Deep Learning for Environmental Impact Analysis

Learning Outcomes

After completing this unit, students will be able to:

- Explain the concept and role of Neural Networks in environmental applications.
- Explain the core ideas behind deep learning (DL) and how Neural Networks are structured and trained.
- Describe how convolutional and recurrent Neural Networks are used in environmental impact analysis.
- Apply a basic deep learning model to analyze environmental data (e.g., pollution, land cover, or temperature trends).
- Interpret results from deep learning models and discuss their role in supporting sustainable decisions.
- Develop simple prototypes or scenario analyses demonstrating deep learning applications for sustainability.
- Discuss how DL can help decision-makers ecological risk assessment.

Introduction

Deep learning (DL) is a branch of Machine Learning that uses artificial Neural Networks to learn complex patterns in data. Unlike traditional algorithms, deep learning models can automatically extract important features, making them especially useful for large and complex environmental datasets. This lecture introduces the fundamental ideas of deep learning and shows how these methods support climate adaptation, ecological protection, and sustainable decision-making.

Definition and distinction: Machine Learning vs. Deep Learning

Artificial intelligence was born in the 1950s, when a handful of pioneers from the nascent field of computer science started asking whether computers could be made to “think” – a question whose ramifications we’re still exploring today [1]. Concisely, AI can be described as the effort to automate intellectual tasks normally performed by humans. As such, AI is a general field that encompasses Machine Learning and deep learning, but that also includes many more

approaches that may not involve any learning. Three nested ovals representing the relationship between artificial intelligence, Machine Learning, and deep learning shown in Figure 6.1.

Machine Learning is a way of programming computers so that they can learn from data and improve their performance without being told the exact rule for every situation. In classic Machine Learning, humans usually choose which input characteristics (features) are important, for example “average temperature”, “amount of rainfall”, or “soil moisture”, and then train an algorithm to make predictions or decisions based on these features. This approach works well when the data is relatively small, structured in tables, and when it is important to keep models simple and easy to explain.

Deep Learning is a special type of Machine Learning that uses many-layered artificial Neural Networks to learn directly from raw data such as images, sound, or long time series. Instead of humans designing features by hand, a deep neural network automatically learns useful patterns step by step in its layers, for example from pixels to shapes to objects in an image. Deep Learning usually needs much more data and computing power, but it is very good at solving complex tasks like image recognition, speech recognition, and understanding natural language.

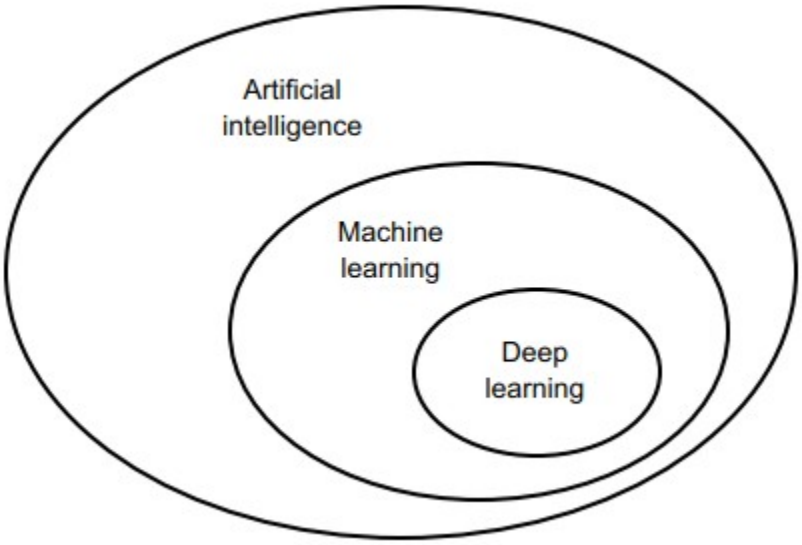

Figure 6.1 Artificial intelligence levels

The key distinction is that Machine Learning is a broader family of methods, where humans typically define features and models are often smaller and easier to interpret, while Deep Learning is a subset that relies on deep Neural Networks to learn features and patterns automatically from large amounts of data. In practice, Machine Learning is often used with structured environmental data in spreadsheets or databases, whereas Deep Learning is preferred when working with satellite images, sensor grids, or other high-dimensional signals. For this course, students should remember a simple rule: all Deep Learning is Machine Learning, but not all Machine Learning is Deep Learning.

The environmental data explosion: satellites, drones, IoT sensors, climate models

Modern environmental science has moved far beyond simple manual measurements; we now have a global network of digital "eyes" and "ears" monitoring the Earth. Satellites capture high-resolution images of the planet daily, tracking changes like melting ice or urban growth, while drones map forests and coastlines in 3D. Simultaneously, millions of Internet of Things (IoT) sensors placed in oceans, soil, and cities stream continuous data about temperature, pollution, and humidity. This creates a massive, messy library of information – often called "unstructured data" – that is far too vast for humans to analyze by hand.

