TU Delft

Delft University of Technology

Document Version
Final published version

Licence
Dutch Copyright Act (Article 25fa)

Citation (APA)
Quiñones, J. (2026). Fundamentals of Artificial Intelligence and Machine Learning. In L. Castillo, G. Carbajal, V. Castaño, A. Ruiz-Columbie, U. Ciri, J. Quiñones, C. E. Rudamas Flores, & M. Velez-Rubio (Eds.), *Energy and Coastal Security, Resilience, and Workforce Development: Fundamentals* (Vol. 1, pp. 81-106). Springer Nature. https://doi.org/10.1007/978-3-032-08309-8_8

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Chapter 8
Fundamentals of Artificial Intelligence and Machine Learning

Jhon Quiñones

8.1 Preliminaries of Artificial Intelligence

AI refers to computational methodologies designed to replicate tasks typically associated with human intelligence, such as reasoning, learning, perception, planning, and decision-making [1]. Historically, AI research began with symbolic or rule-based approaches, employing explicit logical representations of human knowledge. Despite notable early successes, including expert systems and automated reasoning, these symbolic methods struggled with handling uncertainty, complexity, and scalability, prompting researchers to explore alternative frameworks capable of overcoming these limitations [2].

In response, a significant transition to data-driven approaches emerged, leading to ML, a subset of AI that emphasizes algorithms capable of autonomously learning from data without relying on explicitly programmed rules [3]. ML methodologies identify and generalize patterns in empirical data, leveraging statistical techniques to infer insights and make accurate predictions about previously unseen scenarios [4]. ML methods can be broadly classified into supervised learning, where models learn from labeled datasets; unsupervised learning, which discovers structures in unlabeled data; and reinforcement learning, where algorithms learn optimal decision strategies through interactions with an environment guided by reward-based feedback [5]. In addition to extending ML's capabilities, deep learning, a specialized form that employs multilayered artificial neural networks, has improved AI's ability to process and interpret complex, high-dimensional data.

Given their powerful predictive and optimization capabilities, ML techniques have become integral to renewable energy systems, significantly improving resource forecasting, operational efficiency, and infrastructure planning and investment [6]. To effectively exploit the full potential of AI in renewable energy applications, it is

J. Quiñones (✉)
Delft University of Technology, Delft, The Netherlands
e-mail: jquinonescorte@tudelft.nl

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026

L. Castillo et al., *Energy and Coastal Security, Resilience, and Workforce Development, Volume 1*, https://doi.org/10.1007/978-3-032-08309-8_8

crucial to understand the fundamental mathematical concepts, learning paradigms, and algorithmic principles of ML. Thus, this chapter introduces the essential foundations of machine learning, providing the conceptual tools necessary to leverage AI within renewable energy developed in later chapters.

8.2 Foundations of Machine Learning

At its core, the ML methodology begins with the formal representation of data. Consider a dataset X, formally defined as a set containing n samples, each described by m distinct features:

$$X = \{x_1, x_2, \ldots, x_n\}, \quad \text{where} \quad x_i \in \mathbb{R}^m, \quad i = 1, \ldots, n \tag{8.1}$$

Thus, each vector x_i represents an individual data instance in the m-dimensional feature space. The quality of the selected features critically influences the performance of the model. Therefore, feature extraction, feature selection, and dimensionality reduction procedures are required.

ML algorithms are formalized through the concept of learning a functional relationship $f : \mathbb{R}^m \to \mathbb{R}^d$, which maps the input vectors x to the output vectors y. The model learning process involves identifying parameters θ that minimize a predefined cost function $J(\theta)$, quantifying discrepancies between the predicted outputs $\hat{y} = f(x; \theta)$ and the true outputs y. In general form, this optimization problem is expressed as:

$$\underset{\theta}{\operatorname{argmin}}\ J(\theta) = \underset{\theta}{\operatorname{argmin}}\ \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(x_i; \theta)) \tag{8.2}$$

Optimization of the cost function is commonly performed using gradient-based methods, such as gradient descent. The iterative parameter update rule for gradient descent is given by:

$$\theta \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta} \tag{8.3}$$

where α is the learning rate, a hyperparameter controlling the step size of updates in each iteration [4]. Selecting a suitable learning rate is critical because an excessively large learning rate can hinder convergence, whereas a small learning rate may prolong the optimization process or trap the algorithm in local minima [7].

Moreover, robust model validation strategies, such as k-fold cross-validation or bootstrapping, ensure the generalizability of ML models [8]. Formally, in k-fold cross-validation, the dataset X is partitioned into k disjoint subsets $X_1, X_2, \ldots, X_k$. For each iteration j, the model is trained on the combined data $X \setminus X_j$ and validated on subset X_j. This method rigorously assesses the predictive accuracy of the model and its ability to generalize to unseen data [9].

ML methodologies can be classified according to their learning paradigms, reflecting how algorithms process data and derive knowledge. In general, there are three primary paradigms: supervised learning, unsupervised learning, and reinforcement learning [8]. Each paradigm addresses different types of problems and employs unique mathematical frameworks to formalize the learning process, thus facilitating customized solutions across diverse application areas, including renewable energy systems.

8.2.1 *Supervised Learning*

Supervised learning comprises algorithms designed to learn predictive mappings from input variables to known outputs using labeled datasets. Formally, given a dataset of n samples, each associated with an input vector and corresponding output label, the dataset is expressed as:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \quad \mathbf{x}_i \in \mathbb{R}^m, \quad y_i \in \mathcal{Y} \tag{8.4}$$

Here, $\mathbf{x}_i$ denotes the input feature vector with dimensionality m, while y_i represents the corresponding output or label, drawn from a domain $\mathcal{Y}$, which determines whether the task is one of *regression* (continuous targets) or *classification* (categorical targets).

Common supervised learning techniques include linear and logistic regression, support vector machines (SVMs), decision trees and their ensemble extensions such as random forests (RF) and gradient-boosted trees, naïve Bayes classifiers, and multilayer perceptrons (feed-forward neural networks). Figure 8.1 shows graphical representation of classical supervised learning techniques. When these models tackle continuous-valued outputs, the nomenclature typically switches to a regression form. For example, support vector regression (SVR), decision tree regressor, or random forest regressor, while for categorical outputs, their standard classification names are retained [10].

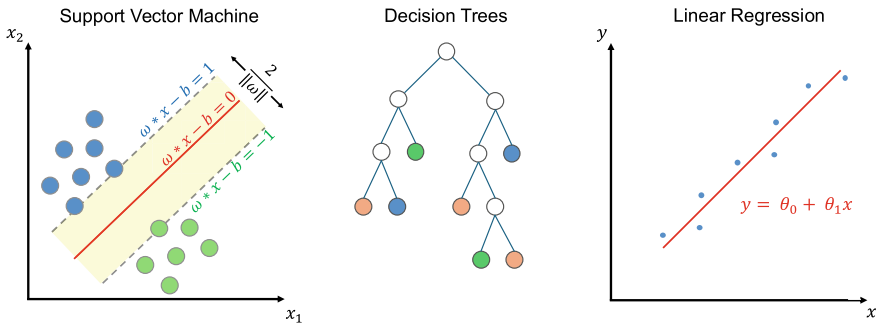


Fig. 8.1 Supervised machine learning models including support vector machines (SVM), decision trees, linear regression

8.2.1.1 Regression Tasks

Regression tasks involve predicting continuous numeric outcomes from input variables. The output domain is typically the set of real numbers, formally denoted as:

$$y_i \in \mathcal{Y} \subseteq \mathbb{R}, \quad \forall i \tag{8.5}$$

The objective is to learn a continuous function $f(\mathbf{x}; \theta)$ parameterized by θ that maps inputs to outputs, minimizing discrepancies between predicted values $\hat{y}_i$ and actual targets y_i. A common loss function for regression tasks is the Mean Squared Error (MSE), mathematically defined as:

$$\mathcal{L}_{\text{MSE}}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{8.6}$$

Alternatively, Mean Absolute Error (MAE), which is less sensitive to outliers, is defined as:

$$\mathcal{L}_{\text{MAE}}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{8.7}$$

Common supervised regression algorithms include Linear Regression, Ridge and Lasso Regression, random forests, SVM, and Neural Networks [10]. Linear regression, for instance, assumes a linear relationship between inputs and outputs expressed by:

$$f(\mathbf{x}; \theta) = \mathbf{w}^{\mathrm{T}} \mathbf{x} + b \tag{8.8}$$

where $\mathbf{w}$ represents a vector of weights, and b is the bias term. These parameters are typically estimated by minimizing the selected loss function using optimization algorithms such as gradient descent.

