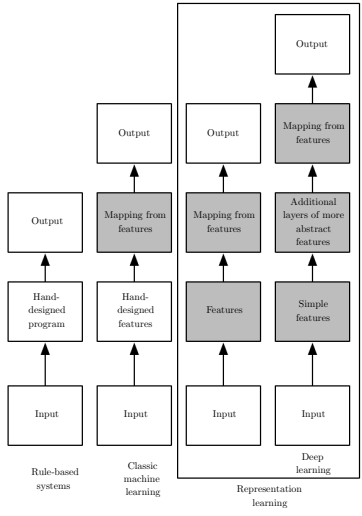
Punto $\longrightarrow$ sistemas de coordenadas



I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press, 2016.

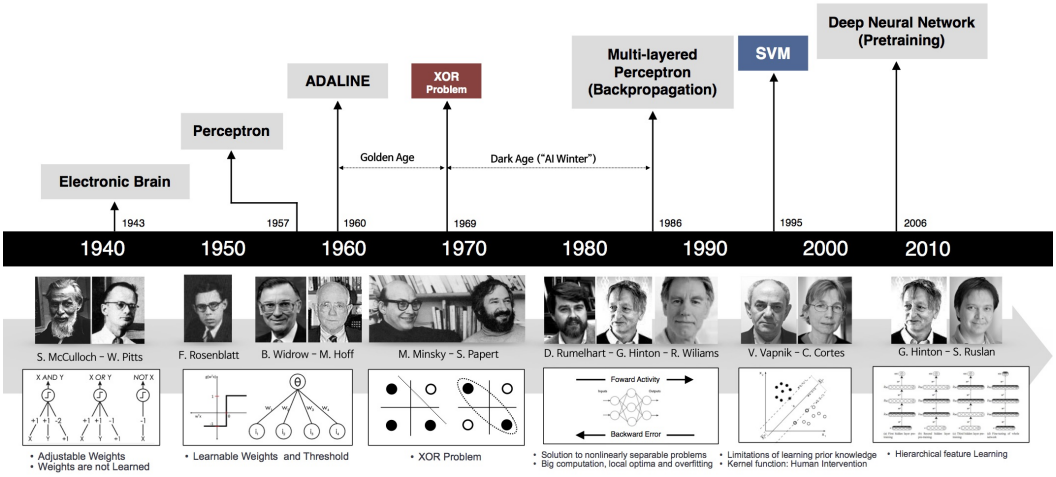
Representación composicional



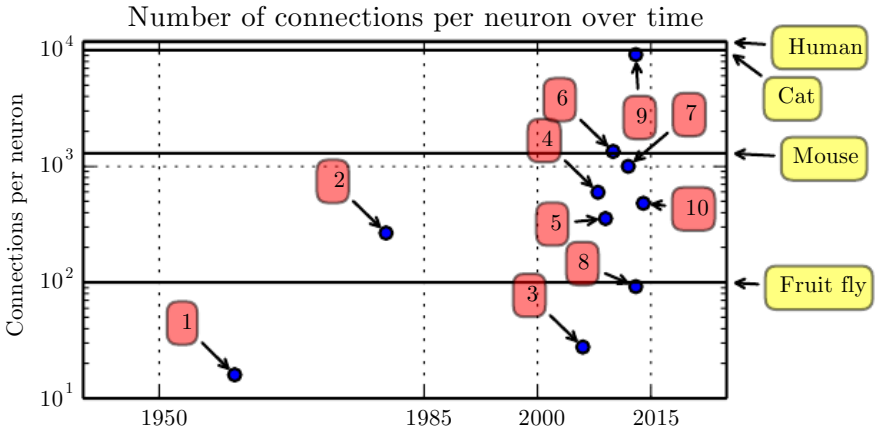


I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press, 2016.