In addition to real-world observations, scientists rely on complex climate models running on supercomputers to simulate weather patterns decades into the future. When we combine these simulations with satellite imagery and sensor streams, we are left with datasets containing millions of variables, or "high-dimensional" data. Traditional Machine Learning methods often struggle here because they require humans to manually simplify the data first (feature engineering). With the sheer volume of data we produce today, manually creating rules for every satellite photo or sensor reading is impossible.

Deep Learning becomes essential. Unlike older algorithms that stop improving after a certain point, deep learning models actually perform better as they are fed more data. They are uniquely capable of processing this "data explosion" – using Convolutional Neural Networks (CNNs) to "see" patterns in satellite photos and Recurrent Neural Networks (RNNs) to "remember" trends in climate sensor logs. By automating the analysis of this complex data, deep learning turns a flood of raw information into actionable insights for protecting our planet.

Why deep learning matters: handling complex, non-linear, and high-dimensional data

Deep learning is essential for environmental science because nature rarely follows a straight line; instead, ecological relationships are highly **non-linear**, meaning a small change in one factor (like temperature) can trigger disproportionately large or complex effects (like a storm). While traditional algorithms struggle to "draw" these complex curves, deep learning models use activation functions to effectively approximate any continuous, curving relationship found in nature. Furthermore, environmental data is often **high-dimensional**, consisting of massive inputs such as satellite images with millions of pixels or simultaneous readings from thousands of weather sensors. While older methods often get overwhelmed by this sheer volume of information – a problem known as the "curse of dimensionality" – deep learning architectures are specifically designed to handle these vast, unstructured datasets, automatically disentangling the noise to find meaningful patterns without requiring humans to simplify the data first. Examples of DL in sustainability include:

- Mapping deforestation. Convolutional Neural Networks (CNNs) analyze satellite images from regions like the Amazon to automatically detect where forest has been cut down or degraded, helping authorities react faster and plan protection measures.

- Forecasting floods. Recurrent or hybrid deep learning models learn from past rainfall, river levels, and soil moisture to predict when and where rivers are likely to overflow, giving communities more time to prepare and reduce damage.
- Classifying coral reefs. Deep learning models trained on thousands of underwater photos can recognize different coral species and detect bleaching or disease, supporting marine biologists in monitoring reef health much faster than manual surveys.
- Predicting emissions. Deep learning models can use data about energy use, traffic, and industrial activity to estimate daily or future carbon dioxide emissions, helping governments and companies track progress toward climate targets.

Basics of Deep Neural Networks

This module establishes the theoretical and practical framework of **Deep Learning (DL)**, a subfield of artificial intelligence that has revolutionized how we interpret complex datasets. DL shifts the paradigm from traditional statistical modeling toward automated feature engineering.

While traditional statistical methods, such as linear regression, rely heavily on manual feature selection – where researchers must identify and pre-process specific variables to determine their influence – Deep Neural Networks (DNN) are designed to learn these representations autonomously. Through a series of hidden layers, a DNN decomposes raw data into multiple levels of abstraction.

- **Initial Layers:** Capture simple patterns or “low-level” features.
- **Deeper Layers:** Combine these patterns to identify intricate, high-level structures within the data.

This hierarchical learning capability is particularly vital for **environmental analytics**. Natural systems are rarely governed by simple, additive relationships; instead, they are defined by variables – such as humidity, atmospheric pressure, and particulate matter – that interact in non-linear and highly complex ways.

Traditional models often struggle with these “chaotic” interactions, but DNNs excel at:

- **Pattern Recognition:** Identifying subtle shifts in climate data that precede extreme weather events,
- **Multi-Dimensional Correlation:** Mapping how disparate environmental factors (like soil moisture and wind speed) converge to influence local ecosystems,

- **Predictive Accuracy:** Providing more robust forecasts by capturing the non-linear “noise” that simpler models might discard as outliers.

By mastering these foundations, researchers can move beyond mere data observation and begin building predictive tools capable of handling the scale and volatility of our changing planet.

The Architecture of Neural Networks: Concept of Layers

A neural network can be understood as a mathematical function mapping some set of input values to output values.