Example: Photovoltaic power forecasting

Utility-scale solar farms introduce high-frequency variability into power systems: a cloud can drop PV output by tens of megawatts within minutes. System operators therefore request hour ahead forecasts to schedule fast acting reserves and avoid imbalance penalties. In the following, we develop a step by step, reproducible Ridge regression baseline that predicts the real power P of a 50 MW PV plant 60 min in advance. Each phase is defined so the reader can replicate the workflow in Python, MATLAB, or even Excel.

Step 1: **Problem formulation**. Index 15 min intervals by t and denote the measured AC power by P_t [kW]. We wish to approximate

$$f^* : \mathbf{x}_t \longmapsto P_{t+4},$$

where the four-step offset corresponds to a one-hour horizon.

Step 2: **Data acquisition and cleaning**.

(a) *Acquire* two years of one-minute SCADA records; down-sample to 15 min means. Retrieve matching numerical weather prediction (NWP) fields: global horizontal irradiance (GHI), direct normal irradiance (DNI), air temperature (T^{air}), cloud-cover fraction (CC), and wind speed (v^{wind}).
(b) *Clean* by dropping nighttime rows ($\text{GHI} \approx 0$), clipping spikes beyond three median absolute deviations, and linearly interpolating gaps shorter than three consecutive points.

Step 3: **Feature engineering**. Each timestamp is mapped to

$$\mathbf{x}_t = \big[\underbrace{\text{GHI}_t, \text{DNI}_t, T_t^{\text{air}}, \text{CC}_t, v_t^{\text{wind}}}_{\text{weather}}, \underbrace{\sin\omega_t, \cos\omega_t}_{\text{diurnal}}, \underbrace{P_t}_{\text{persistence}}\big]^{\mathsf{T}}, \tag{8.9}$$

where $\omega_t = 2\pi\,\big(\text{hour}(t)/24\big)$ provides a continuous encoding of the hour-of-day.

Step 4: **Temporal train-test split**. The earliest 70% of samples form the training set $\mathcal{D}_{\text{tr}}$; the most recent 30% constitute the test set $\mathcal{D}_{\text{te}}$. Shuffling is *not* permitted so as to avoid information leakage across time.

Step 5: **Model fitting (Ridge regression)**. Adopting the linear form (8.10), we solve

$$\hat{\theta} \;=\; \arg\min_{\theta}\; \frac{1}{n_{\text{tr}}} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{tr}}} \big(y - f(\mathbf{x};\theta)\big)^2 \;+\; \lambda\|\mathbf{w}\|_2^2, \tag{8.10}$$

where the regularization strength $\lambda > 0$ is selected via time-series cross-validation. In closed form:

$$\hat{\mathbf{w}} = (X^{\mathsf{T}}X + \lambda I)^{-1}X^{\mathsf{T}}\mathbf{y}, \qquad \hat{b} = \bar{y} - \hat{\mathbf{w}}^{\mathsf{T}}\bar{\mathbf{x}}.$$

Step 6: **Evaluation on unseen data**. For each $(\mathbf{x}, y) \in \mathcal{D}_{\text{te}}$ compute $\hat{y} = f(\mathbf{x};\hat{\theta})$ and report

$$\text{RMSE} = \sqrt{\frac{1}{n_{\text{te}}} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{te}}} \big(y - \hat{y}\big)^2}, \tag{8.11}$$

$$\text{nRMSE} = \text{RMSE}\,/\,P_{\max}, \tag{8.12}$$

with $P_{\max} = 50\,\text{MW}$. Always benchmark against the *persistence* forecast $\hat{P}_{t+4}^{(\text{per})} = P_t$; a viable model must outperform this naive baseline.

Step 7: **Interpretation of coefficients**. A positive $\hat{w}_{\text{GHI}}$ confirms that stronger irradiance increases the output, while a negative $\hat{w}_{T^{\text{air}}}$ reflects losses in thermal efficiency at elevated module temperatures. Such transparency builds operator confidence.

Step 8: **Deployment and drift monitoring**. A quarter-hourly pipeline now *fetches* the latest NWP slice and live P_t, *assembles* $\mathbf{x}_t$ via (8.9), *computes* $\hat{P}_{t+4}$, and *publishes* the forecast to the energy management API. A rolling dashboard tracks nRMSE; when error drifts beyond a predefined guard band, the model is automatically retrained on the expanding data set.

Remark Steps 1–4 (data engineering) and 6–8 (evaluation, interpretation, and deployment) remain untouched if a practitioner switches Ridge regression for SVR, gradient-boosted trees, or an artificial neural network at Step 5. Consequently, the framework scales seamlessly from quick baseline to sophisticated production forecaster, all while maintaining a clear audit trail that operators and regulators can trust.

8.2.1.2 Classification Tasks

In classification tasks, the goal is to assign input instances into discrete classes or categories. Formally, the output labels are drawn from a finite set:

$$y_i \in \mathcal{Y} = \{C_1, C_2, \ldots, C_k\}, \quad \forall i \tag{8.13}$$

where k denotes the number of distinct classes. The predictive function in classification, typically denoted as $f(\mathbf{x}; \theta)$, outputs a probability distribution over possible classes. For binary classification ($k = 2$), logistic regression is frequently used, expressed mathematically as:

$$f(\mathbf{x}; \theta) = P(y = 1|\mathbf{x}; \theta) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b), \quad \text{where} \quad \sigma(z) = \frac{1}{1 + e^{-z}} \tag{8.14}$$

Classification tasks commonly employ the cross-entropy loss function, which measures the discrepancy between predicted probabilities and actual class labels. For binary classification, cross-entropy loss is given by:

$$\mathcal{L}_{\mathrm{CE}}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{8.15}$$

In multiclass classification scenarios, the output predictions are represented as a probability distribution over k classes using softmax activation:

$$f_j(\mathbf{x}; \theta) = P(y = C_j|\mathbf{x}; \theta) = \frac{e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{x}+b_j}}{\sum_{l=1}^{k} e^{\mathbf{w}_l^{\mathrm{T}}\mathbf{x}+b_l}}, \quad j = 1, 2, \ldots, k \tag{8.16}$$

The associated multiclass cross-entropy loss function is defined by:

$$\mathcal{L}_{\mathrm{CE}}(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}\mathbb{I}(y_i = C_j)\log(\hat{y}_{ij}) \tag{8.17}$$

where $\mathbb{I}(\cdot)$ is the indicator function, equal to 1 if the condition is true and 0 otherwise.

Prominent supervised classification methods include Logistic Regression, Support Vector Machines (SVM), Decision Trees, Random Forests, and Neural Networks. SVM, for example, classifies data by finding an optimal hyperplane that maximizes the margin between classes, defined formally by the optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1, \quad \forall i \tag{8.18}$$

In renewable energy applications, supervised learning, both regression and classification, is widely applied for resource forecasting, anomaly detection, and predictive maintenance [11, 12].

Example: incipient gearbox-fault detection in wind turbines

Motivation. A single gearbox failure in a modern 3 MW turbine can cost hundreds of thousands of dollars in parts, labor, and lost production. Catching *incipient faults* early therefore turns unscheduled outages into planned maintenance. The workflow below builds a SVM classifier that turns raw SCADA and vibration streams into a real-time alarm, following the same eight-step structure used for the regression case.

Step 1: **Problem formulation**. At every 10 s interval t we summarize sensor signals into a feature vector $\mathbf{x}_t \in \mathbb{R}^d$. Maintenance logs retrospectively assign a binary label

$$y_t = \begin{cases} 1, & \text{incipient gearbox fault within the next 24 h,} \\ 0, & \text{normal operation.} \end{cases}$$

We seek a decision function $g^* : \mathbf{x}_t \longmapsto y_t$ that maximises early-warning capability.

Step 2: **Data acquisition** & **cleaning**.

