

Nanostructured Materials

Artificial Neural Networks

TU Dresden · June 24, 2025

Attila Cangi · a.cangi@hzdr.de

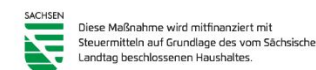
INSTITUTE OF



PARTICIPATING INSTITUTIONS



FUNDED BY



What we will cover today

- 1 — Recap and introduction
- 2 — Perceptron model
- 3 — From perceptrons to multi-layer neural networks
- 4 — Activation functions
- 5 — Universal approximation theorem
- 6 — Forward propagation
- 7 — Training neural networks: Stochastic gradient descent, gradient update, backpropagation
- 8 — Limitations of neural networks and regularization
- 9 — Key takeaways

1 — Recap and introduction

1 — Recap and introduction

What is machine learning?

“The use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyse and draw inferences from patterns in data.”

This process often involves three main steps

- Input data
- Training
- Prediction

Key categories of machine learning

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Machine learning model

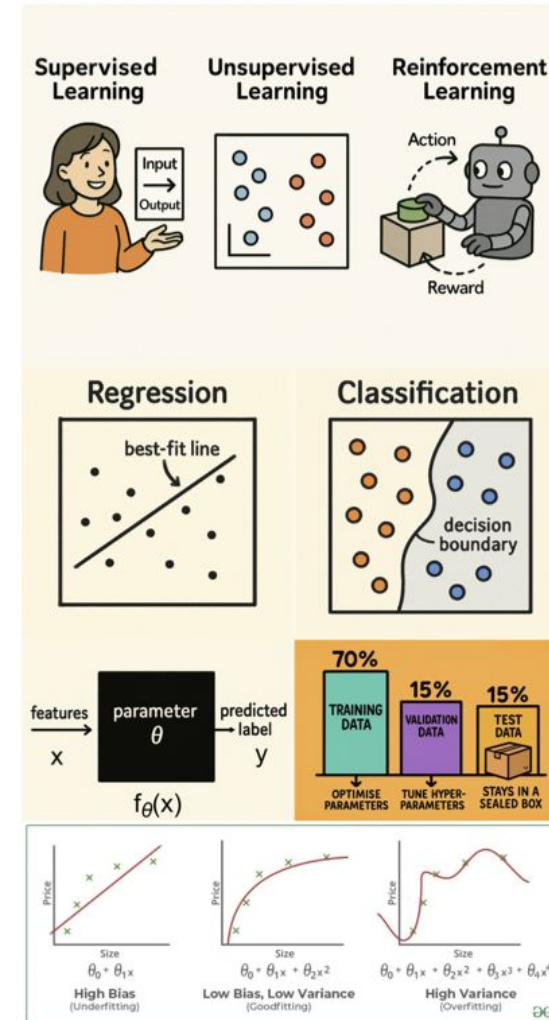
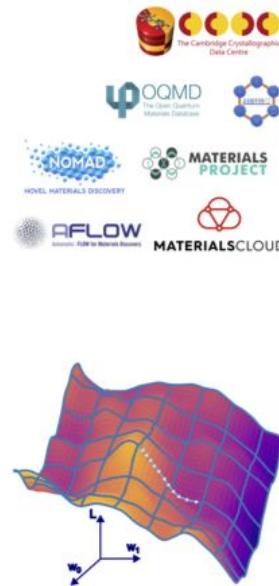
$$\mathbf{y} = M(\mathbf{x})$$

Output Input

1 — Recap and introduction

Machine learning basics

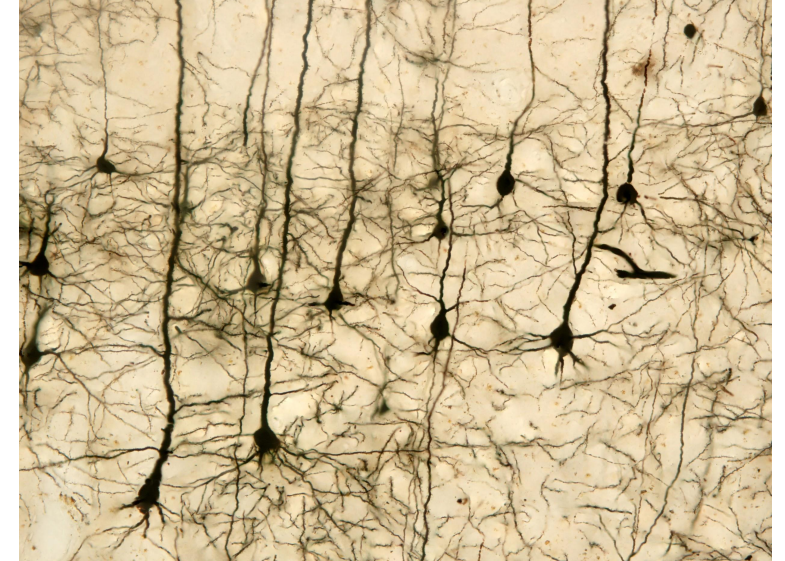
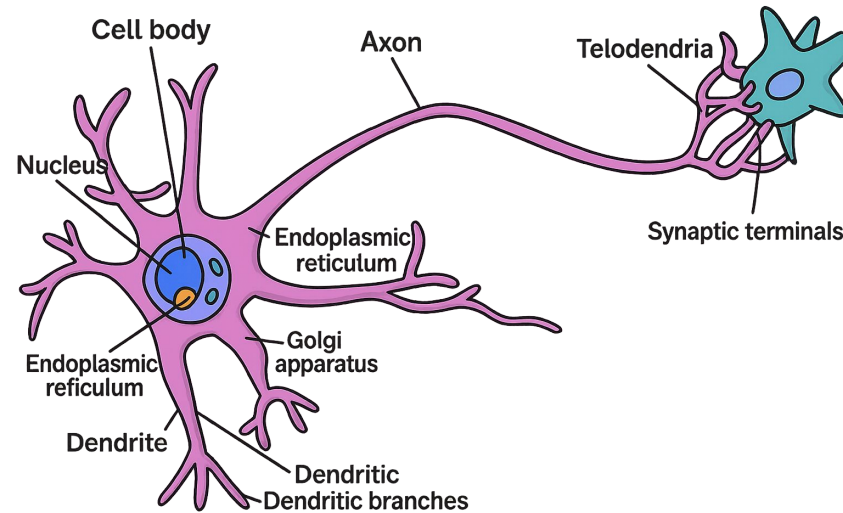
- ▶ Core **vocabulary**: dataset, feature, label, model, loss
- ▶ ML **paradigms**: supervised, unsupervised, reinforcement
- ▶ Regression vs. classification; MSE loss; linear-regression baseline
- ▶ Model **evaluation**: train/val/test, metrics, residuals
- ▶ Diagnose **under-/over-fit**; Ridge & Lasso for regularization



1 — Recap and introduction

Biological neurons

- Neurons are the basic functional unit of the nervous system
- Neurons receive, process and transmit information through electrical and chemical signals
- A neuron is made up of three main parts:
 - **Cell body (soma)**
 - **Dendrites**
 - **Axon**



Neurons in the cerebral cortex.

<https://www.pnas.org/post/journal-club/predictions-could-help-neurons-and-brain-learn>

1 — Recap and introduction

Biological neurons

The function of a neuron can be divided into several steps:

- **Reception of signals**

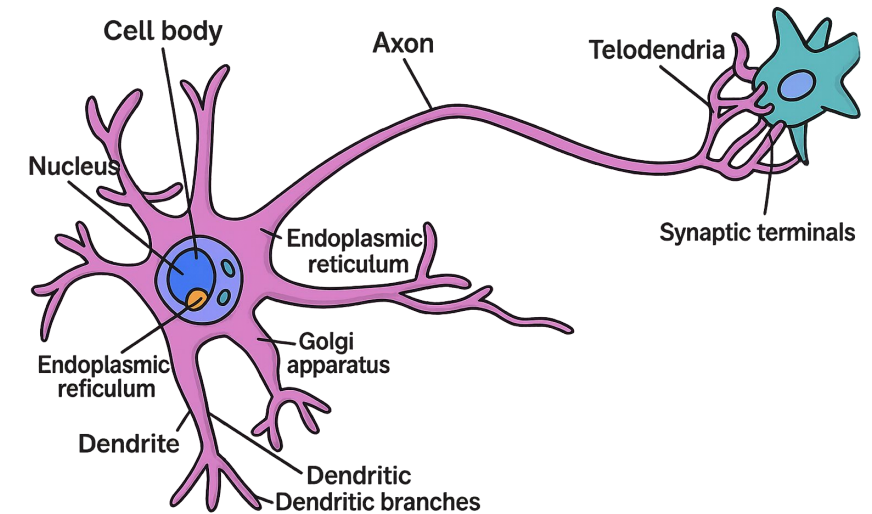
Dendrites receive chemical signals from neighboring neurons. These signals lead to changes in the membrane potential of the neuron.

- **Generation of an action potential**

When the membrane potential reaches a certain threshold, an action potential is triggered. This is a rapid change in the membrane potential that spreads along the axon.