Deep Learning: origen

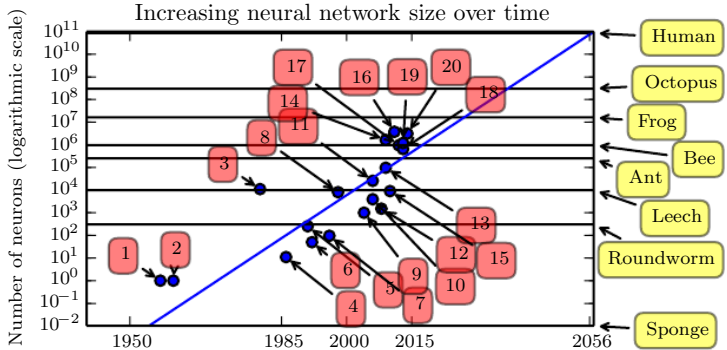


Artificial Neural Networks - Cantidad de connexiones



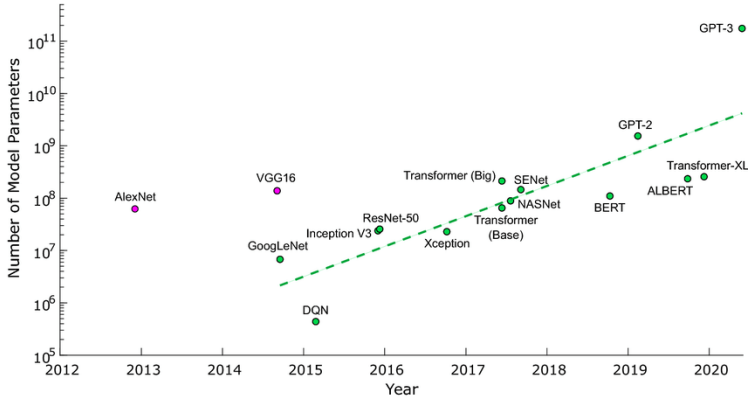
I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press, 2016.

Artificial Neural Networks - Cantidad de neuronas



I. Goodfellow, Y. Bengio, A. Courville. Deep Learning. MIT Press, 2016.

Artificial Neural Networks - Cantidad de parámetros



2015 VGG19 (Oxford): Convolutional, 1.45 10^8

2020 GPT-3 (OpenAI): Transformer, 1.75 10^{11}

Bernstein, Liane, et al. "Freely scalable and reconfigurable optical hardware for deep learning." Scientific reports 11.1 (2021): 3144.

Perceptron

F. Rosenblatt. The Perceptron: A Probabilistic Model For Information. Storage And Organization In The Brain. Psychological Review 65 (6): 386-408.

Fuente: https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

Perceptron: modelo de una neurona

- Modelo paramétrico:

$$\widehat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$$

  donde la función f está parametrizada por $\boldsymbol{\theta}$

- Regresión lineal - Modelo paramétrico básico de regresión

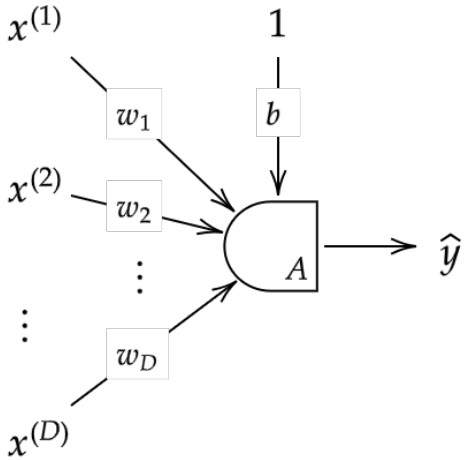
$$\widehat{y} = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_D x^{(D)} + b = \boldsymbol{x}^\top \boldsymbol{w} + b$$

- Regresión logística - Modelo paramétrico básico de clasificación

$$\widehat{y} = \text{Sigmoid}\left(w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_D x^{(D)} + b\right) = \text{Sigmoid}\left(\boldsymbol{x}^\top \boldsymbol{w} + b\right)$$

- En ambos modelos los parámetros son $\boldsymbol{\theta} = (\boldsymbol{w}, b)$

Perceptron: modelo de una neurona



Usando la notación matricial:

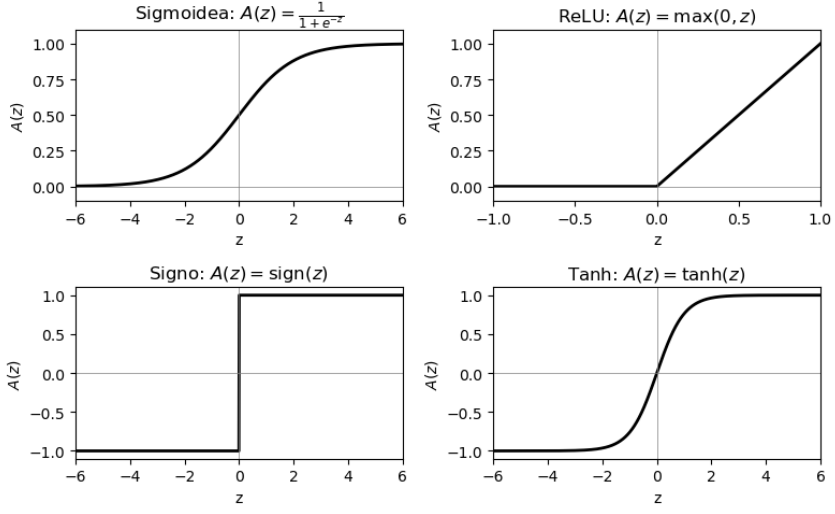
$$\hat{y} = A\left(\boldsymbol{x}^\top \boldsymbol{w} + b\right)$$

en donde

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_D \end{bmatrix} \qquad \boldsymbol{x} = \begin{bmatrix} x^{(1)} \\ \vdots \\ x^{(D)} \end{bmatrix}$$

$A : \mathbb{R} \to \mathbb{R}$ activación

Funciones de activación



Componente básica: Densa

El perceptron es un caso particular de una **componente básica**, llamada densa:

$$\widehat{\boldsymbol{y}} = \mathcal{D}(\boldsymbol{X}; \boldsymbol{W}, \boldsymbol{b}) = A\left(\boldsymbol{X}\boldsymbol{W} + b\right) \in \mathbb{R}^{N \times K}$$

con

$$\boldsymbol{X} \in \mathbb{R}^{N \times D},\ \boldsymbol{y} \in \mathbb{R}^{N \times K},\ \boldsymbol{W} \in \mathbb{R}^{D \times K},\ \boldsymbol{b} \in \mathbb{R}^{N \times K}$$

- el producto $\boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}$ es la unidad lineal
- A es la función de activación

Multi-layer Perceptron

- Un MLP de $d > 1$ *capas* se obtiene componiendo d capas *densas*:

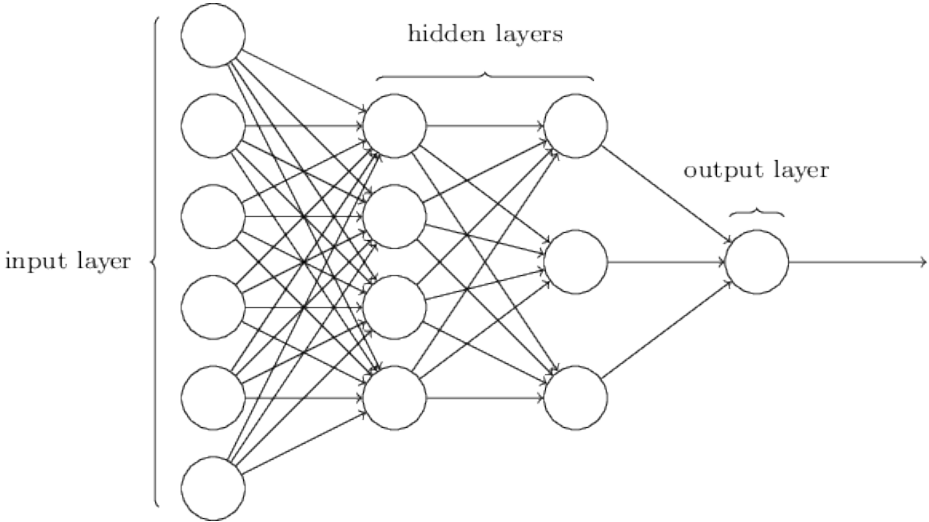
$$M(\boldsymbol{X}; \boldsymbol{W}_1, \ldots, \boldsymbol{W}_d, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_d) = D_d(D_{d-1}(\ldots, \boldsymbol{W}_{d-1}, \boldsymbol{b}_{d-1}); \boldsymbol{W}_d, \boldsymbol{b}_d)$$