(a) *Acquire* rotor speed ω_t, generator torque T_t, oil temperature T_t^{oil}, vibration RMS, spectral side-band energy, and ambient conditions from the turbine SCADA historian.
(b) *Label* by joining SCADA rows to the computerized maintenance management system (CMMS); mark the 24h preceding each confirmed gearbox ticket as "incipient fault."
(c) *Clean* by removing gross outliers, resampling to a uniform 10 s cadence, and forward-filling gaps shorter than 30 s.

Step 3: **Feature engineering**. Combine physics-guided and statistical cues:

$$\mathbf{x}_t = \left[\omega_t,\ T_t,\ T_t^{\mathrm{oil}},\ \mathrm{RMS}_t,\ s_t,\ \sin\psi_t,\ \cos\psi_t\right]^{\mathsf{T}}, \tag{8.19}$$

where $\psi_t = 2\pi\ \mathrm{month}(t)/12$ encodes seasonality. Because fault samples are scarce ($< 2\%$), assign higher class weights to $y = 1$ or employ SMOTE-style oversampling.

Step 4: **Temporal train-test split**. To prevent leakage across physically distinct machines, group by turbine ID: select 70% of turbines for $\mathcal{D}_{\text{tr}}$ and keep the remaining 30% as the unseen test fleet $\mathcal{D}_{\text{te}}$.

Step 5: **Model fitting (SVM with RBF kernel)**. Optimize

$$\begin{aligned} &\min_{\mathbf{w},b,\boldsymbol{\xi}} \tfrac{1}{2}\|\mathbf{w}\|^2 + C\sum_t \xi_t \\ &\text{s.t. } y_t\left(\mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}(\mathbf{x}_t) + b\right) \geq 1 - \xi_t,\ \xi_t \geq 0, \end{aligned} \tag{8.20}$$

where $\boldsymbol{\phi}(\cdot)$ is the radial-basis-function (RBF) feature map $k(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2\right)$. Choose (C, γ) via grouped time-series cross-validation; weight C more heavily for the minority class (fault).

Step 6: **Evaluation on unseen turbines**. Apply the fitted classifier to $\mathcal{D}_{\text{te}}$ to obtain fault probabilities p_t. Select a threshold τ in concert with operations engineers (e.g., to guarantee minimum recall) and compute

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{8.21}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{8.22}$$

$$F_1 = 2\,\frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{8.23}$$

where TP, FP, and FN follow standard confusion matrix definitions. Plotting the receiver operating characteristic (ROC) curve further illustrates the trade-off between false alarms and missed faults.

Step 7: **Deployment and drift monitoring**. An edge computer in each turbine now *streams* live features every 10 s, *scores* them through $g(\mathbf{x}_t)$, and *raises* an alarm when $g(\mathbf{x}_t) \geq \tau$ in N successive windows. A scheduled retraining routine adds new technician-verified labels and re-optimizes (8.20) whenever rolling F_1 degrades beyond a preset margin.

Remark By swapping the SVM in **Step 5** for a gradient-boosted tree, a convolutional network in raw vibration spectra, or even a hybrid physics/data model, practitioners can enhance accuracy without modifying the step-by-step process, ensuring that predictive maintenance remains robust and explainable.

8.2.2 Unsupervised Learning

Unlike supervised learning, unsupervised learning algorithms analyze unlabeled datasets, with the aim of identifying inherent structures, patterns, or groupings without explicit guidance. This paradigm is particularly valuable in domains where

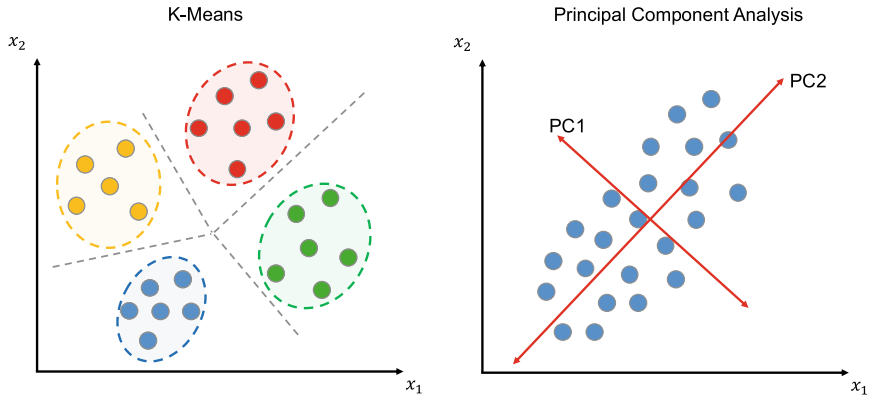


Fig. 8.2 Unsupervised machine learning models including k-means and principal component analysis (PCA)

labeled data is scarce, expensive to obtain, or subjective, as is often the case in energy consumption, sensor networks, and climate data analysis. Figure 8.2, illustrates common unsupervised learning techniques such as clustering algorithms (k-Means), and dimensionality reduction (principal component analysis (PCA)).

Formally, given a dataset without labels:

$$\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{n}, \quad \mathbf{x}_i \in \mathbb{R}^m \tag{8.24}$$

the goal of unsupervised learning is to learn a transformation or representation $g : \mathbb{R}^m \to \mathbb{R}^d$ (with $d \leq m$) or a discrete mapping to clusters or latent variables that captures underlying structure in the data.

8.2.2.1 Clustering Algorithms

Clustering remains one of the most fundamental tasks in unsupervised learning. It involves grouping data points so that points within the same group (or cluster) are more similar to each other than to those in other groups. A classic approach is the *K-Means* algorithm, which partitions data into k clusters by minimizing the within-cluster sum of squared Euclidean distances:

$$J = \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in C_j} \|\mathbf{x}_i - \boldsymbol{\mu}_j\|^2 \tag{8.25}$$

Here, C_j denotes the set of data points assigned to cluster j, and $\boldsymbol{\mu}_j$ is the centroid of that cluster.

Clustering methods, such as DBSCAN (Density-Based Spatial Clustering of Applications with Noise), avoid specifying the number of clusters in advance and

instead rely on local density criteria [13]. Hierarchical clustering builds a tree-like structure of nested clusters using bottom-up (agglomerative) or top-down (divisive) approaches, offering interpretability in applications where multilevel grouping is beneficial.

8.2.2.2 Dimensionality Reduction

Another major class of unsupervised learning is dimensionality reduction, which aims to project high-dimensional data into a lower-dimensional subspace while retaining essential information [14]. This is crucial for visualization, noise reduction, and computational efficiency in large-scale datasets.

A foundational method is *Principal Component Analysis* (PCA), which finds orthogonal directions (principal components) capturing the maximum variance in the data. PCA solves the eigenvalue problem of the sample covariance matrix:

$$\Sigma \mathbf{w} = \lambda \mathbf{w}, \quad \text{where} \quad \Sigma = \frac{1}{n-1}\mathbf{X}\mathbf{X}^{\mathrm{T}} \tag{8.26}$$

The eigenvectors $\mathbf{w}$ correspond to the principal directions, and the eigenvalues λ reflect the amount of variance captured.

Beyond PCA, non-linear dimensionality reduction methods have gained traction, particularly for data that lies on nonlinear manifolds. T-distributed Stochastic Neighbor Embedding (t-SNE) [15] and Uniform Manifold Approximation and Projection (UMAP) [16] are powerful techniques that preserve local and global structures respectively, and are widely used in exploratory data analysis.

In the renewable energy domain, unsupervised learning techniques play an essential role in tasks such as:

- **Consumer segmentation**: Clustering household or industrial consumption profiles to design dynamic pricing or demand response strategies [17].
- **Sensor anomaly detection**: Identifying faults or abnormal behaviors in wind turbines, solar panels, or microgrid systems using reconstruction-based or clustering-based models [18].
- **Data compression and preprocessing**: Extracting low-dimensional, informative features from high-resolution weather or satellite data for forecasting tasks [19].

Example: clustering household load profiles for demand-response solutions

Motivation. Residential demand is notoriously heterogeneous: some homes peak at breakfast, others spike after sunset, while a few barely vary at all. However, tariff designers, aggregators, and distribution system operators need a concise catalog of *typical* behaviours to target demand-response (DR) incentives effectively. Unsupervised learning, specifically a PCA $\rightarrow$ k-means pipeline, offers a principled way to distill thousands of smart meter traces into a handful of representative load shapes, without a single labeled example.