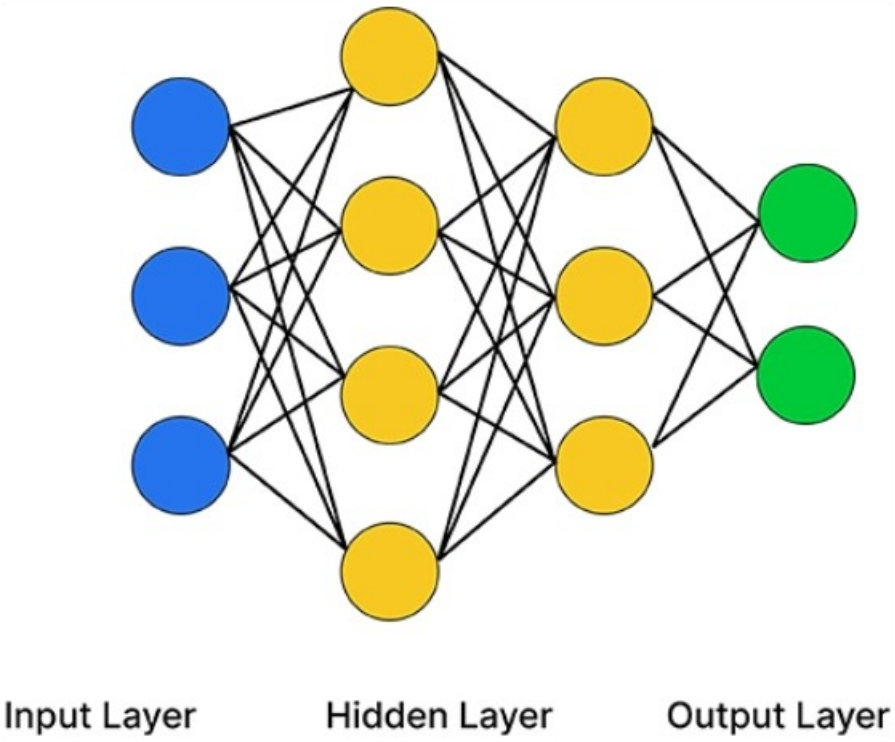

Figure 6.2 Neural Networks Architecture

A Deep Feedforward Network [1], or Multilayer Perceptron (MLP), is composed of three types of layers (Figure 6.2):

- **Input Layer:**
  This layer receives the raw environmental data. It does not perform any computation. The number of neurons in this layer corresponds exactly to the number of features in the dataset.

- *Environmental Context:* If we are analyzing air quality, the input neurons might represent specific sensor readings, for example: $[x_1: \text{Temperature}, x_2: \text{Wind Speed}, x_3: \text{Humidity}]$.

- **Hidden Layers:**
  These layers are sandwiched between the input and output. They are called "hidden" because their values are not observed in the training data.
  - **Heaton** explains that as data passes through these layers, the network calculates weighted sums of inputs. "Deep" learning simply refers to a network having more than one hidden layer.
  - *Function:* These layers act as feature extractors. Early layers might detect simple patterns (e.g., daily temperature cycles), while deeper layers combine these to detect complex phenomena (e.g., specific conditions preceding a heatwave).
- **Output Layer:**
  The final layer produces the prediction (Figure 6.3).
  - *Regression:* A single neuron outputting a continuous value (e.g., predicted rainfall in mm).
  - *Classification:* Multiple neurons representing probabilities for different classes (e.g., Class 0: "Safe Air," Class 1: "Hazardous Air").

