

Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

November 24, 2025

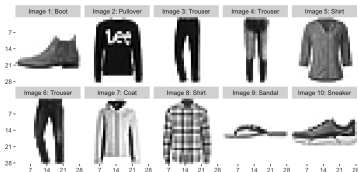


- ① Introduction
- ② Multi-Layer Perceptron (MLP)
- ③ Neural Networks
- ④ Neural Networks as Universal Approximators
- ⑤ Training Neural Networks
- ⑥ References

- 1 Introduction
- 2 Multi-Layer Perceptron (MLP)
- 3 Neural Networks
- 4 Neural Networks as Universal Approximators
- 5 Training Neural Networks
- 6 References

Why Neural Networks?

- We can find explicit formulas for some problems (no machine learning)
 - $\Delta x = \frac{1}{2} a \cdot t^2 + v_0 \cdot t$
- We can model some problems by assuming simple relationships (classical machine learning)
 - House price as a linear function of its features
 - $y = a_1 \cdot x_1 + a_2 \cdot x_2 + \dots + a_p \cdot x_p$
- How can we classify these images?

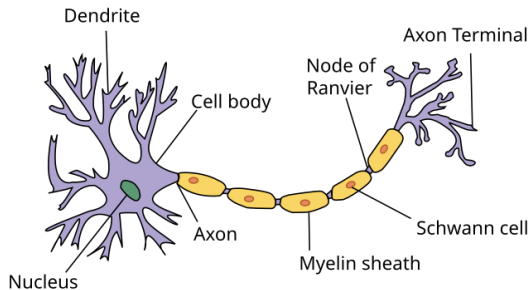


Why Neural Networks? Cont.

- **No explicit formula** exists to recognize a sneaker
- We intuitively recognize any sneaker
- Our brains use a **complex function** for this recognition
- **Deep neural networks** can learn this complex function

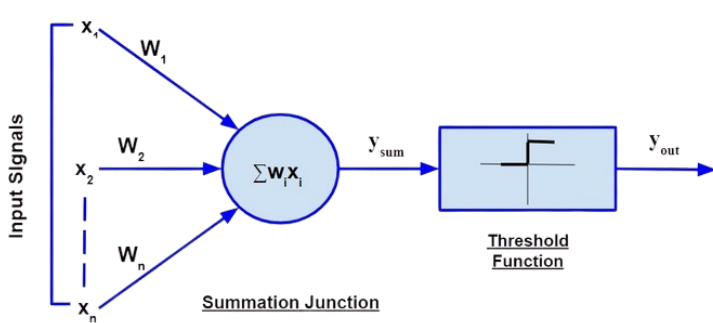


Modelling the Brain



- Building units are neurons.
- **Dendrite:** Receives signals from other neurons neurons.
- **Soma:** Processes the information
- **Axon:** Transmits the output of this neuron
- **Synapse:** Point of connection to other neurons

McCulloch-Pitts Neurons

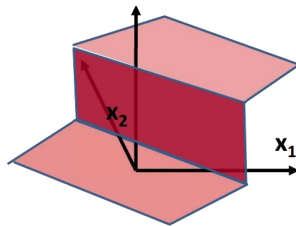
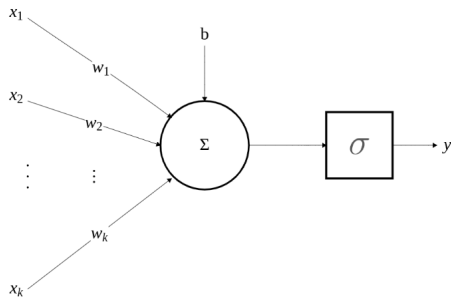


$$y = \begin{cases} 1 & \text{if } \sum_i w_i x_i - T \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

Perceptron Reminder

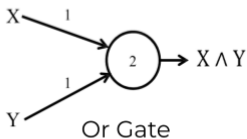
The building block of each neural network is the perceptron:

- $\{x_1, x_2, \dots, x_k\}$: input features
- $\{w_1, w_2, \dots, w_k\}$: feature weights
- b : bias term
- $\sigma(\cdot)$: activation function
- y : output of the neuron