Step 1: **Problem set-up**. For each customer h and day d we resample the smart meter data to 48 half hour points,

$$\mathbf{x}^{(h,d)} = \left[P_{00:30},\ P_{01:00}, \ldots, P_{24:00}\right]^{\mathsf{T}} \in \mathbb{R}^{48},$$

yielding an unlabeled collection $\mathcal{D} = \{\mathbf{x}^{(h,d)}\}_{h,d}$ that now feeds the unsupervised pipeline.

Step 2: **Pre-processing** & **Normalization**.

(a) *Filtering:* drop days with $> 10\,\%$ missing intervals; fill shorter gaps by linear interpolation.
(b) *Scale separation:* divide each profile by its daily mean, producing a shape vector with unit energy. This prevents high-consumption households from dominating similarity.

Step 3: **Dimensionality reduction (PCA)**. Stack the centered vectors into $X \in \mathbb{R}^{N\times 48}$, compute the covariance $\Sigma = \frac{1}{N} X^{\mathsf{T}} X$, and diagonalize $\Sigma = U \Lambda U^{\mathsf{T}}$. Retain the first d principal components such that $\sum_{i=1}^{d} \lambda_i / \sum_{i=1}^{48} \lambda_i \geq 0.95$, and project every profile on to

$$\mathbf{z}^{(h,d)} = U_{[:,1:d]}^{\mathsf{T}} \mathbf{x}^{(h,d)} \ \in \ \mathbb{R}^{d}.$$

Step 4: **Clustering (k-means)**. Run k-means on the PCA scores $\{\mathbf{z}^{(h,d)}\}$ for a sweep $k \in \{2, \ldots, 10\}$. Choose k^* at the elbow of the within cluster sum of squares curve or by maximizing average silhouette width.

Step 5: **Cluster interpretation**. Back-project the centroid of each cluster to the original 48-point space and visualize:

- **Cluster A**: flat work from home demand all day;
- **Cluster B**: double breakfast and dinner peaks (classical 9-to-5);
- **Cluster C**: pronounced late evening spike (EV charging).

Overlaying temperature or calendar metadata can reveal latent drivers (e.g., Cluster C grows on hot summer Fridays).

Step 6: **Operational use**. Assign each new daily profile to the nearest centroid in $\mathbb{R}^{d}$. Operators can now

(a) tailor DR tariffs: push time-of-use rates to Cluster B, EV-specific incentives to Cluster C;
(b) forecast aggregate demand by weighting cluster archetypes with daily membership counts;
(c) monitor drift-if the silhouette score slides, retrain PCA and clusters on the rolling data window.

Remark This unsupervised pipeline extracts an actionable structure from a sea of smart-meter readings without any labeled ground truth. PCA compresses 48-dimensional curves into a handful of orthogonal "load eigenshapes," while k-means groups households by genuine behavioural similarity. Because each step is modular, analysts can swap PCA for auto-encoders, replace k-means with density-based clustering, or append anomaly-detection layers.

8.2.3 Reinforcement Learning

Reinforcement learning (RL) involves sequential decision-making where an agent learns to perform actions within an environment, aiming to maximize cumulative rewards through iterative interactions [20], as shown in Fig. 8.3. Mathematically, RL problems are modeled using Markov Decision Processes (MDP), represented by the tuple (S, A, P, R, γ), where:

- S is a set of possible states,
- A is a set of possible actions,
- $P : S \times A \times S \rightarrow [0, 1]$ denotes transition probabilities between states,
- $R : S \times A \rightarrow \mathbb{R}$ is a reward function that assigns immediate feedback to actions, and
- $\gamma \in [0, 1]$ is the discount factor, balancing immediate and future rewards.

The agent seeks a policy $\pi(s)$ mapping states to actions that maximizes expected discounted cumulative reward:

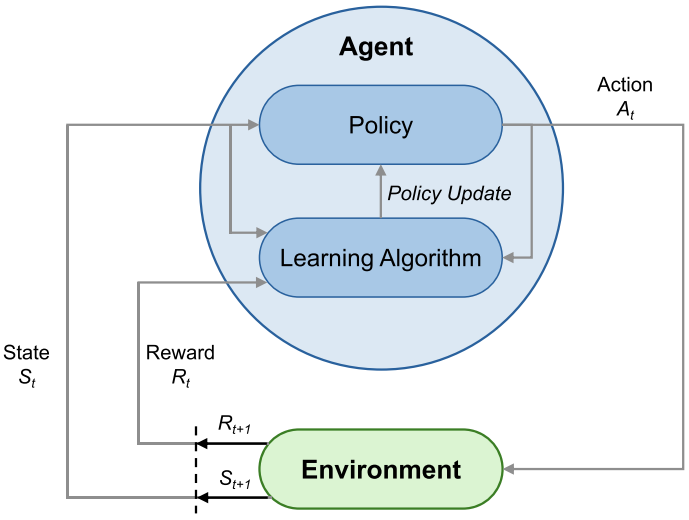


Fig. 8.3 Reinforcement learning framework

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{8.27}$$

Optimal action value functions, known as Q-functions, represent the expected rewards of action state pairs under an optimal policy:

$$Q^*(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a, \pi^*] \tag{8.28}$$

These functions guide policy improvements, typically through iterative algorithms like Q-learning, defined as:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{8.29}$$

where α is the learning rate, controlling the magnitude of updates at each step.

In renewable energy systems, RL is particularly suitable for adaptive real-time control of microgrids, energy storage systems, demand-side management, and active wake control in wind farms [21] optimizing decisions dynamically under uncertainty to achieve operational efficiency and reliability.

Example: Active wake control in wind farms with Q-learning

Wind turbines arranged in large wind farms often suffer from power losses due to turbine wakes regions of slower, turbulent airflow downstream from individual turbines (Fig. 8.4). Active wake control, implemented by adjusting yaw angles of

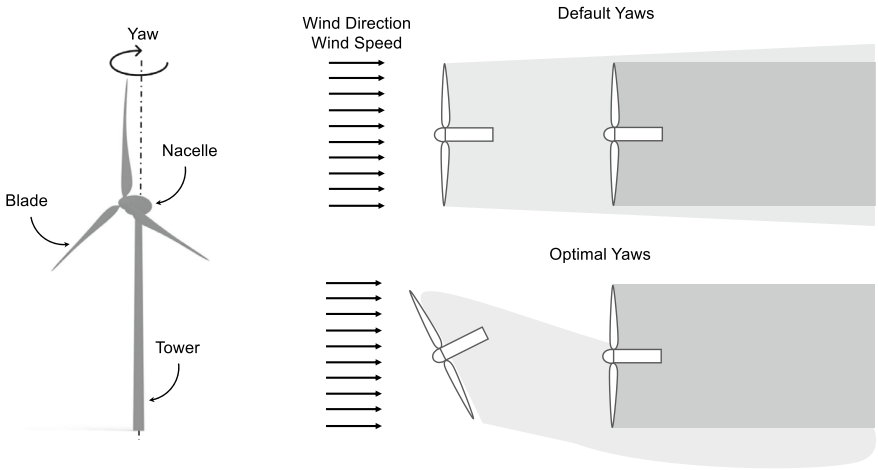


Fig. 8.4 View of two wind turbines without (right top) and with (right bottom) wake control. Darker areas have slower wind

upstream turbines, redirects wakes away from downstream turbines, thus increasing total energy capture. Determining optimal yaw angles is challenging due to dynamic wind conditions and complex wake interactions. Here, we illustrate how a tabular Q-learning agent can autonomously learn optimal yaw adjustments to maximize wind-farm energy production.

Step 1: **Define the environment**. The agent interacts with a simulated wind farm comprising multiple wind turbines. At discrete time steps indexed by t, the environment provides current wind speed and direction, turbine yaw angles, and turbine power outputs.

Step 2: **Construct the state space**. Quantize continuous variables into discrete bins to manage complexity:

$$s_t = (\text{wind direction bin},\ \text{wind speed bin},\ \text{yaw angle bins}) \in \mathcal{S}.$$

For instance, wind direction could be divided into eight sectors, wind speed into five ranges, and yaw angles into discrete increments (e.g., every 5°).

Step 3: **Specify the action set**. Actions correspond to discrete yaw adjustments for each controllable turbine:

$$a_t \in \mathcal{A} = \{\text{yaw left by } \Delta\theta,\ \text{hold},\ \text{yaw right by } \Delta\theta\},$$

typically with increments $\Delta\theta \approx 5°$. Actions exceeding mechanical yaw limits are prohibited.