- **Transmission of the action potential**

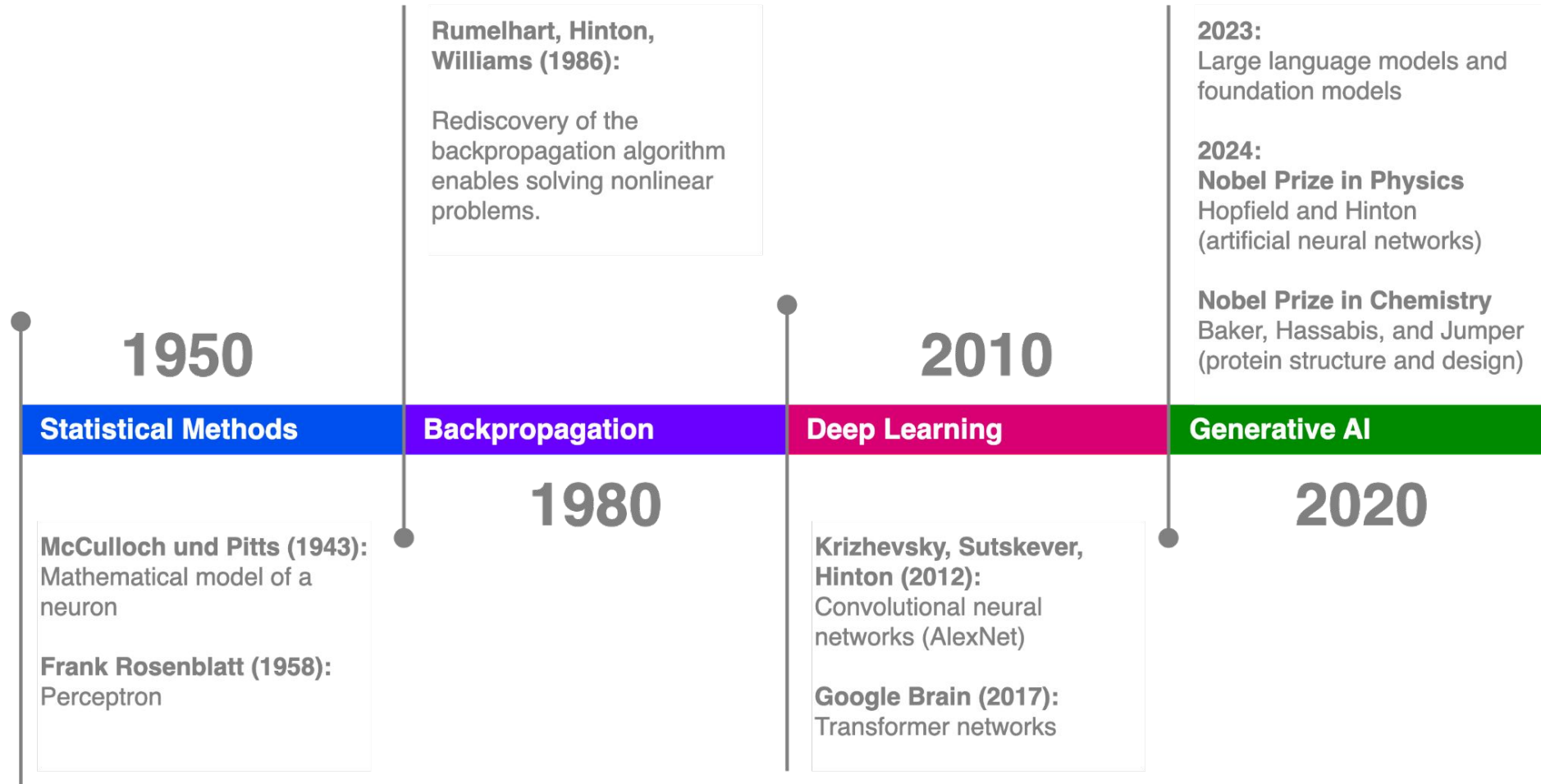
The action potential travels along the axon to the axon terminals. This occurs through a series of depolarizations and repolarizations of the cell membrane.



This variability of information transmission is reflected in the **weights of artificial neural networks**.

1 — Recap and introduction

A brief history of neural networks

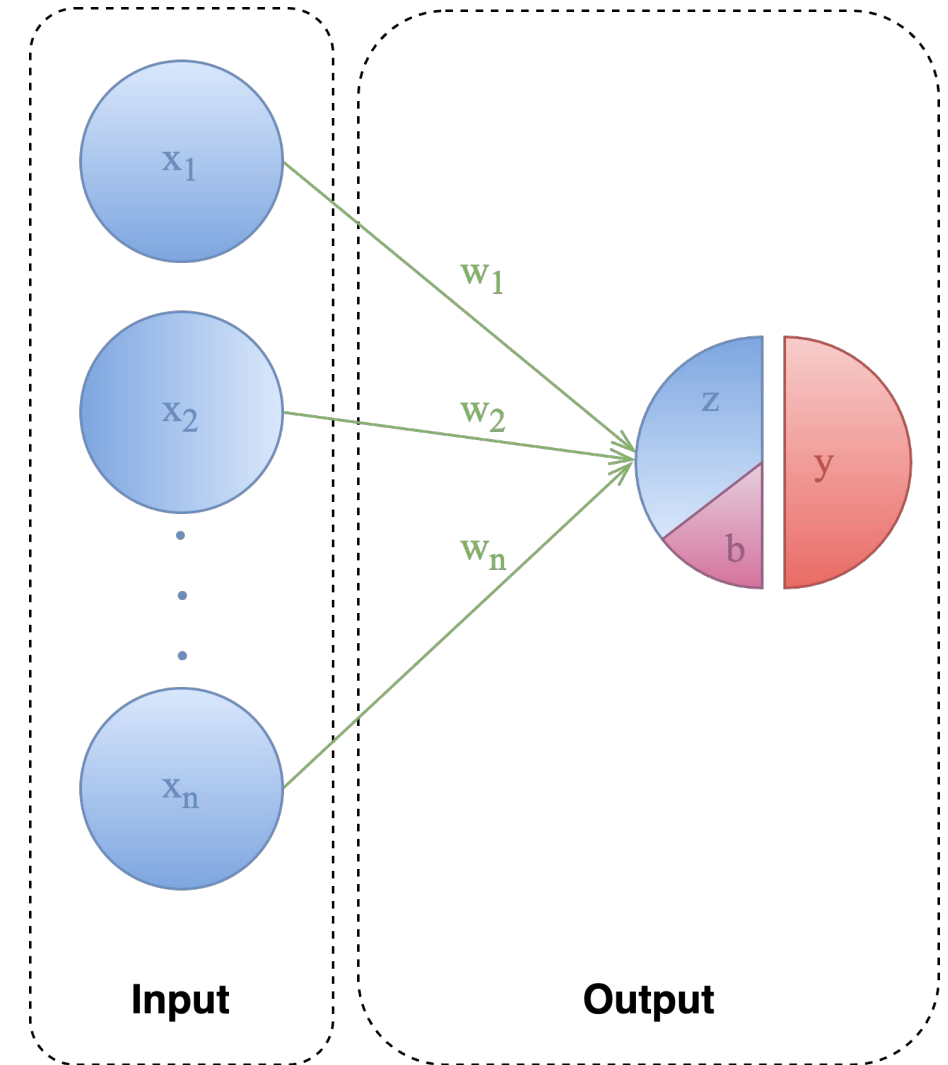


2 — Perceptron model

2 — Perceptron model

Structure

- The simple perceptron model consists of an input layer and an output layer
- In the input layer, the input is (x_1, x_2, \dots, x_n)
- The output layer consists of a single neuron. It contains the network input (z) and the output value (y)
- This network can be used for binary classification, i.e. the network can decide for an input whether it belongs to a certain category
- The classification is expressed by the output value (y)



2 — Perceptron model

Forward propagation

- Input

$$\vec{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad n \in \mathbb{R}, \quad x_i \in \mathbb{R}$$

- Weights

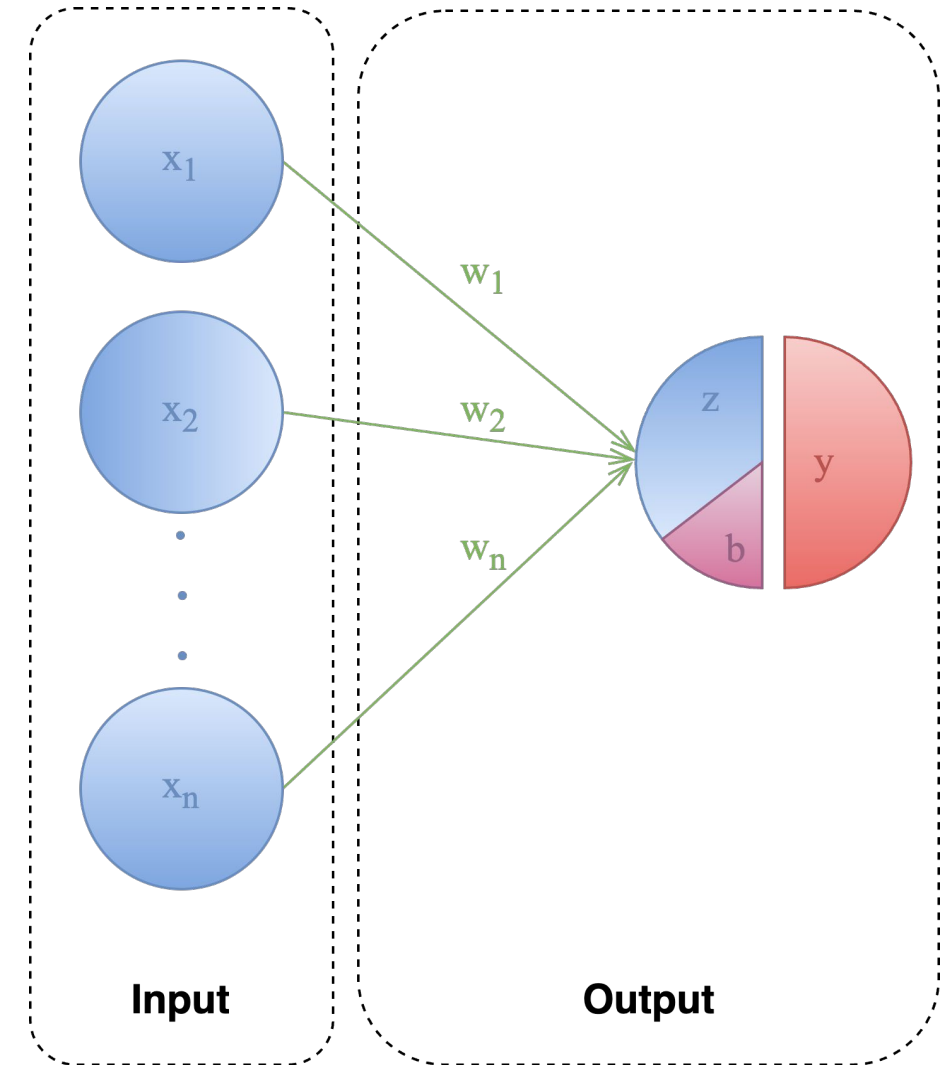
$$\vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix}, \quad w_i \in \mathbb{R}$$