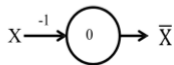


Example: Perceptron as a Boolean Gate

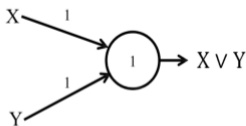
And Gate



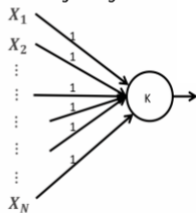
Not Gate



Or Gate

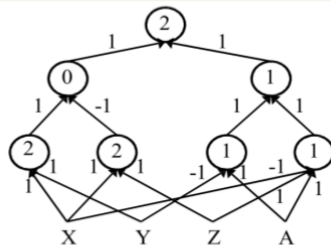


Majority Gate



A more complex example

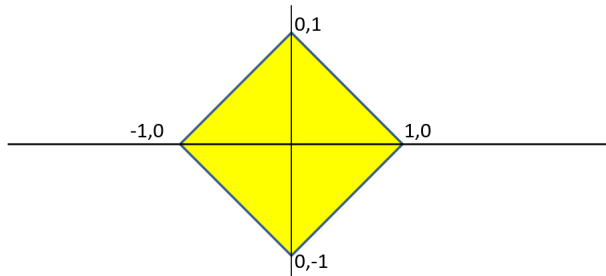
$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | (\bar{X} \& \bar{Z}))$$



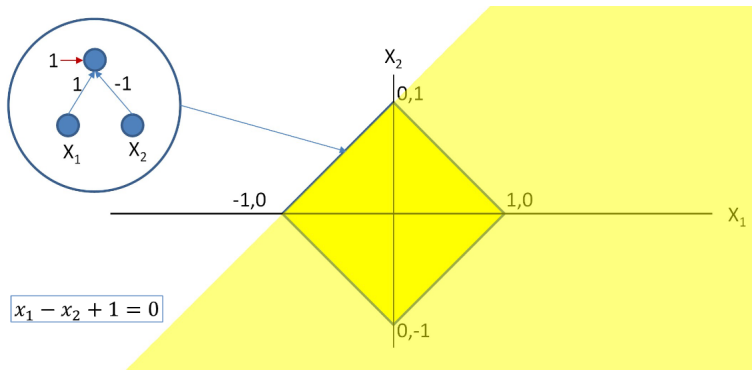
- A perceptron can model any simple binary Boolean gate.
- MLPs are universal Boolean functions.

Example: MLP for Complex Patterns

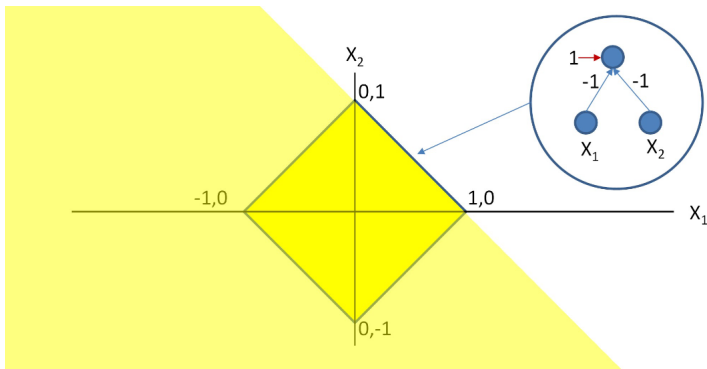
- What network to learn this area?
- Example is adapted from [1].



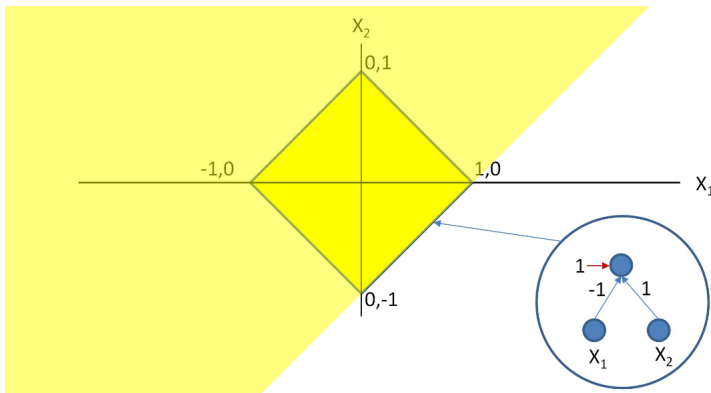
Example: MLP for Complex Patterns Cont.