Step 4: **Design the reward function**. The immediate reward balances total farm power production with yaw adjustment penalties (reflecting mechanical wear):

$$r_t = \sum_{i=1}^{N_{\text{turbines}}} P_{i,t}(a_t) - c_{\text{yaw}} \sum_{i=1}^{N_{\text{turbines}}} |\Delta\text{yaw}_{i,t}|,$$

where $P_{i,t}(a_t)$ is the power output from turbine i at time t, and c_{yaw} penalizes frequent yaw adjustments.

Step 5: **Update rule (tabular Q-learning)**. Initialize the Q-table $Q_0(s, a) = 0$ for all (s, a). After performing action a_t from state s_t, receiving reward r_t, and transitioning to state s_{t+1}, update the Q-values as:

$$Q_{t+1}(s_t, a_t) = (1 - \alpha)\, Q_t(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q_t(s_{t+1}, a') \right],$$

where α is the learning rate and γ the discount factor. Exploration is implemented using an ε-greedy strategy.

Step 6: **Training loop**. Run extensive simulations across various wind scenarios (varying speed and direction). Each simulation involves iteratively applying the Q-update rule. Monitor Q-table convergence through measures such as root mean square Q-value changes between iterations.

Step 7: **Policy extraction**. Once training converges, extract the optimal policy by selecting actions with highest Q-values:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q(s, a).$$

This policy serves as a practical lookup table for operational yaw-angle decision making.

Step 8: **Deployment and online refinement**. Deploy the policy within the wind farm's real-time control system. Continuously gather operational data and periodically refine the Q-table, enabling adaptation to evolving turbine conditions, wind patterns, or maintenance constraints.

Remark Unlike supervised approaches, the Q-learning agent does not require labeled examples of optimal yaw configurations; instead, it autonomously explores and discovers strategies through trial and error. The reward structure explicitly balances energy maximization and mechanical stress minimization, resulting in practically meaningful policies. Furthermore, the transparency of a tabular approach enables direct interpretation and straightforward operational audits, building trust with operators who can examine and validate each learned decision.

8.2.4 Deep Learning

Deep learning (DL) is a branch of machine learning characterized by models composed of multiple layers of interconnected artificial neurons to automatically learn non-linear representations of data (Fig. 8.5). Unlike traditional ML methods, which often depend heavily on carefully hand-engineered features, DL models are capable of directly processing raw input data, such as images, time series, or text, thus significantly reducing the need for domain-specific feature extraction [4, 22]. This capacity for representation learning has revolutionized tasks across numerous fields, from computer vision and natural language processing to energy and climate sciences.

Mathematically, given a labeled dataset comprising n samples, each with an input vector and corresponding label:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \quad \mathbf{x}_i \in \mathbb{R}^m, \quad y_i \in \mathcal{Y}, \tag{8.30}$$

a DL model aims to approximate a mapping $f(\mathbf{x}; \theta)$ that accurately predicts the output labels y_i. This mapping is typically implemented as a multi-layer network:

$$f(\mathbf{x}; \theta) = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}(\mathbf{x}), \tag{8.31}$$

where each layer $f^{(\ell)}$ comprises linear transformations followed by nonlinear activation functions, effectively learning features of increasing complexity at each subsequent layer.

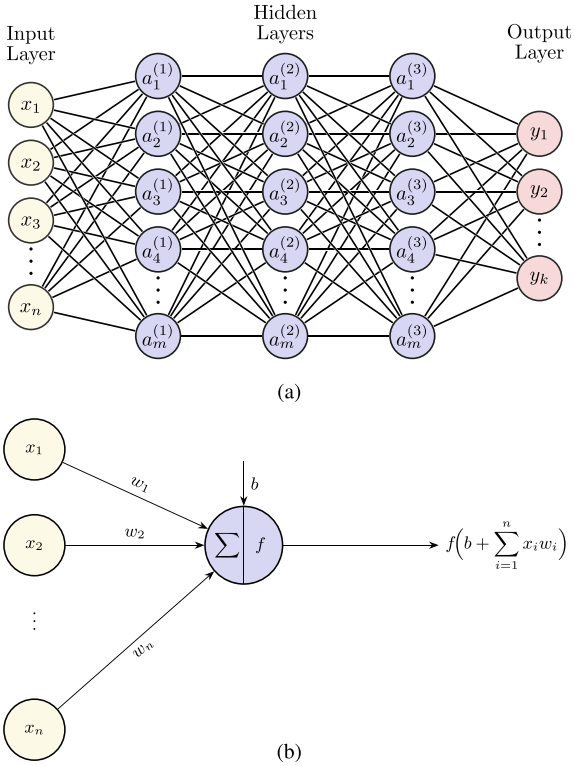


Fig. 8.5 **a** Depicts the network at the architectural level: inputs $x_1, \ldots, x_n$ (yellow) are connected to a stack of fully-connected hidden layers (blue). Each neuron $a_j^{(\ell)}$ in layer ℓ receives signals from *all* neurons in layer $\ell - 1$ and passes its activation forward until the final layer produces the outputs $y_1, \ldots, y_k$ (red). **b** zooms in on a single neuron to show its internal computation. Each input x_i is scaled by a weight w_i, the weighted signals are summed, a bias b is added, and the result is passed through a nonlinear activation $f(\cdot)$. Algebraically the neuron outputs $f\big(b + \sum_{i=1}^{n} x_i w_i\big)$. The dense web of black connectors in (**a**) is therefore a collection of such operations, repeated layer after layer to form a universal function approximator

The parameters of the model, collectively denoted as θ, include weights and biases which are adjusted iteratively through training to minimize a loss function measuring discrepancies between the model predictions and true labels. Optimization techniques such as stochastic gradient descent (SGD), Adam, RMSProp, and their variants are commonly employed. Training leverages the concept of backpropagation, where gradients of the loss function are computed efficiently and propagated backward through the network layers to adjust parameters [4].

DL models typically excel when large datasets are available, given their high representational capacity and complexity. However, several strategies, such as transfer learning, data augmentation, and regularization techniques such as dropout and weight decay can enhance model performance and generalization even with

moderate-sized datasets, making deep learning broadly applicable across various domains.

In renewable energy applications, deep learning has made significant contributions in addressing complex prediction and classification challenges previously difficult to solve with traditional approaches. Tasks such as high-resolution solar irradiance forecasting from sky imagery, wind-power prediction from complex meteorological patterns, anomaly detection in grid infrastructure using sensor data, and the optimization of energy storage management in uncertain market conditions have all benefited considerably from advances in DL methodologies [18, 23, 24].

The remainder of this section explores several DL architectures, convolutional neural networks (CNNs), recurrent neural networks (RNNs), transformers, and auto encoders that have proven particularly effective in energy-related contexts.

8.2.4.1 Convolutional Neural Networks (CNNs)

CNNs excel in extracting meaningful local patterns from spatially structured data such as images, sensor arrays, or geospatial maps. A CNN processes inputs through layers consisting of convolutional filters, nonlinear activations (commonly Rectified Linear Units, ReLU), pooling operations, and fully connected layers at the output, as illustrated in Fig. 8.6. The convolutional operation is mathematically described as:

$$(\mathbf{f} * \mathbf{x})[i, j] = \sum_m \sum_n \mathbf{f}[m, n] \cdot \mathbf{x}[i - m, j - n], \tag{8.32}$$

where $\mathbf{f}$ is the convolutional filter (kernel), and $\mathbf{x}$ the input image or spatial data. Pooling operations (e.g., max or average pooling) then aggregate feature maps spatially, effectively reducing dimensions and computational complexity.