- Net input

$$z = w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

$$= \vec{w}^T \cdot \vec{x}$$

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

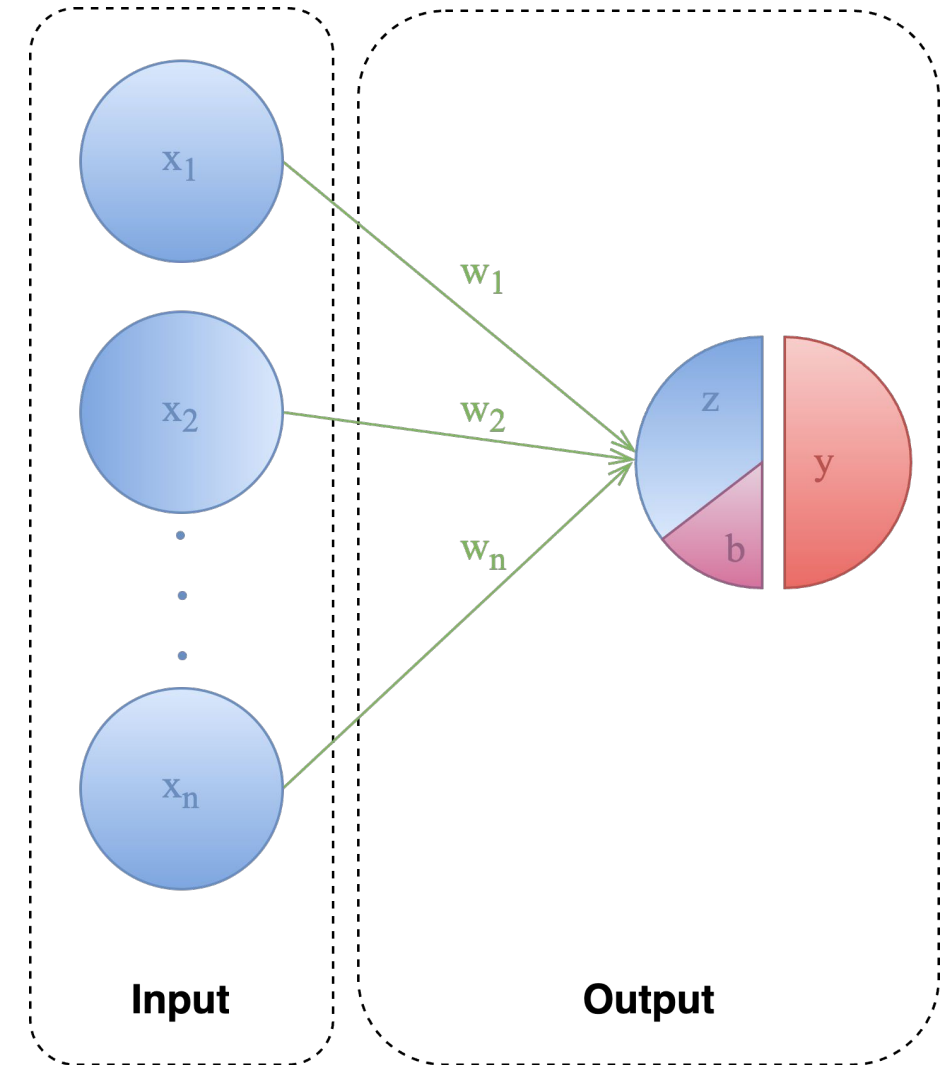


2 — Perceptron model

Forward propagation

- **Activation and output**
 - In the second and final step, we calculate the activation of the output neuron, which also corresponds to the output of the perceptron model
 - An activation function is applied to the network input:

$$y = g(z + b)$$



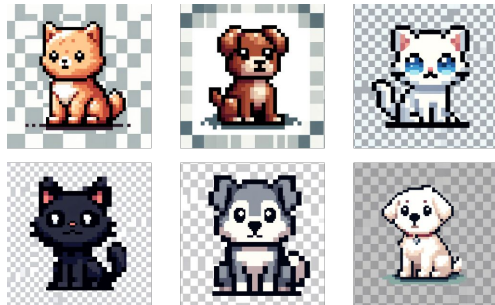
2 — Perceptron model

Learning process

In the context of the perceptron model, we understand learning as the **gradual adaptation of the weights to the desired target function** with the help of training data.

Labeled Data

Training data is, for example, a series of data with a label.



Data labeled as
cat and non-cat.

The training data can therefore be written as pairs of feature vectors and labels:

$$(\vec{x}^k, y^k) \quad k \in \{1, \dots, N\}$$

With a neural network, we must distinguish between the calculated output of the current network and the correct output of a training example.

2 — Perceptron model

Learning process

Learning Step

Learning means that we calculate the output for each training example and then adjust the weights. This is called a learning step.

We can therefore adjust the weight vector for the pairs of feature vectors and labels in each learning step by adding a change of all weights to the current value:

$$w_i := w_i + \Delta w_i$$

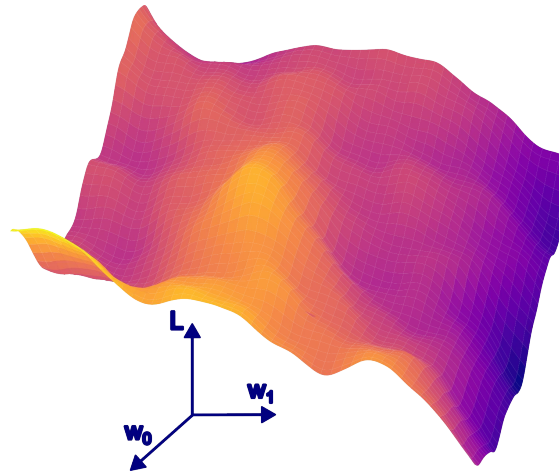
2 — Perceptron model

Loss function

- In order to derive a suitable learning rule, we must first define the term loss function:

$$L(w) = \frac{1}{2N} \sum_{k=1}^N [y^k - \bar{y}^k]^2$$

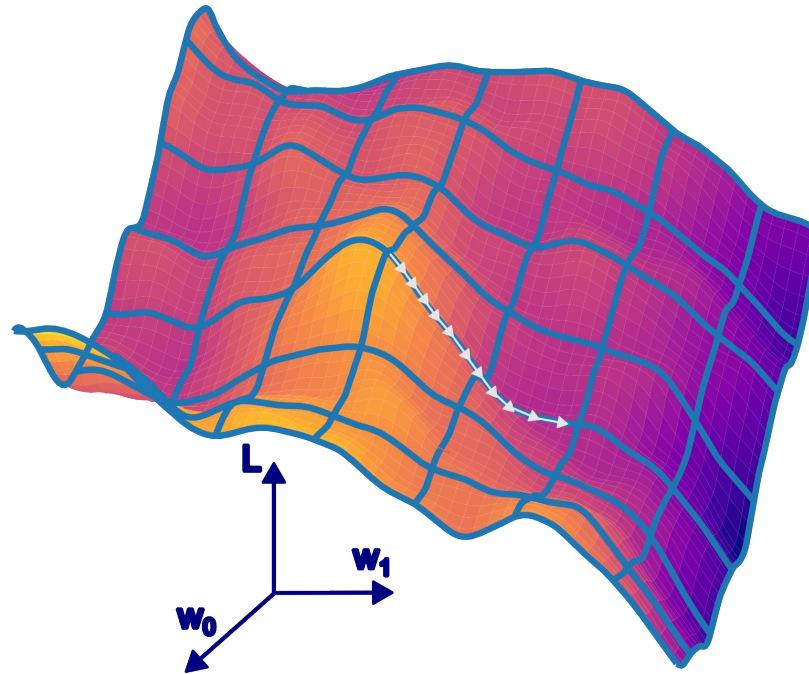
- It is a multidimensional function of the weights or the weight vector



2 — Perceptron model

Gradient descent

- **Learning Rule: Minimum of the Loss Landscape**
 - We can now use the loss function to derive a learning rule by setting ourselves the goal of minimizing the value of the loss function. This means that we look for valleys in the parameter landscape (we did the same in the last lecture for linear regression).



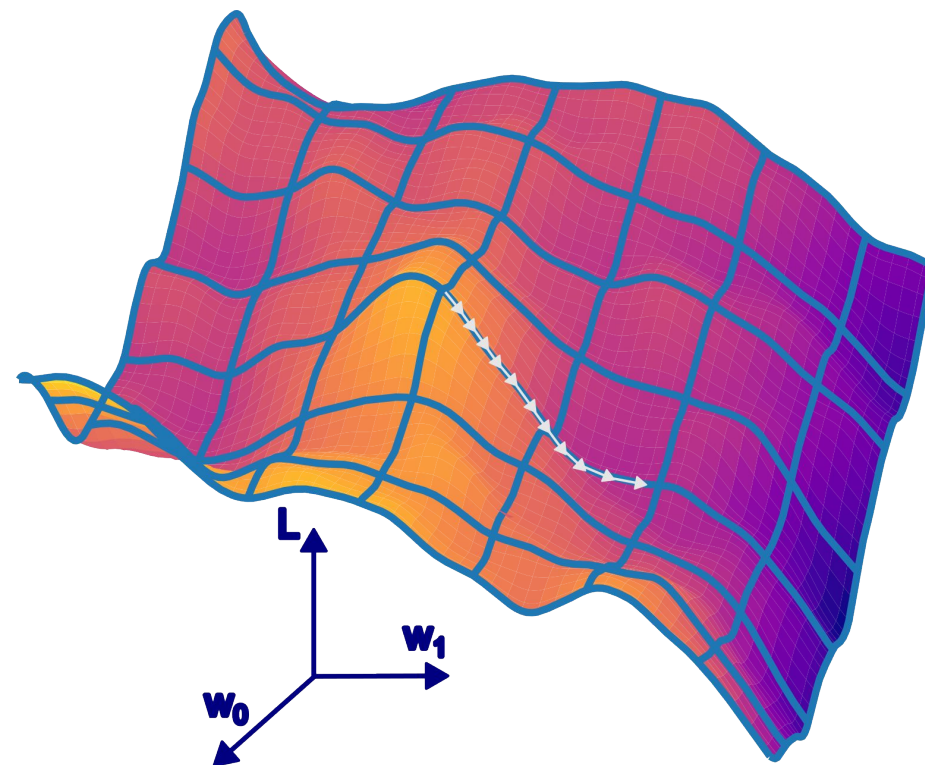
2 — Perceptron model

Optimizing the weights: Weight update

Use the gradient of the loss landscape, which informs us about the largest increase in the error landscape. We therefore take the negative gradient and thus obtain our learning rule:

$$\begin{aligned}\Delta \vec{w} &= -\alpha \nabla L(w) \\ &= -\alpha \begin{pmatrix} \frac{\partial L(w)}{\partial w_1} \\ \vdots \\ \frac{\partial L(w)}{\partial w_n} \end{pmatrix}\end{aligned}$$

As can be seen in the figure, updating the weights in the direction of the negative gradient causes us to run in the direction of the minimum of the loss landscape.