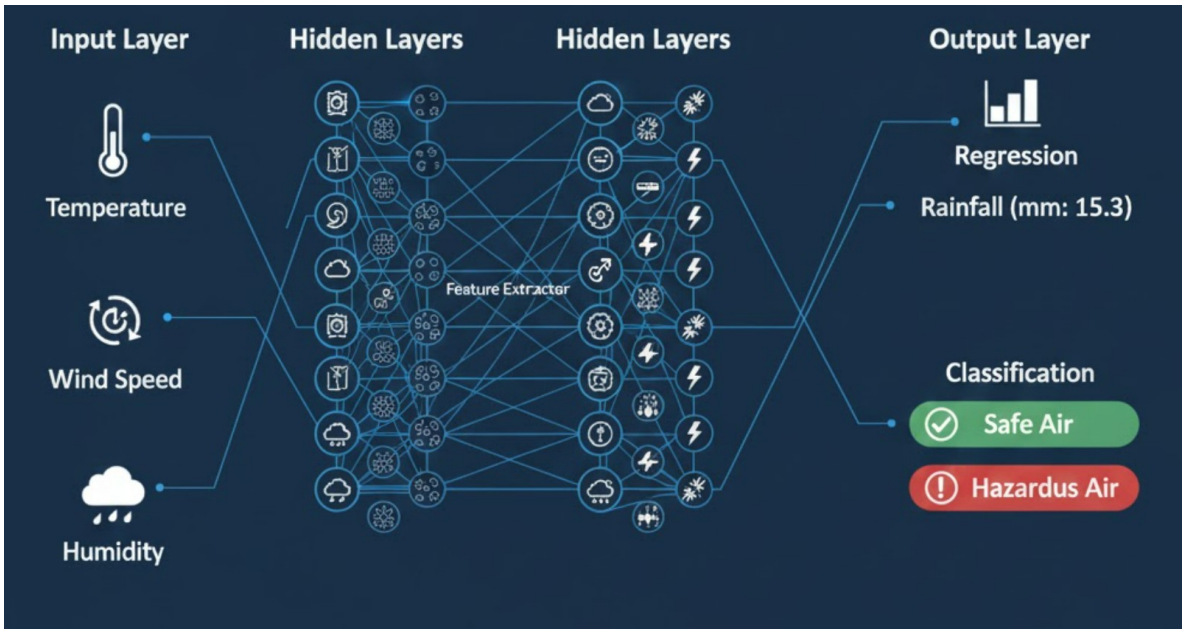

Figure 6.3 Neural Networks Layers

Activation Functions: Introducing Non-Linearity

If a neural network consisted only of weighted sums and bias additions, it would remain a linear model – essentially a giant regression equation – no matter how many layers it had. In such a system, the composition of linear functions is itself a linear function, meaning the network could never grasp the intricate, "jagged", or curved relationships found in real-world data.

To model complex environmental phenomena – such as the **non-linear growth rates of bacteria** or the **chaotic threshold of a storm cell** – we must introduce a mathematical "gatekeeper" at each neuron. This is the role of the **Activation Function** (Figure 6.4).

- **ReLU (**Rectified Linear Unit): defined as $f(x)=max\ (0,x)$, a mathematical function that simply passes positive signals through but completely blocks any negative ones.
  - The Rectified Linear Unit is the current "gold standard" for hidden layers in deep learning,
  - The Logic: It passes positive signals through unchanged but completely blocks any negative signals,
  - Computational Efficiency: Unlike complex exponential functions, ReLU only requires a simple threshold check, making it incredibly fast to compute during training,

- Mitigating Vanishing Gradients: In very deep networks, gradients (the signals used to "teach" the network) tend to get smaller and smaller until the network stops learning. ReLU[1] helps prevent this by maintaining a constant gradient for all positive inputs,
- Sparsity: By outputting zero for all negative inputs, ReLU creates "sparse" representations where only a subset of neurons are active at any given time. This mimics the biological efficiency of the human brain.

- **Sigmoid**
  - Binary Classification**:** Because its output stays within the (0,1) range, it is the ideal choice for the **output layer** of a model predicting a binary outcome. *Example:* Predicting the probability of a "Flood" (>0.5) vs. "No Flood" (<0.5).
  - Historical Context: While once used in hidden layers, it has largely been replaced by ReLU there because Sigmoid "saturates" (the ends of the curve become very flat), which can cause learning to stall.

- **Softmax:**
  - While Sigmoid handles "A vs. B," Softmax is designed for mutually exclusive categories,
  - The Probability Distribution: Softmax takes a vector of raw scores from the final layer and squashes them into a probability distribution. The key feature is that all outputs sum exactly to 1.0 (100%),
  - Environmental Application: If you are classifying land cover from satellite imagery, Softmax ensures the model doesn't just say a pixel is "mostly Forest," but gives a precise breakdown: 85% Forest, 10% Urban, 5% Water,
  - Decision Making: The class with the highest probability is typically selected as the model's final prediction.