- La dimensión de salida de D_i tiene que ser igual a la de entrada de D_{i+1}

- La cantidad de *parámetros* de M es:

$$\text{params}(M) = \sum_{i=1}^{d} \text{params}(D_i) = \sum_{i=1}^{d} D_i K_i + K_i \;\; \boldsymbol{W}_i \in \mathbb{R}^{D_i \times K_i}, \boldsymbol{b}_i \in \mathbb{R}^{1 \times K_i}$$

MLP: ejemplo gráfico



Funciones de Pérdida

- Regresión (MSE):

$$L(\widehat{y}, y) = (\widehat{y} - y)^2$$

- Clasificación binaria (BCE): la última función de activación es una **sigmoidea**

$$L(\widehat{y}, y) = -\Big(y \log \widehat{y} + (1 - y) \log(1 - \widehat{y}) \Big)$$

- Clasificación con K clases (CCE): la última función de activación es una **softmax** e $y \in \{0, 1\}^K$ (one-hot)

$$L(\widehat{y}, y) = -\sum_k y_k \log \widehat{y}_k$$

Costo Empírico

- Debemos elegir una función de pérdida $L(\widehat{y}, y)$
- Para un $(\boldsymbol{W}, \boldsymbol{b})$ dado, la costo empírico es:

$$J(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} L(M(\boldsymbol{x}_i; \boldsymbol{W}, \boldsymbol{b}), y_i)$$

  donde $M(\boldsymbol{x}_i; \boldsymbol{W}, \boldsymbol{b})$ es la salida de la red para la entrada $\boldsymbol{x}_i$.

- Entrenar la red significa encontrar el argumento mínimo de J (ERM):

$$(\widehat{\boldsymbol{W}}, \widehat{\boldsymbol{b}}) = \underset{(\boldsymbol{W}, \boldsymbol{b})}{\arg\min}\, J(\boldsymbol{W}, \boldsymbol{b}).$$

ERM: entrenamiento del MLP

- Problema de optimización - variantes de **descenso por gradiente**:

$$(\boldsymbol{W}_0, \boldsymbol{b}_0) = \text{inicializar random}$$
$$(\boldsymbol{W}_{k+1}, \boldsymbol{b}_{k+1}) = (\boldsymbol{W}_k, \boldsymbol{b}_k) - \eta \cdot \nabla J(\boldsymbol{W}_k, \boldsymbol{b}_k)$$

  donde η es la **tasa de aprendizaje**

- Debemos calcular el gradiente de J en cada paso.

- Linealidad:

$$\nabla J(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(M(\boldsymbol{x}_i; \boldsymbol{W}, \boldsymbol{b}), y_i) = \underset{(\boldsymbol{x},y)\sim T}{\boldsymbol{E}} [\nabla L(M(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}), y)]$$

ERM: entrenamiento del MLP

Algoritmo básico: Descenso del gradiente estocástico (SGD)

- Comenzar con $(\boldsymbol{W}_0, \boldsymbol{b}_0)$ random
- En cada step t: seleccionar $(x, y) \in T$ aleatoriamente y actualizar

$$(\boldsymbol{W}_{t+1}, \boldsymbol{b}_{t+1}) = (\boldsymbol{W}_t, \boldsymbol{b}_t) - \eta \nabla L(M(\boldsymbol{x}; \boldsymbol{W}_t, \boldsymbol{b}_t), y) \qquad \eta \text{ es el learning rate}$$

En la práctica se actualiza por **batch** (se aprovecha el paralelismo)

Se usa una técnica llamada **back-propagation** basada en la regla de derivación de la cadena $d(f \circ g) = df \cdot dg$ para entrenar un MLP de muchas capas

1986 Rumelhart, Hinton & Williams introducen **Backpropagation Algorithm**



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton†
& Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

Nature. 323 (6088): 533-536.

Backpropagation (Opcional)

Objetivo: Calcular $\nabla L(\widehat{y}, y)$ eficientemente para un MLP con K capas.