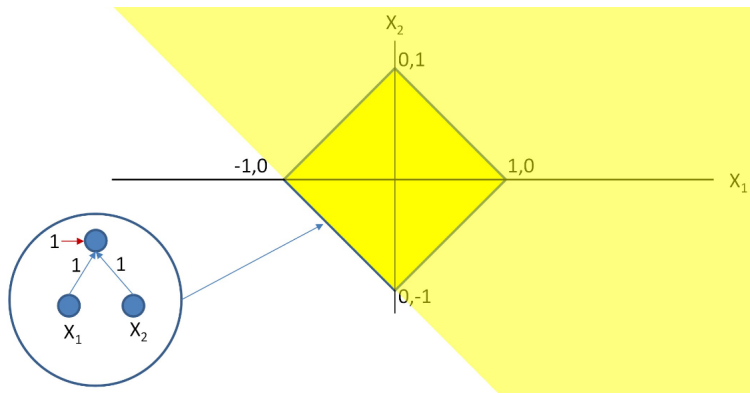
Example: MLP for Complex Patterns Cont.

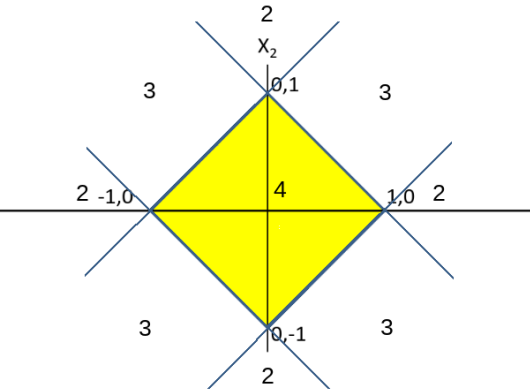


Example: MLP for Complex Patterns Cont.






Example: MLP for Complex Patterns Cont.



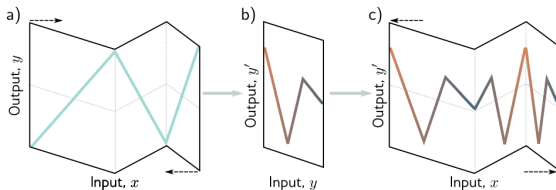
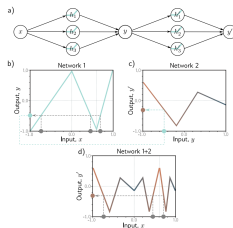


Neural Networks for Region Classification

Layer Type	Decision Region	Interpretation	Visualization
Single layer (perceptron)	Half-space	Linear separator defined by a hyperplane $w^\top x + b = 0$	
Two layers (1 hidden)	Closed, convex regions	Intersections of half-spaces \Rightarrow convex polytopes	
Three layers (2+ hidden)	Arbitrary (finite unions)	Union of polytopes; universal approximation of regions	

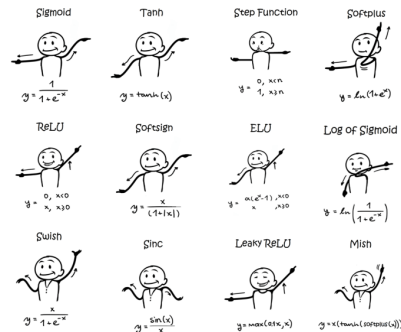
Example: Composing Neural Networks

- Each layer forms a simple piecewise-linear function (ReLU units).
- Network 1 maps $x \rightarrow y$; Network 2 maps $y \rightarrow y'$.
- Composing layers increases the number of linear regions.
- The full MLP produces a more complex nonlinear function than either layer alone.