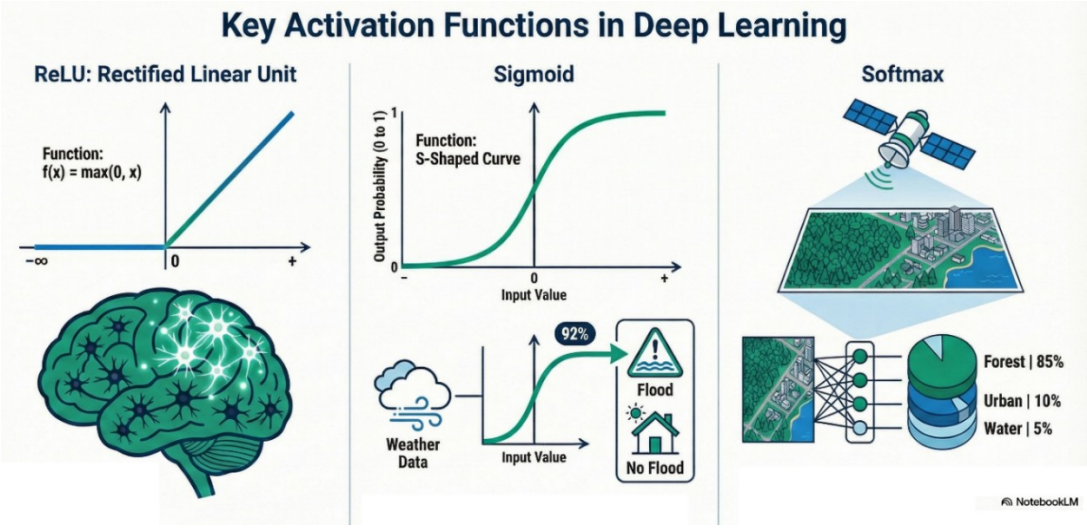

Figure 6.4 Key Activation Functions

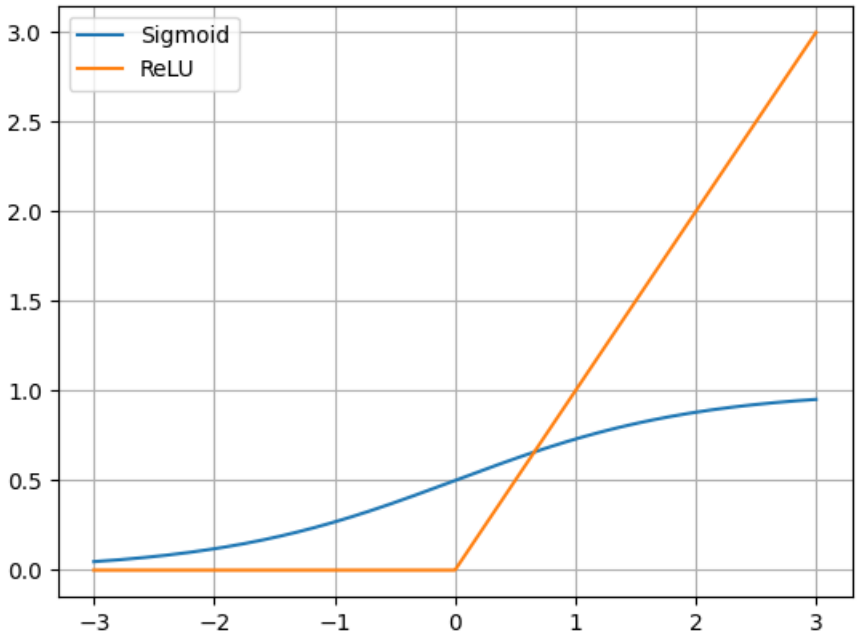

Figure 6.5 Sigmoid and ReLU activation functions example

The Figure 6.5 shows the plots of Sigmoid and ReLU activation functions.

Training and Optimization

Learning in a neural network is not a single event, but a continuous cycle of trial and error. To master complex environmental data, the network follows a three-step process: **Forward**

Propagation (making a guess), **Loss Calculation** (measuring the error), and **Backpropagation** (correcting the internal weights).

- **The Loss Function (Objective Function):**

  Before a network can improve, it needs a mathematical definition of "wrong." This is the **Loss Function** (also called the Objective Function). It acts as a single scalar value that represents the distance between the network's prediction and the actual ground-truth data.

  - **Mean Squared Error (MSE):** Ideal for regression tasks, such as predicting the exact concentration of CO2 in the atmosphere or the temperature for the following day. It squares the differences, which penalizes large errors more heavily than small ones.
  - **Cross-Entropy Loss:** Typically used for classification tasks (e.g., "Is this satellite pixel Forest or Urban?"). It measures how well the predicted probability distribution matches the actual label.

- **Backpropagation (Conceptual Overview):**

  Once the loss is calculated, the network must determine which specific weights and biases contributed most to the error. This is achieved through **Backpropagation**.

  Information flows backward from the output layer toward the input[2]. Using mathematical principles, the algorithm determines exactly how much each individual weight contributed to the final error.

  - A **gradient** is essentially a vector that points in the direction of the greatest increase in error.
  - By calculating these gradients, the network learns exactly how much a slight increase or decrease in a specific weight would affect the final prediction.

- **Gradient Descent:**

  With the gradients in hand, the network uses an Optimizer to update its weights. The most common metaphor for this is Gradient Descent.

  Imagine you are standing on a foggy mountain (the "Loss Landscape") and your goal is to reach the lowest valley (the point of minimum error). You cannot see the valley, but you can feel the slope beneath your feet.

- **The Gradient:** Tells you which way is "uphill.",
- **The Step:** To reduce error, you take a step in the *opposite* direction of the gradient (downhill),
- **Learning Rate:** This determines the size of your step. If the step is too large, you might overshoot the valley; if it's too small, the journey will take forever.