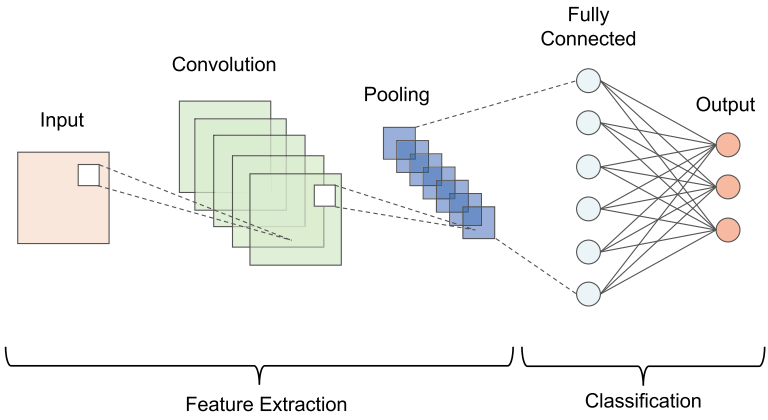


Fig. 8.6 Schematic structure of a convolutional neural network (CNN)

In renewable energy, CNNs have been successfully applied to detect defects or hot spots in photovoltaic modules using thermal imagery, to identify cloud formations from sky cameras to improve solar irradiance forecasting, and to interpret spatial wind patterns from radar or satellite imagery for wind resource assessment [25, 26].

8.2.4.2 Recurrent Neural Networks (RNNs)

RNNs are used to manipulate sequential data. They maintain an internal hidden state that captures temporal dependencies. Unlike feedforward neural networks, a standard RNN architecture has recurrent connections, as shown in Fig. 8.7a. Mathematically, given an input sequence $\{\mathbf{x}_t\}_{t=1}^{T}$, the hidden state $\mathbf{h}_t$ is updated recursively by:

$$\mathbf{h}_t = \sigma(W_{xh}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + b_h), \tag{8.33}$$

where W_{xh} and W_{hh} represent input-to-hidden and hidden-to-hidden weight matrices, respectively, b_h is a bias vector, and $\sigma(\cdot)$ is an activation function, typically hyperbolic tangent or sigmoid.

Due to the vanishing gradient problem, variants such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) are often employed. For instance,

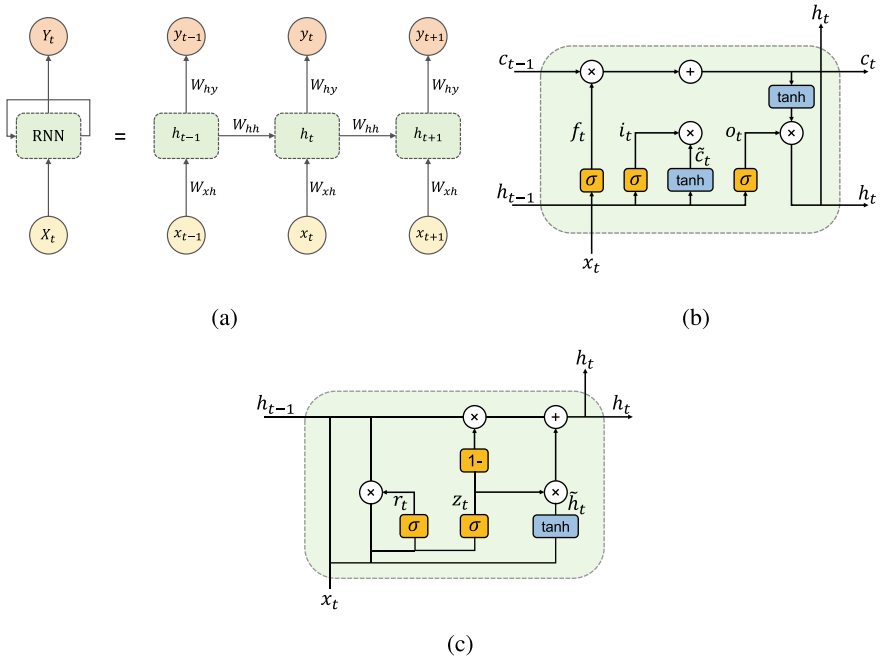


Fig. 8.7 **a** Standard recurrent neural network (RNN) architecture **b** Long short term memory (LSTM) **c** Gated recurrent unit

Fig. 8.7b depicts a LSTM cell unit. The LSTM introduces gating mechanisms (input, forget, and output gates) to selectively retain or discard temporal information as follows:

$$\mathbf{i}_t = \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + b_i), \tag{8.34}$$
$$\mathbf{f}_t = \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + b_f), \tag{8.35}$$
$$\mathbf{o}_t = \sigma(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + b_o), \tag{8.36}$$
$$\tilde{\mathbf{c}}_t = \tanh(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + b_c), \tag{8.37}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{8.38}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{8.39}$$

where $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ denotes the input, forget, and output gates, respectively. $\tilde{\mathbf{c}}_t$ is the new candidate memory state. $\mathbf{h}_{t-1}$ and $\mathbf{h}_t$ are the hidden states of the previous and current status. $\mathbf{c}_{t-1}$ and $\mathbf{c}_t$ are the memory blocks or cell states of the previous and current status [27].

Although LSTMs remain popular, GRUs offer a leaner alternative that addresses the problem of vanishing gradients with fewer gates and parameters (Fig. 8.7c). A GRU merges the cell and hidden states and removes the separate output gate, relying instead on an *update gate* $\mathbf{z}_t$ to control long-term information flow and a *reset gate* $\mathbf{r}_t$ to modulate the short-term context:

$$\mathbf{z}_t = \sigma\big(W_{xz}\mathbf{x}_t + W_{hz}\mathbf{h}_{t-1} + b_z\big), \tag{8.40}$$
$$\mathbf{r}_t = \sigma\big(W_{xr}\mathbf{x}_t + W_{hr}\mathbf{h}_{t-1} + b_r\big), \tag{8.41}$$
$$\tilde{\mathbf{h}}_t = \tanh\big(W_{xh}\mathbf{x}_t + \mathbf{r}_t \odot (W_{hh}\mathbf{h}_{t-1}) + b_h\big), \tag{8.42}$$
$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t. \tag{8.43}$$

Because the update gate directly blends the previous hidden state with the new candidate state $\tilde{\mathbf{h}}_t$, a GRU achieves long-range retention comparable to an LSTM but with fewer parameters, an advantage in memory-constrained or real-time applications.

RNNs and their variants are widely used for wind and solar power forecasting, capturing complex temporal dependencies inherent in renewable resource variability. They are also effective in predicting short-term electricity consumption and detecting anomalies in operational time series data from the energy infrastructure.

8.2.4.3 Transformers

Transformers have changed sequence modeling by replacing recurrence entirely with self-attention mechanisms [28]. A transformer architecture comprises multiple layers, each including multi-head self-attention and position wise feed-forward networks. Self-attention computes a weighted representation of each input vector $\mathbf{x}_i$ as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V, \tag{8.44}$$

where Q, K, and V represent queries, keys, and values (projections of input), and d_k is the dimensionality of these projections. Multi-head attention concatenates attention results computed in parallel, enhancing representational power.

Transformers have proven effective in forecasting renewable energy generation over multiple horizons, modeling complex interdependencies in large-scale electricity grids, and predicting energy consumption at high temporal resolution, capturing intricate seasonal patterns [29, 30].

8.2.4.4 Autoencoders (AEs)

Autoencoders (AE) are unsupervised neural networks designed to compress input data into a lower-dimensional latent representation and reconstruct it back [31]. As shown in Fig. 8.8 an AE consists of an encoder $f_{\text{enc}}(\cdot)$ mapping inputs to latent space and a decoder $f_{\text{dec}}(\cdot)$ reconstructing inputs from latent vectors:

$$\mathbf{z} = f_{\text{enc}}(\mathbf{x}; \theta_{\text{enc}}), \quad \hat{\mathbf{x}} = f_{\text{dec}}(\mathbf{z}; \theta_{\text{dec}}), \tag{8.45}$$

where training minimizes reconstruction error (e.g., mean squared error):

$$\mathcal{L}_{AE} = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2. \tag{8.46}$$

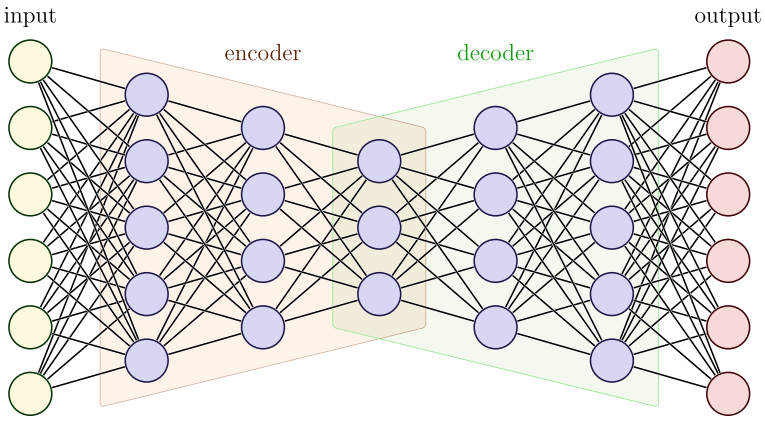


Fig. 8.8 Autoencoder architecture

Variational Autoencoders (VAEs) introduce probabilistic latent representations by enforcing the latent variables to follow a prior distribution (typically Gaussian), optimizing the evidence lower bound (ELBO) [32]:

$$\mathcal{L}_{VAE} = \mathbb{E}_{q_{\theta_{\text{enc}}}(\mathbf{z}|\mathbf{x})}[\log p_{\theta_{\text{dec}}}(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_{\theta_{\text{enc}}}(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})). \tag{8.47}$$

Autoencoders have used for anomaly detection in renewable-energy infrastructure, such as detecting faulty sensors in wind turbines, solar inverters, or battery storage systems [33]. They also aid in compressing operational data streams for efficient storage and analysis [34].