MLP Capacity

- Increasing **width and depth** allow us to approximate **complex decision boundaries**
- An **activation function** makes a neuron's output **non-linear**, allowing the network to learn complex pattern
- It is **not limited** to Boolean or step functions
- With appropriate activation functions, neural networks can **approximate any real-valued function** (More details later)



Adapted from Sefiks

- 1 Introduction
- 2 Multi-Layer Perceptron (MLP)
- 3 Neural Networks**
- 4 Neural Networks as Universal Approximators
- 5 Training Neural Networks
- 6 References

Single Hidden Layer Neural Network

- Hidden layer pre-activation:

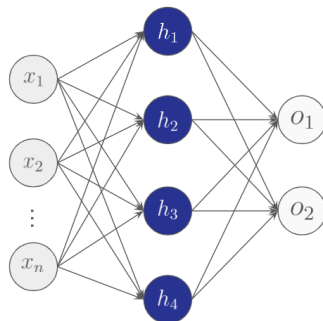
$$a_i(x) = b_i^{(1)} + \sum_j W_{ij}^{(1)} \cdot x_j$$

- Weight between neuron i and j in layer ℓ : $W_{ij}^{(\ell)}$
- Activated hidden layer:

$$h^{(1)}(x) = \sigma^{(1)}(a(x))$$

- Output layer:

$$o(\mathbf{x}) = \sigma^{(2)}(\mathbf{b}^{(2)} + \mathbf{W}^{(2)} h^{(1)}(\mathbf{x}))$$



input layer hidden layer output layer

- Let $h_i^0 = x_i$ for $i \in \{1, 2, \dots, n\}$
- For $\ell \in \{0, 1, \dots, L\}$:

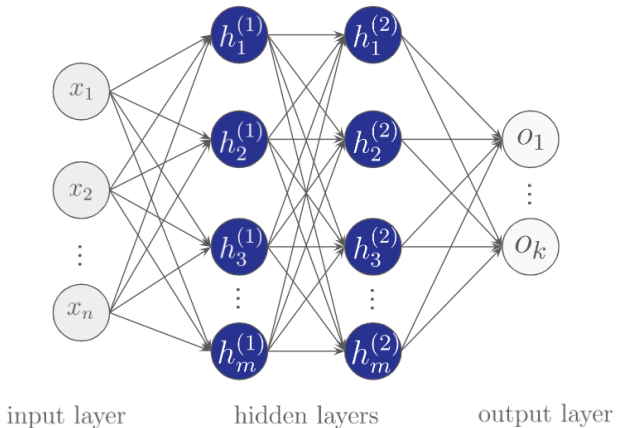
$$h_j^{(\ell)} = \sigma^{(\ell)}(a_j^{(\ell)})$$

- Learnable parameters:

$$b_j^{(\ell)}, W_{ij}^{(\ell)}$$

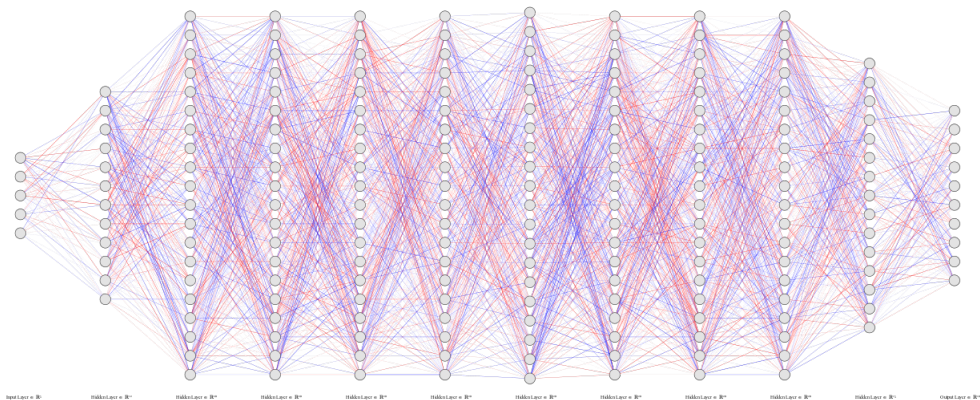
- Number of learnable parameters:

$$(n+1)m_1 + (m_1+1)m_2 + \dots + (m_L+1)k$$



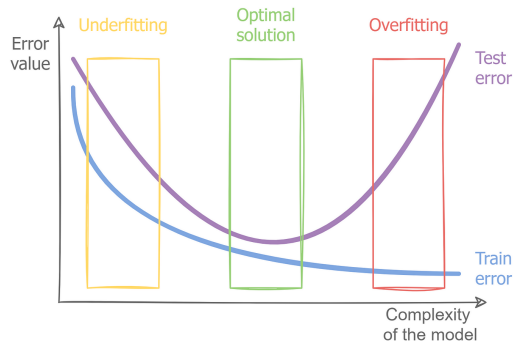
Deep Neural Network Architecture

- More than a few hidden layers: **Deep Neural Network (DNN)**
- Designing neural network architecture is **more of an art than a science**.