- **Common Optimizers**

  While basic Gradient Descent is the foundation, modern deep learning uses more sophisticated "hikers" to find the bottom of the hill:

  - **Stochastic Gradient Descent (SGD):** Instead of looking at the entire dataset at once, it calculates gradients using small "batches" of data, making the process faster and more dynamic.
  - **Adam (Adaptive Moment Estimation):** Currently the most popular optimizer. It automatically adjusts the learning rate for each weight, slowing down when it gets close to a solution and speeding up when the path is clear.

Overfitting and Generalization in Environmental Data

A critical challenge in training Neural Networks is **overfitting**. Overfitting occurs when a model learns the training data *too* well, including the random noise or anomalies specific to that dataset, rather than the underlying physical laws. **Heaton** emphasizes that a model that overfits will have low error on training data but high error on new, unseen data (poor **generalization**).

Environmental Consequence: An overfitting model trained on weather data from 2010–2020 might memorize specific storms from those years. If deployed in 2025, it might fail to predict a drought because it is looking for the exact patterns of 2010, rather than general climate indicators.

Mitigation Strategies:

- **Dropout:** A technique where randomly selected neurons are ignored (dropped out) during training. This forces the network to learn robust features rather than relying on specific pathways.

- **Early Stopping:** Monitoring the model's performance on a separate "validation set" and stopping training immediately when validation error starts to increase, even if training error is still decreasing.

Sequence Modeling: From MLP to LSTM

Unlike traditional algorithms or basic Feed-Forward networks that treat data points largely independently, environmental data is inherently sequential. To process such data effectively, we employ a **Long Short-Term Memory (LSTM)** network [2].

LSTM is a specialized Recurrent Neural Network (RNN) designed to 'remember' long-term trends and solve the mathematical challenges of retaining information over long periods (known as the "vanishing gradient" problem). Imagine a conveyor belt running parallel to the sequence you are processing. Information from the past can jump onto this conveyor belt, be transported across many time steps, and jump off intact when needed. LSTMs utilize a system of learnable **gates** (input, output, and forget gates) that regulate this flow, deciding exactly what information to keep and what to discard.

Ideally suited for environmental applications, this architecture allows for deeper analysis than simple regression. Instead of using just yesterday's data to predict today, an LSTM model looks back at a **sequence of the past 14 days** to identify warming or cooling trends before making a forecast.

Core Deep Learning Architectures in Environmental

This section explores the fundamental neural network architectures used to model environmental systems. Rather than focusing on complex mathematical derivations, we focus on the **intuition** behind these models: understanding *when* to use a specific architecture based on the structure of the environmental data (e.g., tabular sensor readings, satellite imagery, or historical weather time-series).

Feed-Forward Networks (FFN): The Foundation

The Feed-Forward Network, often called a Multilayer Perceptron (MLP), is the quintessential Deep Learning model. In these networks, information flows in one direction – from input nodes, through hidden layers, to output nodes – without any loops.

How it works: FFNs approximate functions by mapping input vectors to output targets. They are excellent at learning global patterns in tabular data.

Environmental Application: FFNs are suitable for **static prediction tasks** where the order of data points does not matter or time dependencies are weak.

Example: Predicting the current pollution level at a specific location based on a snapshot of current wind speed, temperature, and traffic density.

Limitation: FFNs have no "memory." They process each input independently, making them less effective for data where the sequence matters, such as prolonged heatwaves or seasonal trends.

Convolutional Neural Networks: Processing Spatial Data

While FFNs treat input features independently, environmental data often has a grid-like topology, such as satellite images or radar maps. CNNs are specialized for processing such data.

How it works: Instead of connecting every neuron to every other neuron, CNNs use small "filters" (kernels) that slide over the input image to detect local patterns like edges, textures, or shapes. This allows the model to recognize a feature (e.g., a tree or a building) regardless of where it appears in the image (translation invariance).

Environmental Application: CNNs are the standard for **remote sensing**. *Example:* analyzing satellite imagery to map deforestation in the Amazon or classifying coral reef health from underwater photography.

Recurrent Architectures (RNN, LSTM, GRU): Analyzing Time-Series

Environmental data is inherently sequential; knowing the temperature yesterday is crucial for predicting the temperature today. Recurrent Neural Networks (RNNs) address this by introducing an internal loop that allows information to persist.

Standard RNNs: These networks maintain a "state" containing information relative to what they have seen so far. However, standard RNNs suffer from the "vanishing gradient problem," making them unable to remember information over long sequences (e.g., relating a rainfall event two weeks ago to river levels today).

Long Short-Term Memory (LSTM): To solve the memory problem, LSTMs were introduced with a specialized internal structure involving "gates" (input, forget, and output gates). These gates allow the network to explicitly learn what to remember and what to forget over long periods.

Gated Recurrent Unit (GRU): A streamlined variant of LSTM that is often faster to train while offering comparable performance for many environmental time-series tasks. Environmental Application: These architectures are critical for forecasting.