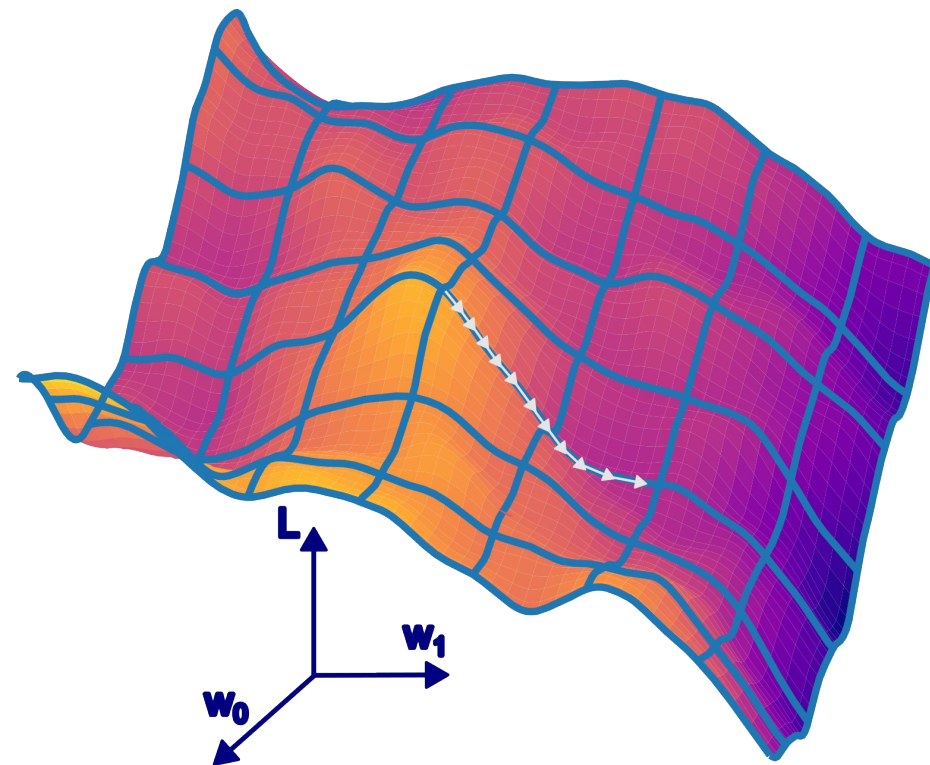
2 — Perceptron model

Weight update demonstration

Derive the weight update for the perceptron model with a linear activation function:

$$\begin{aligned}\frac{\partial L}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2N} \sum_{k=1}^N (y^k - \vec{w}^T \vec{x}^k)^2 \\ &= \frac{1}{2N} \sum_{k=1}^N \frac{\partial}{\partial w_i} (y^k - \vec{w}^T \vec{x}^k)^2 \\ &= \frac{1}{2N} \sum_{k=1}^N 2 (y^k - \vec{w}^T \vec{x}^k) \frac{\partial}{\partial w_i} (y^k - \vec{w}^T \vec{x}^k) \\ &= \frac{1}{N} \sum_{k=1}^N (y^k - \vec{w}^T \vec{x}^k) \frac{\partial}{\partial w_i} \left[-(w_1, \dots, w_i, \dots, w_n) \begin{pmatrix} x_1^k \\ \vdots \\ x_i^k \\ \vdots \\ x_n^k \end{pmatrix} \right] \\ &= -\frac{1}{N} \sum_{k=1}^N (y^k - \vec{w}^T \vec{x}^k) \frac{\partial}{\partial w_i} [(w_1 x_1^k + \dots + w_i x_i^k + \dots + w_n x_n^k)] \\ &= -\frac{1}{N} \sum_{k=1}^N (y^k - \vec{w}^T \vec{x}^k) x_i^k .\end{aligned}$$

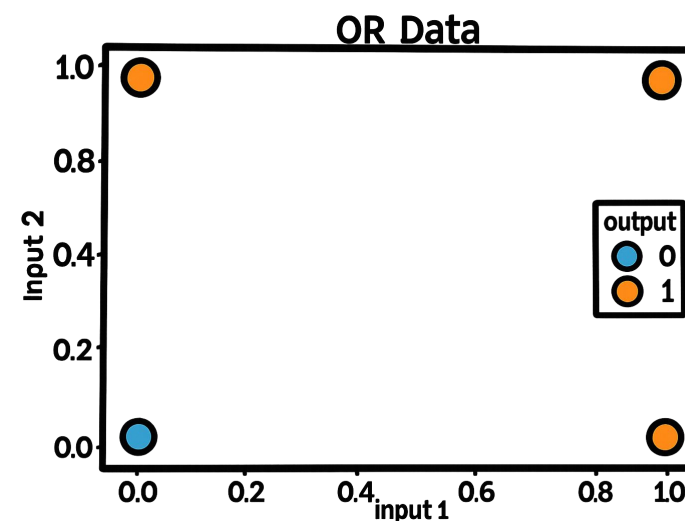
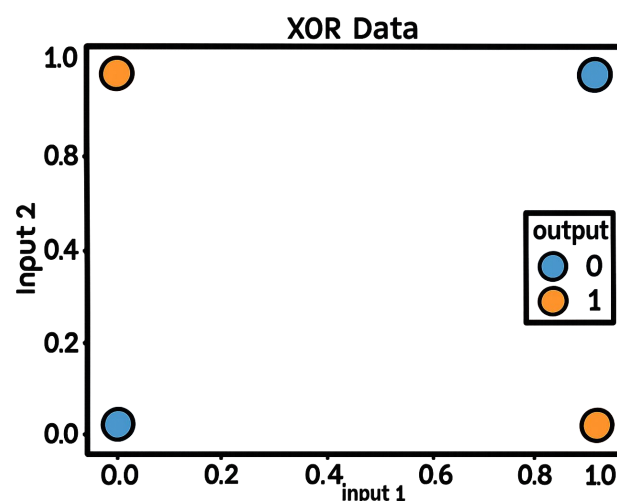
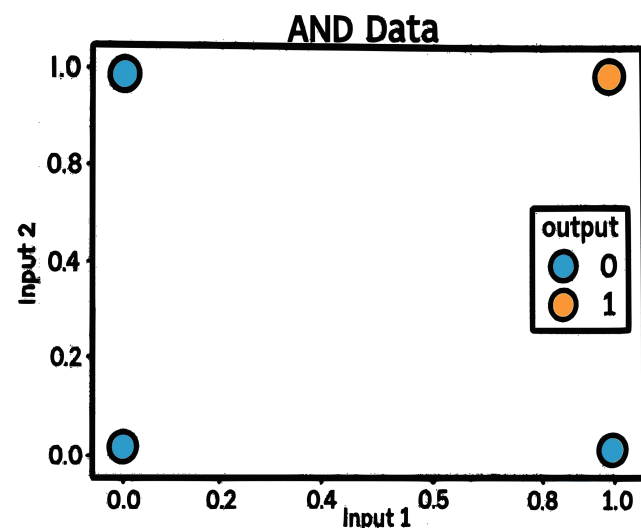
Let us derive
this on the
board!



2 — Perceptron model

XOR problem — Where the perceptron model fails

- Inability of the perceptron model to solve non-linearly separable problems
- A perceptron can only draw a straight line (or hyperplane in higher dimensions) to separate two classes
- If the data points from different classes cannot be separated by a straight line, the perceptron fails to classify them correctly
- An illustration of this is the well-known XOR problem
- **Need to go beyond the perceptron model to solve nonlinear problems!**

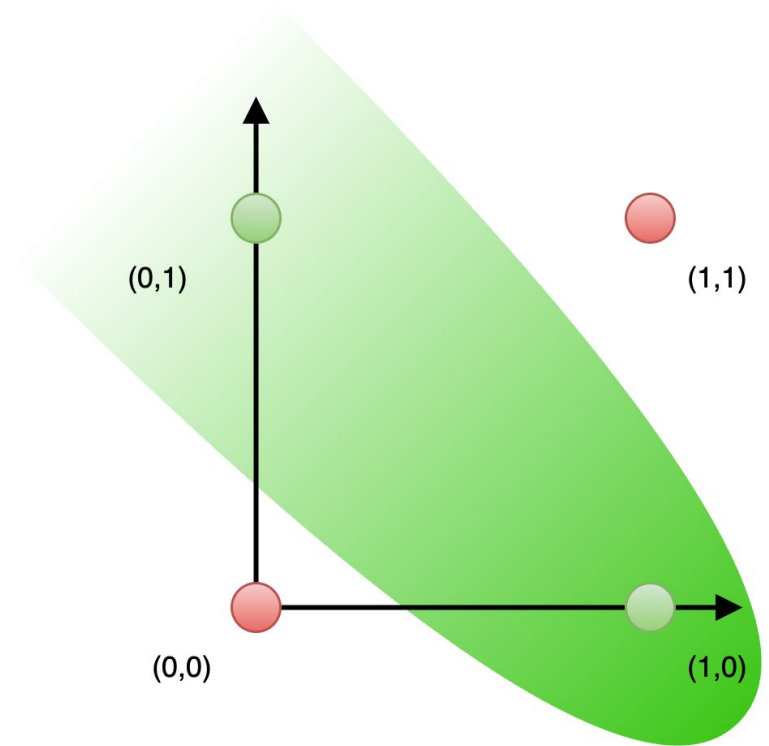


3 — From perceptrons to multi-layer neural networks

3 — From perceptrons to multi-layer neural networks

Anatomy of multi-layer neural networks

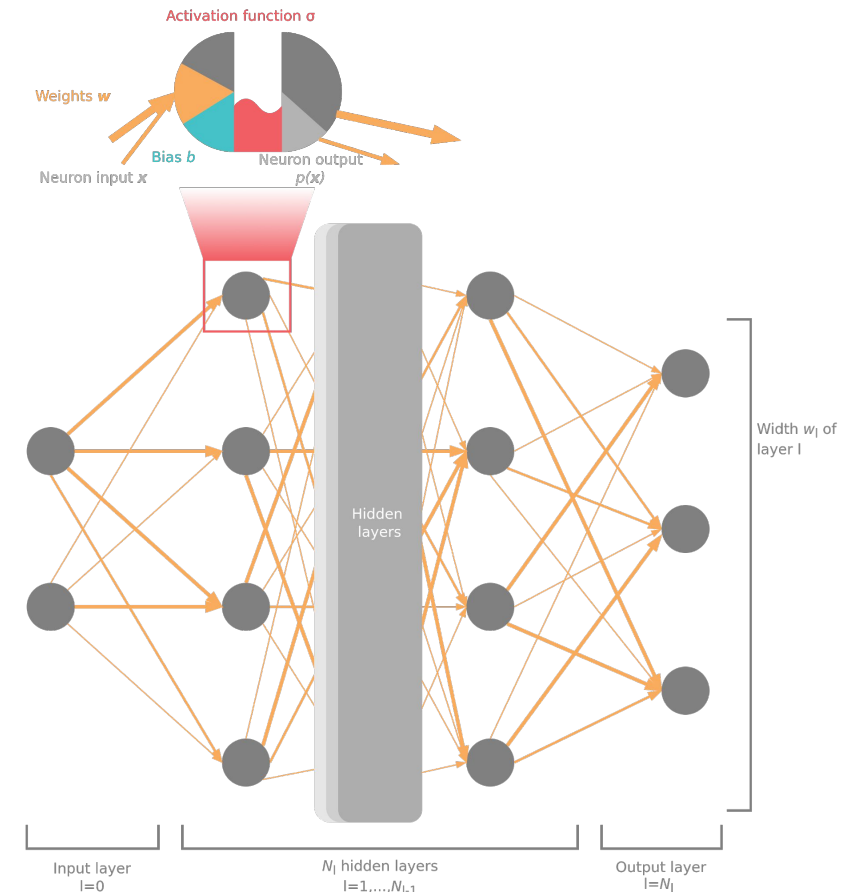
- Single perceptron → linear decision only
- Insert one or more hidden layers to compose features
- Each hidden neuron learns an intermediate pattern; network stacks simple pieces into complex functions
- Even one hidden layer and non-linear activation dramatically increases representational power