Network Width and Depth

- **Width:** More neurons, more complexity
- **Depth:** More layers, more abstraction
- **Balance:**
 - Too narrow/shallow: risk of underfitting
 - Too wide/deep: risk of overfitting



Adapted from Towards Data Science

Universal Approximation Theorem

Key Concept

- The Universal Approximation Theorem states that a feedforward neural network with:
 - A single hidden layer
 - Sufficient number of hidden neurons
 - Appropriate activation functions (e.g., sigmoid)

Can approximate any continuous function on a **compact subset** of \mathbb{R}^n to any desired accuracy.

Understanding Compact Sets

What is a Compact Set?

- In the context of the Universal Approximation Theorem, approximation is guaranteed on a **compact subset** of \mathbb{R}^n .
- A set is compact if it is both:
 - **Bounded:** Enclosed within a finite space.
 - **Closed:** Contains all its boundary points.
- Compact sets ensure certain mathematical properties that enable reliable function approximation by the MLP within that region.

Breaking Down Complex Functions

- Idea: Complex functions can be decomposed into multiple smaller parts, each represented by a simpler function.
- By combining a series of simpler functions (like square pulses), the target function can be closely approximated.

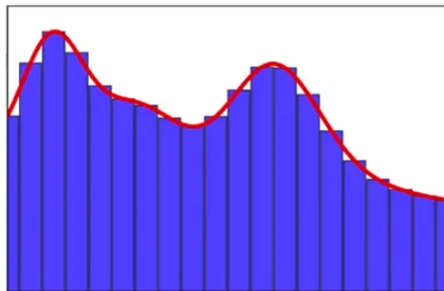
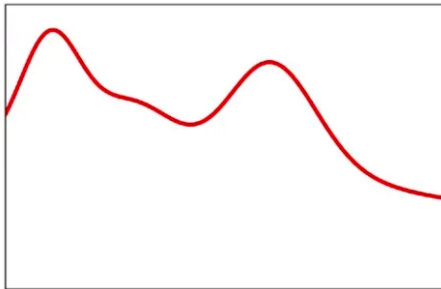
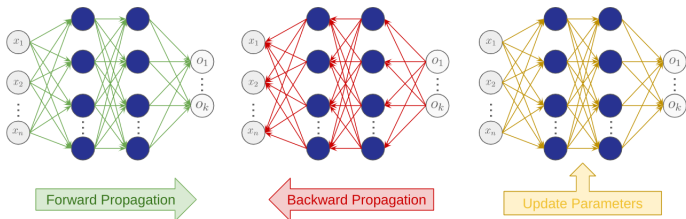


Figure adapted from Niranjn Kumar, Illustrative Proof of Universal Approximation Theorem

Training Phases

- **Initialize weights and biases:** These values control how the network initially processes information (more details later)
- **Forward pass:** Pass the input through the network to get an output
- **Calculate the error:** Compare the network's output to the correct answer to measure the difference (called the 'loss' – more details later)
- **Backpropagation:** Use the loss value to adjust the weights and biases to improve the network's accuracy

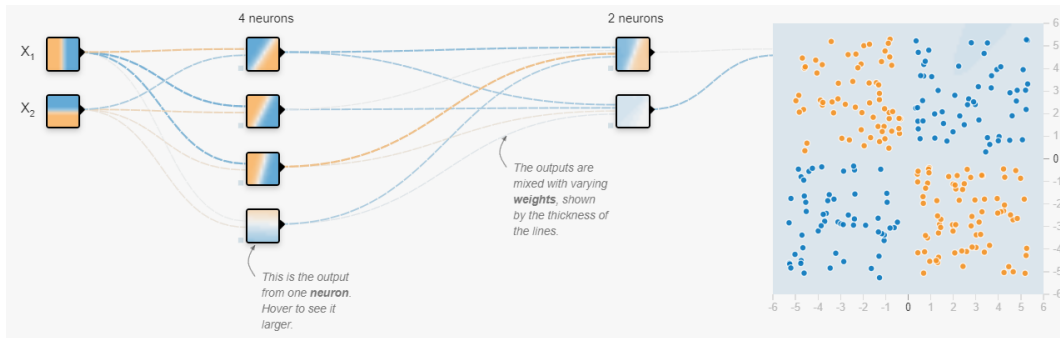


Forward Propagation

- This is the pass where we send input data through the network to make a prediction (likely inaccurate at first).
- The prediction is made by calculating weighted sums and applying an activation function in each layer

$$o(x) = a^{(L)} = \sigma^{(L)} \left(b^{(L)} + W^{(L)} \sigma^{(L-1)} \left(\dots \sigma^{(1)} (b^{(1)} + W^{(1)} x) \dots \right) \right)$$