- $\boldsymbol{z}_k = \boldsymbol{W}_k \boldsymbol{a}_{k-1} + \boldsymbol{b}_k$ (pre-activación)
- $\boldsymbol{a}_k = \sigma(\boldsymbol{z}_k)$ (post-activación)
- $\boldsymbol{a}^0 = \boldsymbol{x}$

Forward pass:

- Para $k = 1$ hasta K: calcular $\boldsymbol{z}_k$ y $\boldsymbol{a}_k$

Backward pass (errores):

- Calcular $\delta_K = \sigma'(\boldsymbol{z}_K) \nabla_{\boldsymbol{a}_K} L$
- Para $k = K - 1$ hasta 1: $\delta_k = \sigma'(\boldsymbol{z}_k) \boldsymbol{W}_{k+1}^\top \delta_{k+1}$

Backpropagation (Opcional)

Gradientes:
Para cada capa $k = 1, \ldots, K$:

$$\nabla_{\boldsymbol{W}_k} L = \delta_k \cdot \boldsymbol{a}_{k-1}^\top$$
$$\nabla_{\boldsymbol{b}_k} L = \delta_k$$

Actualización (ej. SGD):

$$\boldsymbol{W}_k \leftarrow \boldsymbol{W}_k - \eta \nabla_{\boldsymbol{W}_k} L, \quad \boldsymbol{b}_k \leftarrow \boldsymbol{b}_k - \eta \nabla_{\boldsymbol{b}_k} L$$

Regularización

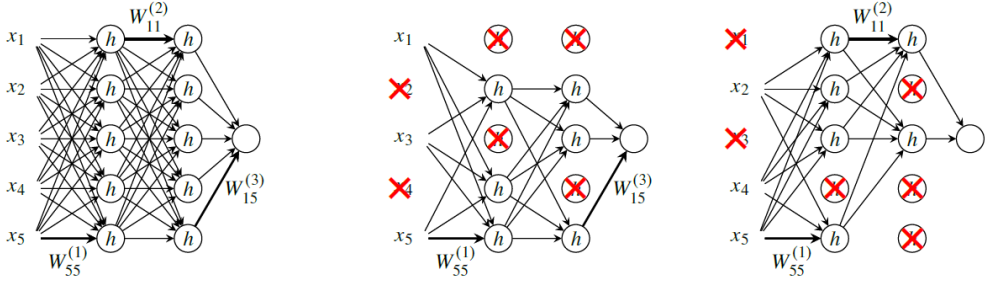
Se usan diferentes regularizadores:

- Ridge (norma l_2), lasso (norma l_1)
- Dropout: no actualizar ciertos coeficientes seleccionados aleatoriamente (efecto "ensemble")
- **Early stopping**: usar un conjunto de validación V y parar de entrenar cuando el error en V no decrece por un cierto tiempo
- Data augmentation: generar datos a través de transformaciones (típico en análisis de imágenes: rotación, zoom, ...)

Dropout

Entrenamiento

Máscara que apaga aleatoriamente neuronas de una capa con probabilidad $1 - r$



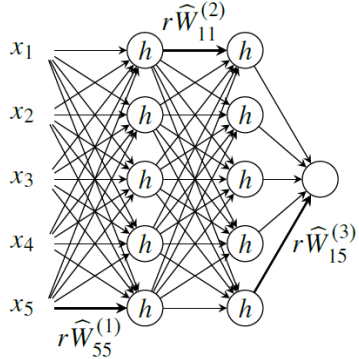
(a) A standard neural network

(b) Two sub-networks

Dropout

Predicción

Se multiplican los pesos por la probabilidad r



Early Stopping

Early Stopping

```
01: Inicializar W_best ← random
02: Inicializar L_best ← +∞
03: Inicializar p paciencia
04: Inicializar c ← 0 contador paciencia
05:
06: Para cada epoch en epochs:
07:     Entrenar red con datos de entrenamiento
08:     L ← evaluar red con datos de validación
09:     Si L < L_best:
10:         L_best ← L
11:         W_best ← pesos actuales de la red
12:         c ← 0
13:     Sino:
14:         c ← c + 1
15:         si c ≥ p: break
16: Establecer pesos de la red ← W_best
```

Bibliografía

- Deep Learning. Ian Goodfellow, Yoshua Bengio, Aaron Courville. MIT Press, 2016.
- Prince, Simon JD. Understanding deep learning. MIT press, 2023.
- Bishop, Christopher M., and Hugh Bishop. Deep learning: Foundations and concepts. Springer Nature, 2023.
- Scardapane, Simone. "Alice's Adventures in a Differentiable Wonderland–Volume I, A Tour of the Land." arXiv preprint arXiv:2404.17625 (2024).