8.2.5 Training Considerations

Deep networks are data-hungry and computationally intensive. Training commonly relies on GPU or TPU accelerators, mini-batch processing, and careful hyperparameter tuning (learning rate schedules, depth, width, activation choice). When labelled data are scarce, as is typical for renewable energy in developing countries, transfer learning (initialising from networks pre-trained on related tasks) and data augmentation (adding noise, time warps, or synthetic images) often close the gap [35].

8.2.6 Interpretability and Trust

Although deep models are sometimes denominated as "black boxes," a growing tool set, feature map visualisation, saliency heat-maps, SHAP values, and counterfactual explanations, helps practitioners to verify that the network's decisions stem from physically meaningful hints (e.g., soiling patterns on panels or turbine wake effects) [36–38].

8.2.7 Extended and Emerging Learning Paradigms

While supervised, unsupervised, and reinforcement learning constitute the foundational paradigms of machine learning, current challenges in data availability, privacy, scalability, and real-time adaptation have motivated the development of several extended or hybrid paradigms. These paradigms adapt core machine learning principles to better align with constraints encountered in practice, particularly in domains such as renewable energy, smart infrastructure, and climate science. This section introduces some emerging paradigms, each addressing critical limitations of conventional learning frameworks.

8.2.7.1 Self-Supervised Learning (SSL)

Self-supervised learning (SSL) bridges supervised and unsupervised learning by generating supervision signals automatically from unlabeled data [39]. SSL methods define auxiliary or pretext tasks that enable a model to learn useful representations without requiring manually annotated labels. Once trained, the learned features can be fine-tuned on downstream tasks.

Formally, let $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^{n}$ be an unlabeled dataset. SSL defines a function $T : \mathbb{R}^m \to \mathbb{R}^m$ representing a transformation (e.g., rotation, masking), and trains a model f_θ to solve a task involving both original and transformed instances:

$$\hat{y}_i = f_\theta(T(\mathbf{x}_i)), \quad \text{with pseudo-label } y_i \text{ derived from } \mathbf{x}_i. \tag{8.48}$$

A prominent SSL method is contrastive learning, where the objective is to bring together representations of similar inputs (positive pairs) while pushing apart representations of dissimilar inputs (negative pairs) [40]. This is achieved by minimizing the contrastive loss:

$$\mathcal{L}_{\text{contrast}} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2n} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)} \tag{8.49}$$

where z_i and z_j are embeddings of positive pairs, $\text{sim}(\cdot)$ denotes cosine similarity, and τ is a temperature parameter.

SSL has achieved state-of-the-art performance in domains like computer vision (e.g., SimCLR, MoCo), natural language processing (e.g., BERT), and is increasingly utilized in geospatial and renewable energy analytics, where large amounts of unlabeled sensor data are available [40].

8.2.7.2 Semi-Supervised Learning

Semi-supervised learning leverages both labeled and unlabeled data to improve generalization when annotated data is limited [41]. Let $\mathcal{D}_l = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n_l}$ be a small labeled set and $\mathcal{D}_u = \{\mathbf{x}_j\}_{j=1}^{n_u}$ be a large unlabeled set. The goal is to train a function f_θ that benefits from both datasets:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{sup}}(\mathcal{D}_l) + \lambda \mathcal{L}_{\text{unsup}}(\mathcal{D}_u) \tag{8.50}$$

where $\mathcal{L}_{\text{unsup}}$ might involve consistency regularization or entropy minimization, and λ balances their contributions.

Semi-supervised methods are widely applied in fields like remote sensing, wind turbine monitoring, and grid anomaly detection, where expert labeled data is expensive, but raw data is plentiful [42].

8.2.7.3 Online and Continual Learning

Online learning refers to algorithms that learn incrementally from data streams, updating model parameters in real time [43]. Given a sequence of data points $\{(\mathbf{x}_t, y_t)\}_{t=1}^{\infty}$, the model f_{θ_t} is updated at each time step using:

$$\theta_{t+1} = \theta_t - \alpha_t \nabla \mathcal{L}(f_{\theta_t}(\mathbf{x}_t), y_t) \tag{8.51}$$

Continual learning, a related concept, focuses on preserving knowledge over time and avoiding *catastrophic forgetting* when new tasks are learned sequentially. Methods such as Elastic Weight Consolidation (EWC) introduce regularization terms to retain previously learned parameters [44].

These paradigms are essential for dynamic environments such as smart grids or autonomous renewable systems that must adapt to evolving conditions without retraining from scratch [45].

8.2.7.4 Active Learning

Active learning is a strategy to optimize labeling efforts by allowing the model to select the most informative data points for annotation [46]. Let U be the unlabeled pool. The model selects a subset $Q \subset U$ using an acquisition function (e.g., entropy, margin sampling) and requests labels from an oracle (e.g., human expert):

$$Q = \arg\max_{\mathbf{x} \in U} \text{Uncertainty}(f_\theta(\mathbf{x})) \tag{8.52}$$

Active learning is beneficial in high-cost domains like structural integrity analysis, where simulations or field experiments are expensive. By querying the most ambiguous cases, fewer labels are needed to achieve high performance [46, 47].

8.2.7.5 Federated Learning

Federated learning enables multiple decentralized devices or clients to collaboratively on a shared model without transmitting raw data. Each client k trains a local model on its private data $\mathcal{D}_k$ and sends only the updated parameters θ_k to a central server. The server aggregates them via weighted averaging:

$$\theta = \sum_{k=1}^{K} \frac{n_k}{n} \theta_k \tag{8.53}$$

where n_k is the number of data samples on client k, and $n = \sum_k n_k$.

This paradigm is gaining traction in energy systems, where data privacy and regulatory compliance are critical, for example, in energy usage profiling in distributed smart meters or microgrids [48].

8.3 Conclusion

In this chapter, we explore the fundamental concepts and methodologies of Artificial Intelligence (AI) and Machine Learning (ML), emphasizing their role within renewable energy systems. Beginning with preliminaries of AI, we established a clear understanding of foundational principles that underpin intelligent computational methods. Subsequently, we delved into the core foundations of ML, examining supervised, unsupervised, and reinforcement learning paradigms, each demonstrating distinct approaches to extracting meaningful insights from data. Deep learning techniques were highlighted.

In addition, we discussed vital considerations in the training of machine learning models, including best practices to achieve reliable, robust, and efficient systems. The significance of interpretability and trust was underscored, acknowledging the growing need for transparent and understandable AI solutions in critical infrastructure such as energy systems. Finally, we introduced extended and emerging learning paradigms, pointing towards future advancements and innovative methodologies.