Example: Predicting flood risks based on a sequence of rainfall and soil moisture data over the past 14 days, or modeling climate trends where historical context defines future states.

Responsible AI: Overfitting and Ethics

Deploying Deep Learning in the real world requires more than just high accuracy; it demands reliability and ethical responsibility.

Avoiding Overfitting: A common pitfall is creating a model that memorizes the training data (e.g., specific weather anomalies from 2010) but fails to generalize to the future. We utilize techniques like **Dropout** (randomly ignoring neurons during training to prevent reliance on specific pathways) and **Early Stopping** (halting training when validation performance degrades) to ensure models are robust.

Ethical & Interpretable AI: Deep learning models are often criticized as "black boxes." In environmental policy, we must strive for transparency. Technology is never neutral; technical choices are also ethical choices. When models influence decisions – such as water allocation or disaster response – it is vital to validate that they do not perpetuate biases found in historical data and that their predictions can be interpreted by domain experts.

From Deep Learning to Climate-Adaptive Decision-Making

Once a Deep Learning model is trained and validated, the focus shifts from technical optimization to **operational impact**. Achieving a low error rate (for example, using metrics like Mean Absolute Error or Root Mean Squared Error) is a technical milestone, but the ultimate goal is translating these numerical predictions into meaningful, climate-adaptive actions. Predictive modeling drives three key pillars of sustainability:

- **Proactive Planning.** Using long-term forecasts to design resilient infrastructure (e.g., sizing drainage systems based on predicted rainfall intensity).
- **Risk Reduction:** Implementing early-warning systems where model outputs directly trigger safety protocols.
- **Resource Allocation:** Optimizing the distribution of water or energy based on anticipated environmental conditions.

Predictive Modeling for Climate Adaptation

Deep learning replaces slow physical simulations with rapid data-driven models, enabling real-time climate adaptation in key areas:

- Forecasting extreme weather patterns (heatwaves, storms, droughts)
- Predicting hydrological changes (flooding, river flow variability, water scarcity)
- Modeling wildfire probability and spread
- Anticipating air-quality deterioration and human health impacts.

These models enable climate-smart planning such as resilient infrastructure development, early-warning systems, and emergency response readiness. Deep learning predictions enable precise climate adaptation by linking model outputs to specific policy triggers and resilience measures. Below are expanded examples structured as **(Detection → Action)** with details on model mechanics, real-world applications, and adaptation outcomes drawn from established case studies.

Examples of Climate-Adaptive Applications Scenarios:

- **Flood Prediction Applications**
  - **(Detection)** CNN-LSTM models process satellite imagery and tide gauge data to forecast 48-hour coastal flood probabilities, achieving 20% higher accuracy than traditional models under sea-level rise scenarios. → **(Action)** Municipalities activate adaptive shoreline defenses like mangrove restoration and elevate critical infrastructure; early warnings reduce economic losses by 30-50% through prepositioned sandbags and traffic rerouting [5],
  - **(Detection)** Spatial-temporal LSTM networks integrate radar precipitation and river gauge data for real-time urban flood depth mapping with hourly resolution. → **(Action)** Emergency services deploy dynamic alert apps to residents, optimize pump station operations, and update urban zoning codes for flood-resilient building standards [6].
- **Wildfire Risk Management**
  - **(Detection)** Ensemble CatBoost-RNN models analyze vegetation indices, wind patterns, and temperature from MODIS satellite data to predict daily wildfire ignition probability across 10,000 km² regions. → **(Action)** Forestry agencies schedule controlled burns during low-risk windows, issue tiered evacuation orders via SMS, and allocate aerial water tankers preemptively, cutting suppression costs by 25%. [7],
  - **(Detection)** Hybrid CNN-BiLSTM frameworks simulate fire spread rates using VIIRS active fire detections and topographic data for near-real-time perimeter forecasting. → **(Action)** Communities install sensor networks for early smoke

detection, expand firebreaks with native vegetation, and train rapid-response volunteer brigades [8].

- **Drought and Heatwave Strategies**
  - **(Detection)** Hybrid QRNN+wavelet Deep Learning models forecast Streamflow Drought Index (SDI-3, SDI-6, SDI-12) for Stockholm, Sweden, integrating time series decomposition for multi-scale accuracy. → **(Action)** Water authorities implement 3–12 month advance restrictions, revise long-term supply contracts, and adjust hydropower operations to mitigate projected streamflow deficits [9],
  - **(Detection)** Deep and ensemble learning from European climate simulations isolate extreme heat and drought episodes amplifying heatwaves in Southern and Western Europe, emphasizing major climate indicators. → **(Action)** Cities like Lisbon and Athens activate "heat emergency plans": cooling centers, outdoor work bans, and targeted SMS alerts for elderly populations [10].