3 — From perceptrons to multi-layer neural networks

Structure of neural networks

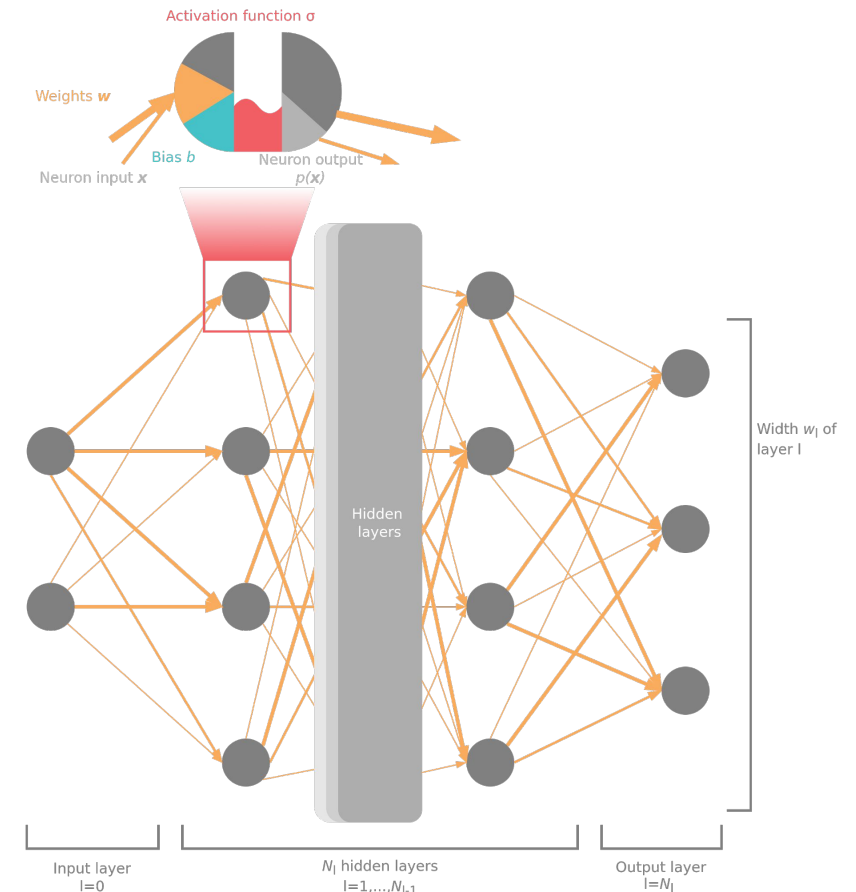
- Terminology
 - Multi-layer perceptron, multi-layer neural network, feedforward neural network
- Composed of simple base units (neurons, perceptrons)
 - The base units perform mathematical operations
- Data flows in one direction
 - Input layer — hidden layers — output layer
- Neurons apply activation functions to introduce non-linearity
$$p(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$$
- Output is obtained from nested evaluation of hidden layers
$$f(\mathbf{x}) = f^{(\dots)}(\dots f^{(2)}(f^{(1)}(\mathbf{x}))\dots)$$



3 — From perceptrons to multi-layer neural networks

Structure of neural networks

- Model parameters such as weights and biases are updated during training using (stochastic) gradient descent.
- What about other parameters, such as layer width, the number of layers, activation function, and learning rate?
- These are **hyper-parameters** and must adjusted during training separately.
- **Hyperparameter optimization** is the process of identifying the ideal model architecture and training procedures.

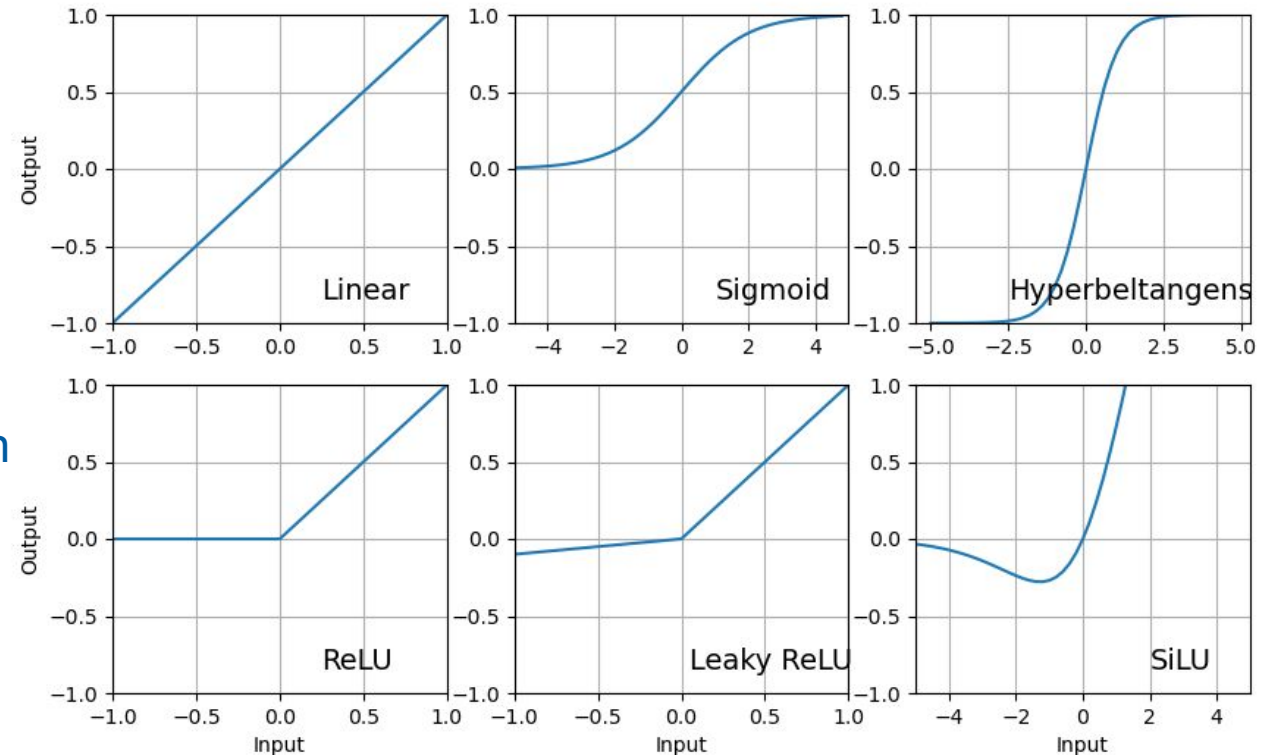


4 — Activation functions

4 — Activation functions

Examples

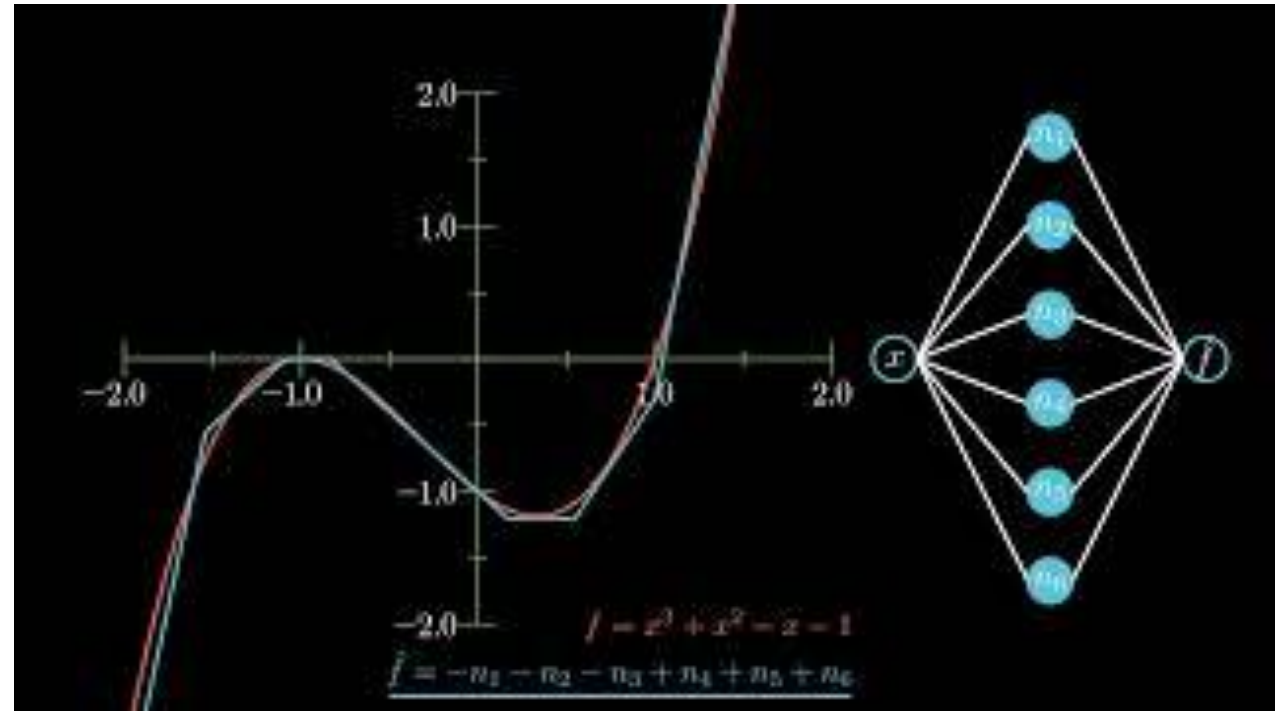
- Stacking purely linear layers \Rightarrow still linear
- Activation functions introduce non-linearity to neural networks so they can approximate complex curves
- Must be differentiable (at least piecewise) to enable backpropagation gradients
- Activation functions is an active area of research
- Rectified linear units (ReLU) is default choice
- Many other types are available
- Often trial and error to find best choice



5 — Universal approximation theorem

5 — Universal approximation theorem

- Informal statement of the theorem
 - ***“A feed-forward neural network with one hidden layer, finite number of neurons, and a non-linear activation can approximate any continuous function on a closed, bounded domain, to any desired accuracy.”***
- First proved (independently) by Cybenko '89 (sigmoid) & Hornik et al. '89/'91 (broader activations)
- Hidden layer \Rightarrow basis-function expansion learned from data
- Guarantees existence of a solution, but does not mean that it is easy to find such a solution



6 — Forward propagation

6 — Forward propagation

Demonstration of the forward pass

- Consider this simple neural network (Input dimension: 3, output dimension: 1, hidden layers: 2)

2. Forward pass in vector / matrix form

- Hidden pre-activation (linear step)

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \in \mathbb{R}^2$$

- Hidden activation (non-linearity)