References

1. Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. pearson, 2016.
2. Roberto Confalonieri, Ludovik Coba, Benedikt Wagner, and Tarek R Besold. A historical perspective of explainable artificial intelligence. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(1):e1391, 2021.
3. Christian Janiesch, Patrick Zschech, and Kai Heinrich. Machine learning and deep learning. *Electronic markets*, 31(3):685–695, 2021.
4. Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
5. Andrew G Barto. Reinforcement learning: An introduction. by richard's sutton. *SIAM Rev*, 6(2):423, 2021.
6. Qiang Wang, Yuanfan Li, and Rongrong Li. Integrating artificial intelligence in energy transition: A comprehensive review. *Energy Strategy Reviews*, 57:101600, 2025.
7. Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 50(8):3668–3681, 2019.
8. Zhi-Hua Zhou. *Machine learning*. Springer nature, 2021.
9. Tadayoshi Fushiki. Estimation of prediction error by using k-fold cross-validation. *Statistics and Computing*, 21:137–146, 2011.
10. FY Osisanwo, JET Akinsola, O Awodele, JO Hinmikaiye, O Olakanmi, J Akinjobi, et al. Supervised machine learning algorithms: classification and comparison. *International Journal of Computer Trends and Technology (IJCTT)*, 48(3):128–138, 2017.

11. Adrian Stetco, Fateme Dinmohammadi, Xingyu Zhao, Valentin Robu, David Flynn, Mike Barnes, John Keane, and Goran Nenadic. Machine learning methods for wind turbine condition monitoring: A review. *Renewable energy*, 133:620–635, 2019.
12. Cyril Voyant, Gilles Notton, Soteris Kalogirou, Marie-Laure Nivet, Christophe Paoli, Fabrice Motte, and Alexis Fouilloy. Machine learning methods for solar radiation forecasting: A review. *Renewable energy*, 105:569–582, 2017.
13. Wei-Tung Wang, Yi-Leh Wu, Cheng-Yuan Tang, and Maw-Kae Hor. Adaptive density-based spatial clustering of applications with noise (dbscan) according to data. In *2015 International conference on machine learning and cybernetics (ICMLC)*, volume 1, pages 445–451. IEEE, 2015.
14. Nema Salem and Sahar Hussein. Data dimensional reduction and principal components analysis. *Procedia Computer Science*, 163:292–299, 2019.
15. Anna C Belkina, Christopher O Ciccolella, Rina Anno, Richard Halpert, Josef Spidlen, and Jennifer E Snyder-Cappione. Automated optimized parameters for t-distributed stochastic neighbor embedding improve visualization and analysis of large datasets. *Nature communications*, 10(1):5415, 2019.
16. John Healy and Leland McInnes. Uniform manifold approximation and projection. *Nature Reviews Methods Primers*, 4(1):82, 2024.
17. Vasilis Michalakopoulos, Elissaios Sarmas, Ioannis Papias, Panagiotis Skaloumpakas, Vangelis Marinakis, and Haris Doukas. A machine learning-based framework for clustering residential electricity load profiles to enhance demand response programs. *Applied Energy*, 361:122943, 2024.
18. Mariam Ibrahim, Ahmad Alsheikh, Feras M Awaysheh, and Mohammad Dahman Alshehri. Machine learning schemes for anomaly detection in solar power plants. *Energies*, 15(3):1082, 2022.
19. Kanishka Tyagi, Chinmay Rane, Raghavendra Sriram, and Michael Manry. Unsupervised learning. In *Artificial intelligence and machine learning for EDGE computing*, pages 33–52. Elsevier, 2022.
20. Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
21. Grigory Neustroev, Sytze PE Andringa, Remco A Verzijlbergh, and Mathijs M De Weerdt. Deep reinforcement learning for active wake control. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pages 944–953, 2022.
22. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
23. Huaizhi Wang, Zhenxing Lei, Xian Zhang, Bin Zhou, and Jianchun Peng. A review of deep learning for renewable energy forecasting. *Energy Conversion and Management*, 198:111799, 2019.
24. Jose L Crespo-Vazquez, C Carrillo, E Diaz-Dorado, Jose A Martinez-Lorenzo, and Md Noor-E-Alam. A machine learning based stochastic optimization framework for a wind and storage power plant participating in energy pool market. *Applied Energy*, 232:341–357, 2018.
25. Hale Bakır, Francis A Kuzhippallil, and Adel Merabet. Automatic detection of deteriorated photovoltaic modules using irt images and deep learning (cnn, lstm) strategies. *Engineering Failure Analysis*, 146:107132, 2023.
26. Cong Feng and Jie Zhang. Solarnet: A sky image-based deep convolutional neural network for intra-hour solar forecasting. *Solar Energy*, 204:71–78, 2020.
27. Jhon J Quiñones, Luis R Pineda, Jason Ostanek, and Luciano Castillo. Towards smart energy management for community microgrids: Leveraging deep learning in probabilistic forecasting of renewable energy sources. *Energy Conversion and Management*, 293:117440, 2023.
28. Derya Soydaner. Attention mechanism in neural networks: where it comes and where it goes. *Neural Computing and Applications*, 34(16):13371–13385, 2022.
29. Erick Giovani Sperandio Nascimento, Talison AC de Melo, and Davidson M Moreira. A transformer-based deep neural network with wavelet transform for forecasting wind speed and wind energy. *Energy*, 278:127678, 2023.

30. Jingxuan Liu, Haixiang Zang, Lilin Cheng, Tao Ding, Zhinong Wei, and Guoqiang Sun. A transformer-based multimodal-learning framework using sky images for ultra-short-term solar irradiance forecasting. *Applied Energy*, 342:121160, 2023.
31. Walter Hugo Lopez Pinaya, Sandra Vieira, Rafael Garcia-Dias, and Andrea Mechelli. Autoencoders. In *Machine learning*, pages 193–208. Elsevier, 2020.
32. Lucas Pinheiro Cinelli, Matheus Araújo Marins, Eduardo Antúnio Barros da Silva, and Sérgio Lima Netto. Variational autoencoder. In *Variational methods for machine learning with applications to deep networks*, pages 111–149. Springer, 2021.
33. Eric Stefan Miele, Fabrizio Bonacina, and Alessandro Corsini. Deep anomaly detection in horizontal axis wind turbines using graph convolutional autoencoders for multivariate time series. *Energy and AI*, 8:100145, 2022.
34. Seunghyoung Ryu, Hyungeun Choi, Hyoseop Lee, and Hongseok Kim. Convolutional autoencoder based feature extraction and clustering for customer load analysis. *IEEE Transactions on Power Systems*, 35(2):1048–1060, 2019.
35. Yassine Himeur, Mariam Elnour, Fodil Fadli, Nader Meskin, Ioan Petri, Yacine Rezgui, Faycal Bensaali, and Abbes Amira. Next-generation energy systems for sustainable smart cities: Roles of transfer learning. *Sustainable Cities and Society*, 85:104059, 2022.
36. Zhe Song, Sunliang Cao, and Hongxing Yang. An interpretable framework for modeling global solar radiation using tree-based ensemble machine learning and shapley additive explanations methods. *Applied Energy*, 364:123238, 2024.
37. Sahil Verma, Varich Boonsanong, Minh Hoang, Keegan Hines, John Dickerson, and Chirag Shah. Counterfactual explanations and algorithmic recourses for machine learning: A review. *ACM Computing Surveys*, 56(12):1–42, 2024.
38. T Nathan Mundhenk, Barry Y Chen, and Gerald Friedland. Efficient saliency maps for explainable ai. arXiv preprint arXiv:1911.11293, 2019.
39. Veenu Rani, Syed Tufael Nabi, Munish Kumar, Ajay Mittal, and Krishan Kumar. Self-supervised learning: A succinct review. *Archives of Computational Methods in Engineering*, 30(4):2761–2775, 2023.
40. Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *IEEE transactions on knowledge and data engineering*, 35(1):857–876, 2021.
41. Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine learning*, 109(2):373–440, 2020.
42. Elizabeth Cook, Shuman Luo, and Yang Weng. Solar panel identification via deep semi-supervised learning and deep one-class classification. *IEEE Transactions on Power Systems*, 37(4):2516–2526, 2021.
43. German I Parisi and Vincenzo Lomonaco. Online continual learning on sequences. In *Recent Trends in Learning From Data: Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL2019)*, pages 197–221. Springer, 2020.
44. Benjamin Maschler, Hannes Vietz, Nasser Jazdi, and Michael Weyrich. Continual learning of fault prediction for turbofan engines using deep learning with elastic weight consolidation. In *2020 25th IEEE international conference on emerging technologies and factory automation (ETFA)*, volume 1, pages 959–966. IEEE, 2020.
45. German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.
46. Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *ACM computing surveys (CSUR)*, 54(9):1–40, 2021.
47. Burr Settles. Active learning literature survey. 2009.
48. Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020.