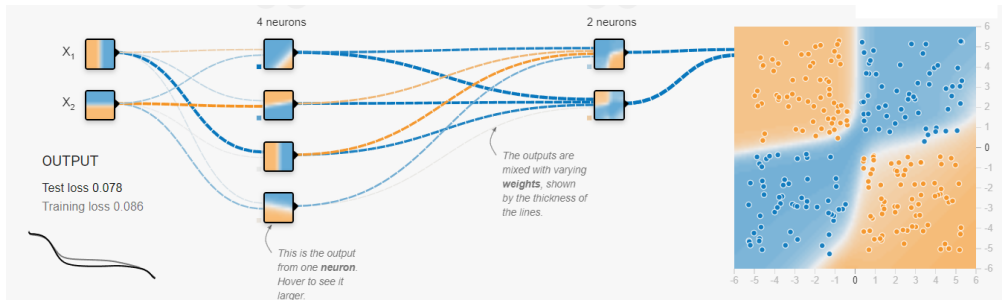
Forward Propagation Cont.



Before making predictions. Adapted from TensorFlow playground: Daniel Smilkov and Shan Carter.

Forward Propagation Cont.

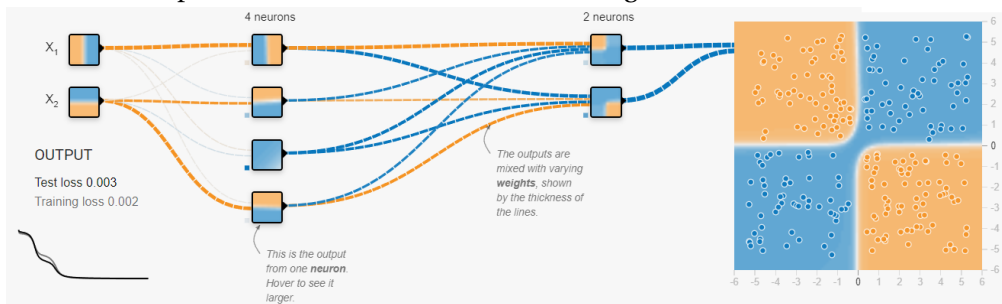
- The goal is to adjust the network's parameters to improve the predictions
- The loss is calculated after the forward pass, indicating how far off our predictions are from the true values



Loss values for predictions. Adapted from TensorFlow playground: Daniel Smilkov and Shan Carter.

BackPropagation and Parameter Update

- The network uses the **loss** to adjust its **weights and biases** through a process known as **backpropagation**
- Backpropagation calculates how much weights should change to reduce the error
- This will be explained in more detail in the following lecture



Predictions improve as the weights get updated. Adapted from TensorFlow playground: Daniel Smilkov and Shan Carter.

- 1 Introduction
- 2 Multi-Layer Perceptron (MLP)
- 3 Neural Networks
- 4 Neural Networks as Universal Approximators
- 5 Training Neural Networks
- 6 References

Contributions

These slides are authored by:

- Sogand Salehi
- Erfan Sobhaei

- [1] R. Ramakrishnan, “Deep learning course at carnegie mellon university.” <https://deeplearning.cs.cmu.edu/F23/index.html>, 2023.
Accessed: 2024-09-04.
- [2] E. Mousavi and K. Alishahi, “Deep learning course at sharif university of technology.” [https://https://dnncourse.github.io/lectures](https://dnncourse.github.io/lectures), 2023.
Accessed: 2024-09-04.
- [3] D. Smilkov and S. Carter, “A neural network playground.” playground.tensorflow.org, 2022.
Accessed: 2024-10-14.
- [4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [5] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques for Building Intelligent Systems*. O'Reilly Media, 2019.