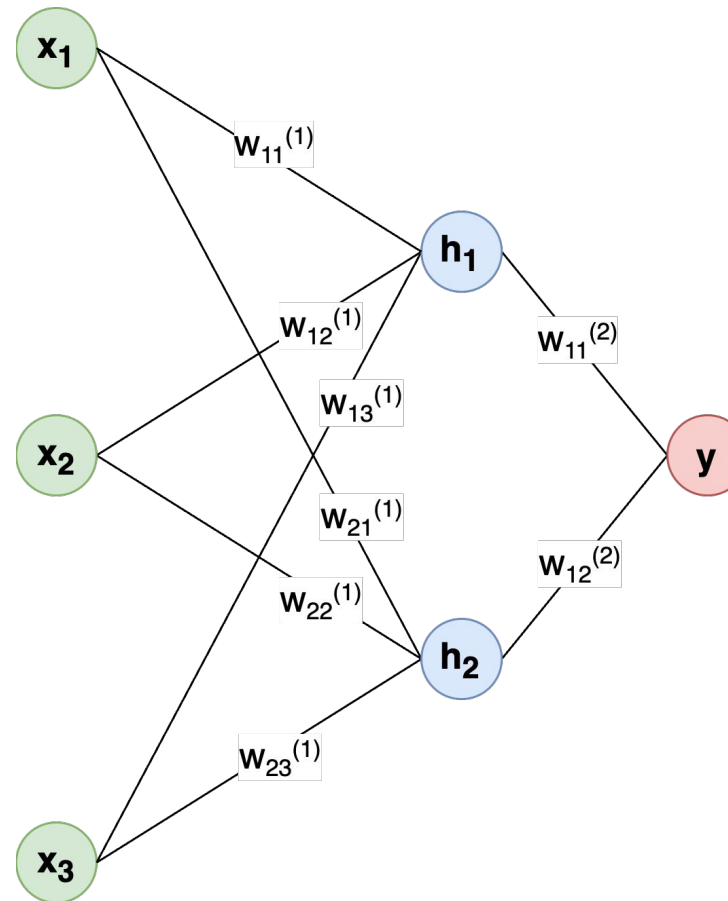
$$\mathbf{h} = \varphi(\mathbf{z}^{(1)}) \in \mathbb{R}^2$$

- Output pre-activation

$$z^{(2)} = \mathbf{W}^{(2)} \mathbf{h} + b^{(2)} \in \mathbb{R}$$

- Final output

$$\hat{y} = \psi(z^{(2)}) \in \mathbb{R}.$$



$$\hat{y} = \psi(\mathbf{W}^{(2)} \varphi(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)}) + b^{(2)})$$

6 — Forward propagation

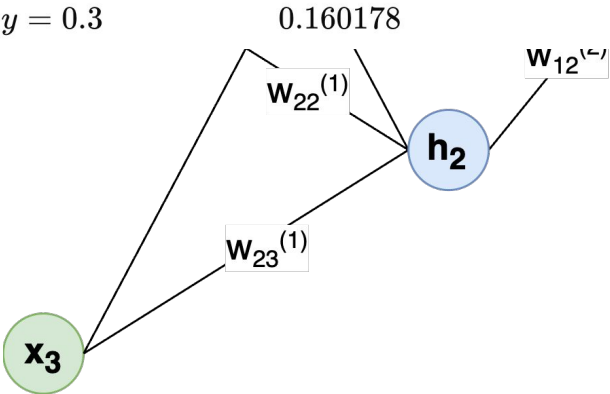
Demonstration of the forward pass

- Consider this simple neural network (Input dimension: 3, output dimension: 1, hidden layers: 2)

Symbol	Value	Comment
Input \mathbf{x}	$[0.6, -1.1, 0.2]^T$	3 features
Target y	0.3	scalar regression goal
Hidden weights $\mathbf{W}^{(1)}$	$\begin{bmatrix} 0.7 & -0.4 & 0.1 \\ 0.05 & 0.9 & -0.3 \end{bmatrix}$	2×3
Hidden bias $\mathbf{b}^{(1)}$	$[-0.2, 0.1]^T$	length 2
Activation φ	ReLU, $\varphi(z) = \max(0, z)$	element-wise
Output weights $\mathbf{W}^{(2)}$	$[1.2, -0.8]$	1×2
Output bias $b^{(2)}$	0.05	scalar
Output act. ψ	identity (regression)	$\psi(z) = z$
Loss L	mean-squared error, $\frac{1}{2}(\hat{y} - y)^2$	

Step	Formula	Numeric result (shape)
Hidden pre-act	$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$	$\begin{bmatrix} 0.68 \\ -0.92 \end{bmatrix}$
Hidden act.	$\mathbf{h} = \varphi(\mathbf{z}^{(1)})$	$\begin{bmatrix} 0.68 \\ 0 \end{bmatrix}$ (ReLU zeros-out second entry)
Output pre-act	$z^{(2)} = \mathbf{W}^{(2)}\mathbf{h} + b^{(2)}$	0.866
Output	$\hat{y} = \psi(z^{(2)})$	0.866
Loss	$L = \frac{1}{2}(\hat{y} - y)^2$ with $y = 0.3$	

Let us derive
this on the
board!

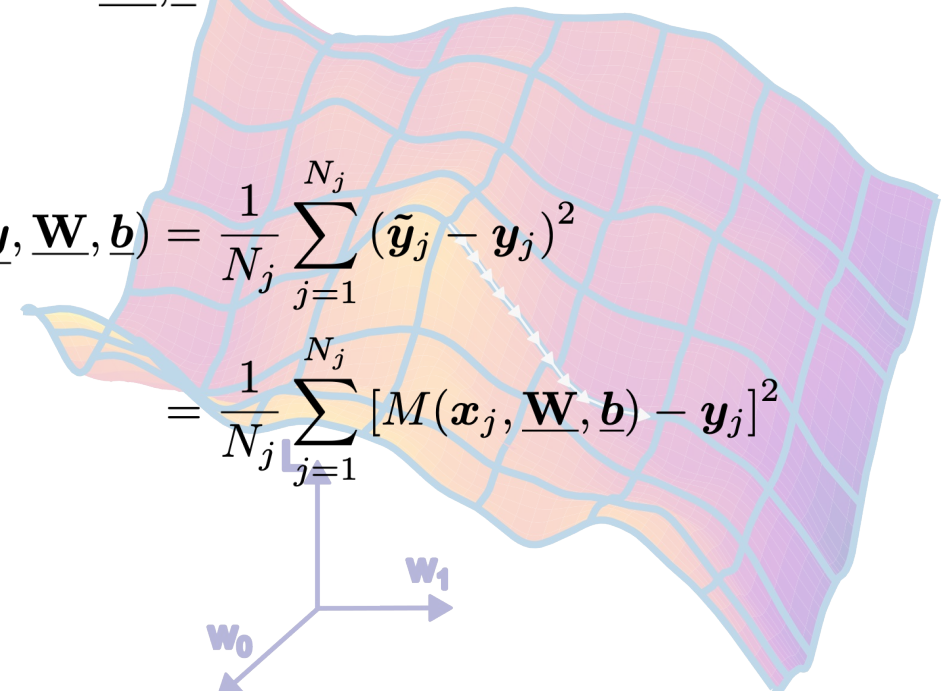



7 — Training neural networks

7 — Training neural networks

Stochastic gradient descent

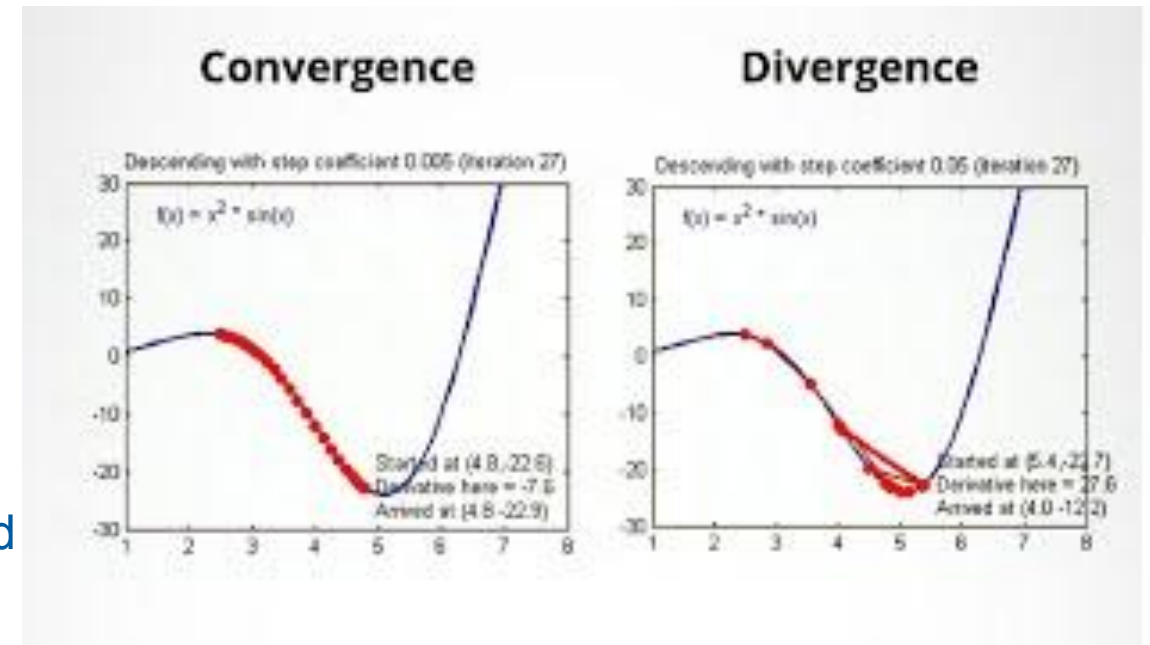
- Evaluate accuracy of predictions based on loss
- Objective
 - Minimize loss w.r.t. weights
- Learning-rate controls step size
 - too large → diverge
 - too small → crawl
- Batch vs. mini-batch
 - Batch → full dataset each step (accurate, slow)
 - Mini-batch (32–1024 samples) → trade-off speed & noise (standard)
- Gradient update of weights (learnable parameters) in terms of backpropagation

$$\min_{\underline{\mathbf{W}}, \underline{\mathbf{b}}} L(\underline{\mathbf{x}}, \underline{\mathbf{y}}, \underline{\mathbf{W}}, \underline{\mathbf{b}})$$

$$L(\underline{\mathbf{x}}, \underline{\mathbf{y}}, \underline{\mathbf{W}}, \underline{\mathbf{b}}) = \frac{1}{N_j} \sum_{j=1}^{N_j} (\tilde{\mathbf{y}}_j - \mathbf{y}_j)^2$$
$$= \frac{1}{N_j} \sum_{j=1}^{N_j} [M(\mathbf{x}_j, \underline{\mathbf{W}}, \underline{\mathbf{b}}) - \mathbf{y}_j]^2$$

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

7 — Training neural networks

Stochastic gradient descent

- Evaluate accuracy of predictions based on loss
- Objective
 - Minimize loss w.r.t. weights
- Learning-rate controls step size
 - too large → diverge
 - too small → crawl
- Batch vs. mini-batch
 - Batch → full dataset each step (accurate, slow)
 - Mini-batch (32–1024 samples) → trade-off speed & noise (standard)
- Gradient update of weights (learnable parameters) in terms of backpropagation



7 — Training neural networks

Gradient update

- Recall forward propagation

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \quad (\in \mathbb{R}^2)$$

$$\mathbf{h} = \varphi(\mathbf{z}^{(1)}) \quad (\in \mathbb{R}^2)$$

$$z^{(2)} = \mathbf{W}^{(2)}\mathbf{h} + b^{(2)} \quad (\in \mathbb{R})$$

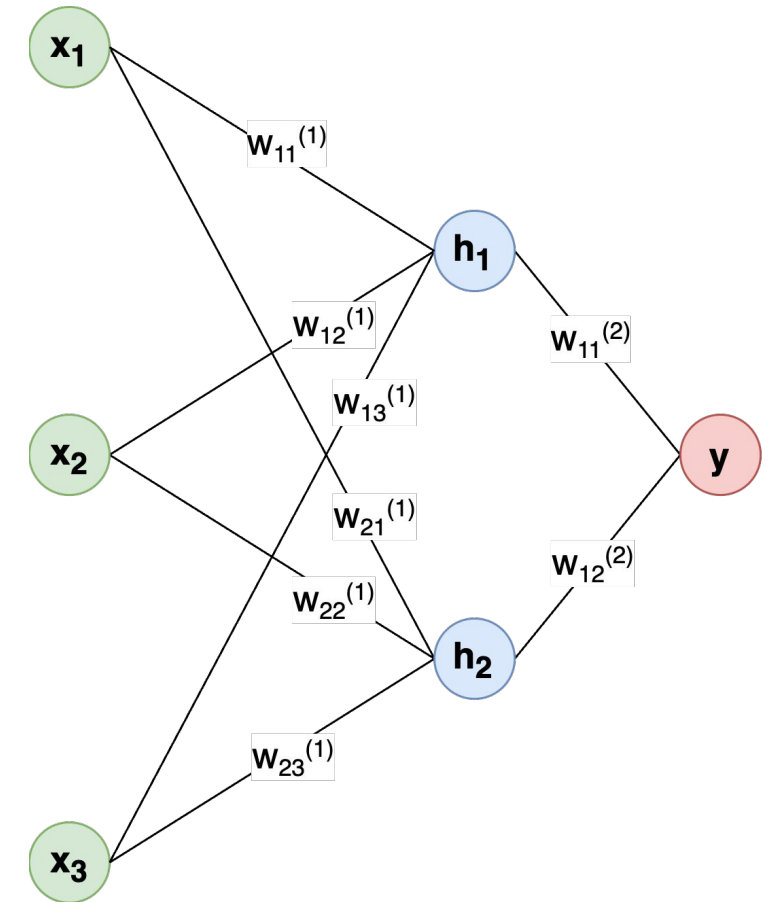
$$\hat{y} = \psi(z^{(2)}) \quad (\in \mathbb{R})$$

- Recall gradient update formula

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$$

Layer 2 (output)

parameter	gradient	update equation
$\mathbf{W}^{(2)} \in \mathbb{R}^{1 \times 2}$	$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \delta^{(2)} \mathbf{h}^\top$	$\mathbf{W}^{(2)} \leftarrow \mathbf{W}^{(2)} - \eta \delta^{(2)} \mathbf{h}^\top$
$b^{(2)} \in \mathbb{R}$	$\frac{\partial L}{\partial b^{(2)}} = \delta^{(2)}$	$b^{(2)} \leftarrow b^{(2)} - \eta \delta^{(2)}$



7 — Training neural networks

Backpropagation

- Let's first consider the **output layer**:

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \delta^{(2)} \mathbf{h}^\top$$

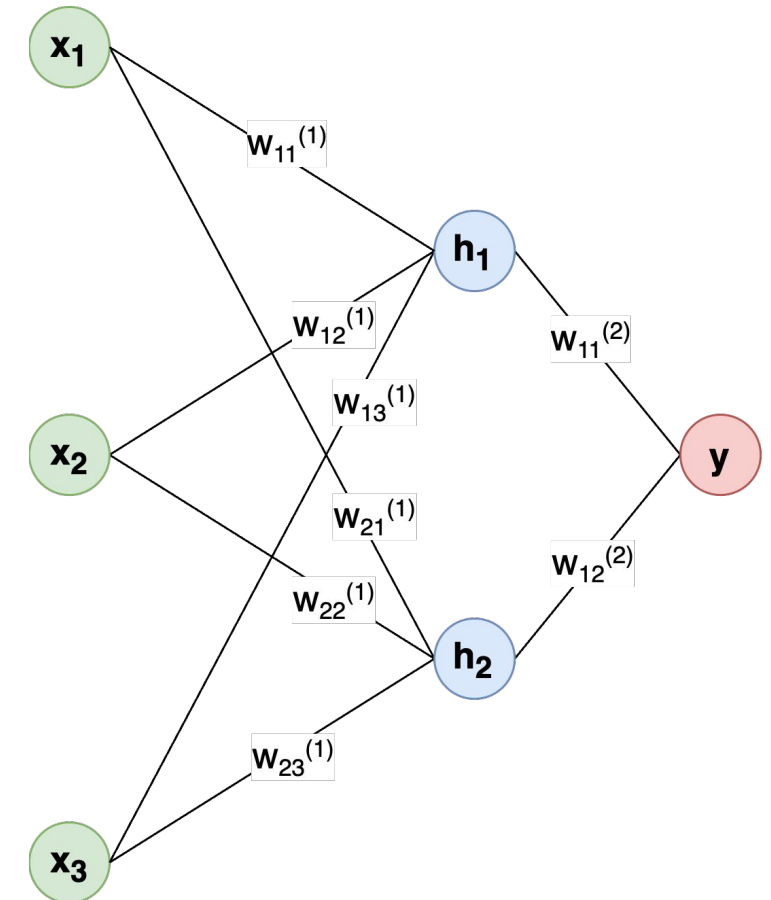
$$\delta^{(2)} = \frac{\partial L}{\partial z^{(2)}} = \frac{\partial L}{\partial \hat{y}} \psi'(z^{(2)}) = L'_{\hat{y}} \psi'(z^{(2)})$$

- The **hidden layer** yields:

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \boldsymbol{\delta}^{(1)} \mathbf{x}^\top$$

$$\boldsymbol{\delta}^{(1)} = \frac{\partial L}{\partial \mathbf{z}^{(1)}} = (\mathbf{W}^{(2)})^\top \delta^{(2)} \odot \varphi'(\mathbf{z}^{(1)})$$

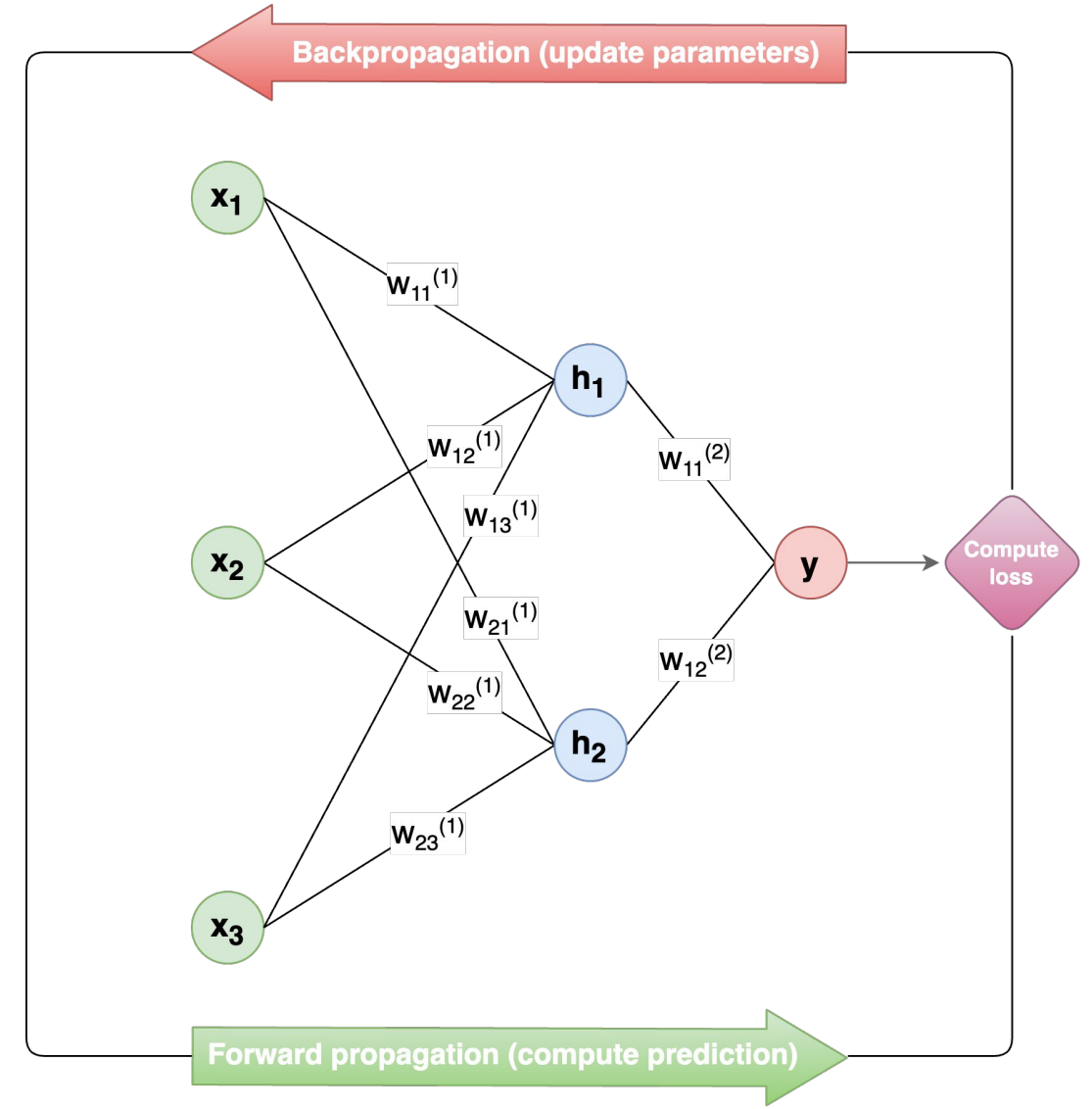
- Error is propagated backward from output to earlier layers



7 — Training neural networks

Workflow overview

- Training a neural network consists of
 - **Forward propagation** → **Compute loss**
 - **Backpropagation** → **Gradients**
 - **Gradient descent** → **Weight update**
- Forward pass builds a computational graph (stores activations)
- Backward pass re-uses those activations to compute gradients once per weight
- Dramatically faster than finite-difference
- Implemented automatic differentiation in libraries (such as PyTorch autograd)



8 — Limitations of neural networks and regularization

8 — Limitations of neural networks and regularization

Common challenges for feedforward neural networks

- **Data**
 - Often need vast amounts of data for training
- **Computational costs**
 - Training and running neural networks requires substantial computational resources (many GPUs needed)
- **Overfitting and generalization**
 - Neural networks can memorize the training data instead of learning the underlying patterns (→ **Regularization** can help!)
- **Hyperparameter optimization**
 - Finding the right hyperparameters can be time-consuming and challenging
- **Interpretability and explainability**
 - Considered as “black boxes”

8 — Limitations of neural networks and regularization

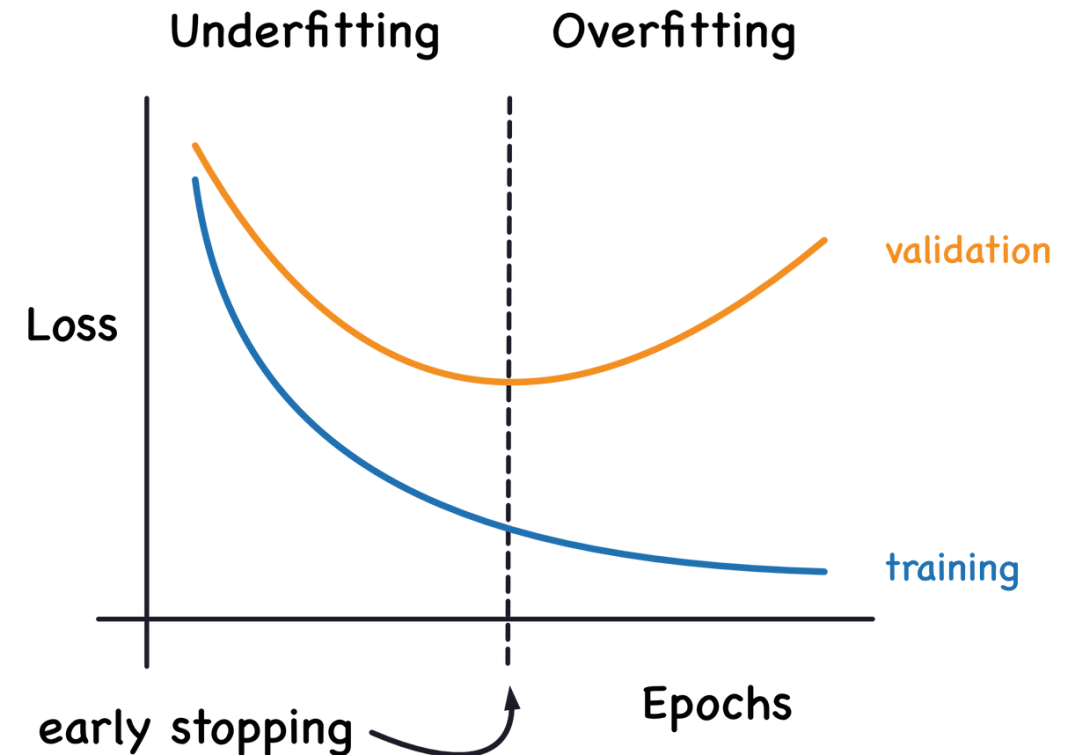
Learning curve

- **Underfitting (bias)**

- Error due to overly simplistic assumptions in the learning algorithm.
- Model cannot capture the underlying pattern. It's not complex enough for the task.
- Example: A shallow neural network trying to model a highly nonlinear function.

- **Overfitting (variance)**

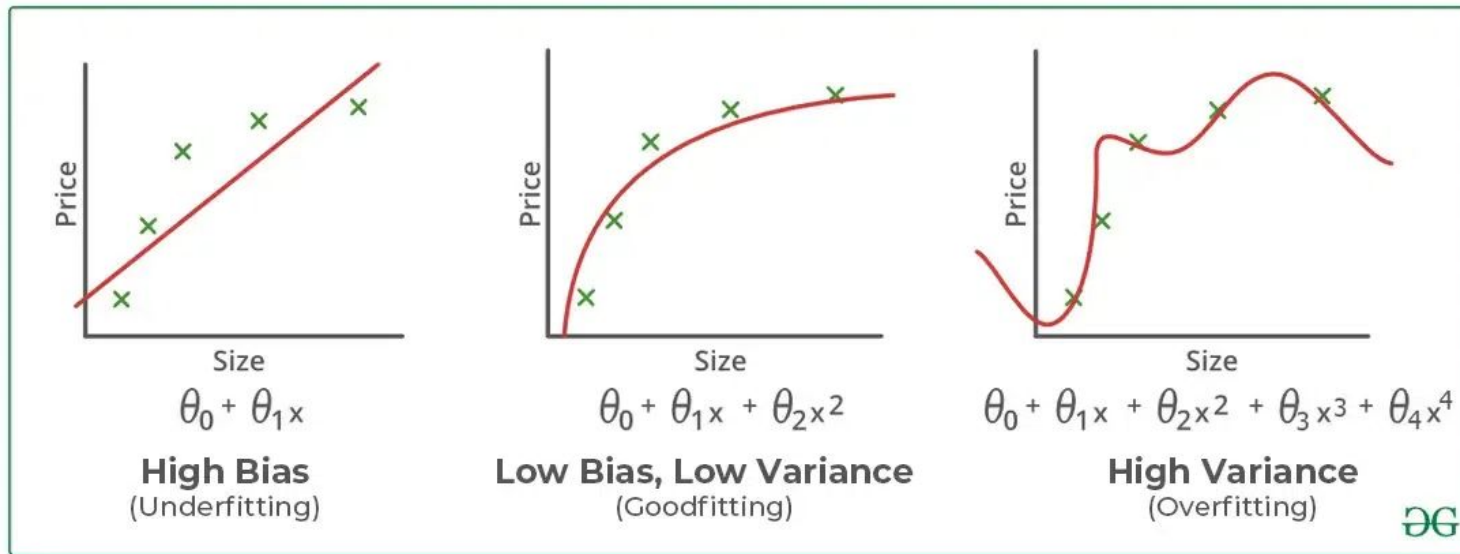
- Error due to the model's sensitivity to small fluctuations in the training set. A model with high variance learns the noise in the training data and fails to generalize well.
 - i. Low training error.
 - ii. High validation error, and there's a large gap between training and validation loss.
- Model is too complex relative to the amount of training data or regularization.



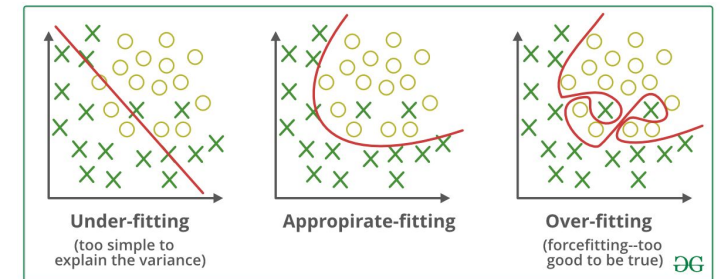
8 — Limitations of neural networks and regularization

Model capacity spectrum

Regression



Classification



8 — Limitations of neural networks and regularization

Regularization methods

- **L1 regularization** (LASSO: Least Absolute Shrinkage and Selection Operator)
 - Lasso encourages sparsity by bringing some coefficients to exactly zero.
 - Can set some weights exactly zero \Rightarrow implicit feature selection.

$$L = \frac{1}{N} \sum_i^N (\hat{y}_i - y_i)^2 + \lambda \sum_i^N |\theta_i|$$

- **L2 regularization**
 - Shrinks all weights smoothly toward zero (also known as **weight decay**).
 - Encourages the model to have smaller and more balanced weights.

$$L = \frac{1}{N} \sum_i^N (\hat{y}_i - y_i)^2 + \lambda \sum_i^N \theta_i^2$$

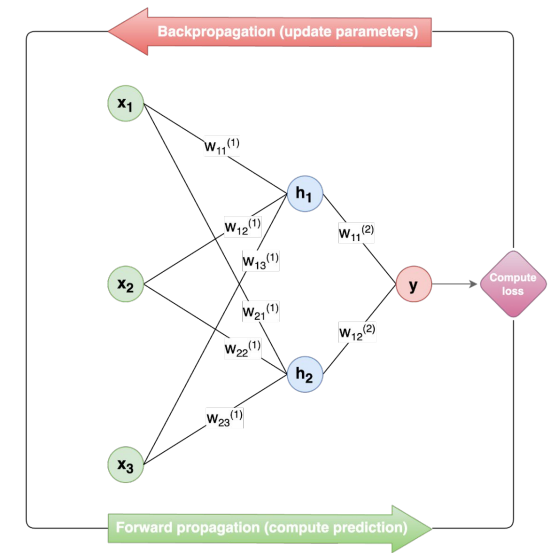
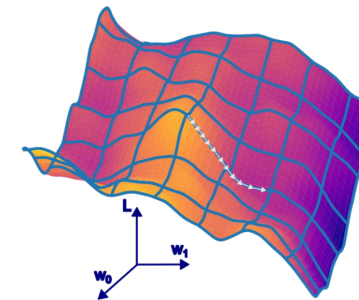
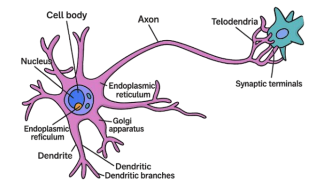
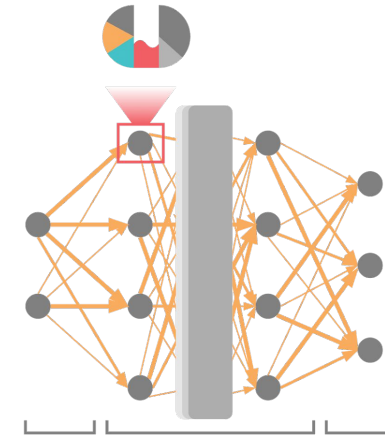
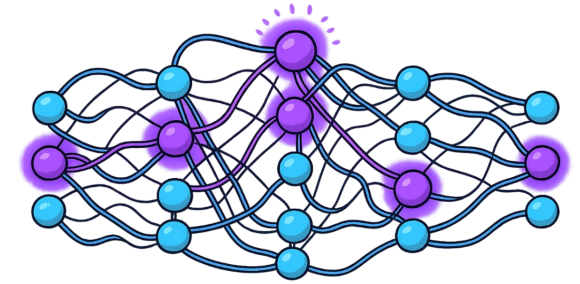
- **Elastic net regularization**

$$L = \frac{1}{N} \sum_i^N (\hat{y}_i - y_i)^2 + \lambda \sum_i^N |\theta_i| + \lambda \sum_i^N \theta_i^2$$

9 — Key takeaways

9 — Key takeaways

- Overview of neural networks
- Perceptron model
- Multi-layer perceptrons / feedforward neural networks
- Activation functions
- Universal approximation theorem
- Training neural networks
 - Forward and backpropagation
 - Stochastic gradient descent
- Regularization



Selected resources for further study

- Some popular online resources about the basics of machine learning and neural networks
 - **Deep Learning Book**
 - <https://www.deeplearningbook.org>
 - **Neural Networks and Deep Learning**
 - <http://neuralnetworksanddeeplearning.com/>
 - **Dive into Deep Learning**
 - <https://d2l.ai/>
- Domain-specific resources
 - **Deep Learning for Molecules and Materials**
 - <https://dmol.pub>
 - **Machine Learning for Materials**
 - <https://aronwalsh.github.io/MLforMaterials>