Case study: Heatwave prediction using Deep Neural Networks

Open-Meteo data source

For this case study, the **Open-Meteo platform** is used as a reference for weather data. Open-Meteo provides free access to historical weather data for any location worldwide through a publicly available API, allowing users to choose a time range and a set of weather variables.

To train a Deep Learning model capable of predicting heatwaves, we require specific variables that drive thermal stress. The download should be performed through a network interface, specifying the geographic location (latitude and longitude), time resolution (daily), and selected meteorological variables.

For this exercise, we will focus on key variable temperature_2m_max - this is the target variable we wish to predict to identify extreme heat events.

The data is imported directly into a Python environment using the Pandas library, which parses the **CSV** response from the API into a structured dataset suitable for Machine Learning.

Example of Deep Learning

In this practical session, we transition from simple statistics to modern Deep Learning. We will build a system that doesn't just say "it's hot today," but actively warns of approaching dangerous heatwaves by analyzing climate dynamics over time.

To achieve this, we use an **LSTM (Long Short-Term Memory)** network – a type of artificial brain capable of finding cause-and-effect relationships in time.

Step-by-Step Guide: How to Run the Simulation

We will use **Google Colab**, a free cloud platform that allows you to run this AI model directly in your browser without installing any software.

1. **Open Google Colab:**
   - Navigate to colab.research.google.com in your web browser,
   - You will see a "Welcome to Colaboratory" window.
2. **Upload the Notebook:**
   - Click on the **Upload** tab in the pop-up window,
   - Drag and drop the file "*ML_DL.ipynb*" into the box (or click "Choose File" to select it from your computer).
3. **Run the Code:**
   - Once the notebook loads, look at the top menu bar,
   - Click on **Runtime** and select **Run all** (or press Ctrl+F9),
   - *Note:* You may see a warning "Notebook requires high RAM" or "Author not trusted" – simply click **Run anyway**.
4. **Interact:**
   - Scroll down as the code executes. You will see data tables loading, training progress bars moving, and finally, interactive graphs appearing.

Technical Prerequisites

The implementation is performed in Python using the following specialized Python libraries:

- **TensorFlow/Keras**: For building and training the **LSTM** (Long Short-Term Memory) neural network,
- **Scikit-learn**: For data normalization (scaling) and calculating performance metrics (MAE, RMSE),
- **Pandas/NumPy**: For processing time-series data and performing numerical calculations,
- **Folium/Plotly/Seaborn**: For creating interactive forecast charts and geospatial decision support maps.

Data Source

We use the **Open-Meteo** platform as our "time machine." It provides historical weather data for any location on Earth. For training our AI, we selected key variable “Maximum Temperature (2 m)” - the target variable we want to predict.

Notebook Flow: Understanding the Program Code

The provided notebook “ML_DL.ipynb” is a complete, fully functional workflow that is ready-to-run. It is pre-configured to download data, train the neural network, and generate interactive reports automatically. No additional coding is required to see the results.

To adapt the simulation for different cities or scenarios, students only need to modify **Block 1 (Configuration & Libraries).** Blocks 2 through 8 **do not need to be modified** and contain the core logic (Data Processing, LSTM Training, Visualization). These blocks are automated and should simply be executed sequentially.

The practical work is divided into 8 logical blocks:

1. Configuration & Libraries. In this first step, we set up the "control panel" for our experiment. Before running the AI, students can modify specific variables to customize the simulation for different cities or scenarios:

- **Geographic Location:** LATITUDE & LONGITUDE: The coordinates of the target city (by default is Kyiv, Ukraine: 50.45, 30.52). Changing these allows you to test how the model performs in completely different climates, such as the rainy Šibenik (Croatia) or the arid Rome (Italy), Skopje (North Macedonia).
- **Time Machine Settings (Data Range)**: START_DATE & END_DATE. The historical period used to train the AI (e.g., "2010-01-01" to "2025-12-31"). Providing more years of history generally helps the model learn better patterns.
- **AI "Memory" Span:** LOOKBACK_DAYS. The size of the "sliding window" (default is **14 days**). This determines how far back the model looks to make a prediction.
- **Visualization Controls:** DISPLAY_START_DATE & DISPLAY_END_DATE. These control the zoom level of the main interactive graph (Step 6). Instead of plotting all 15 years, we can focus on a specific interesting year (e.g., 2025).
- **Safety Thresholds (Decision System):**
    - HEATWAVE_WARNING: The temperature threshold (e.g., **27.0°C**) that triggers an "Orange" advisory alert for the public.

- CRITICAL_EMERGENCY: The extreme temperature (e.g., 31.0°C) that triggers a "Red" critical alert for emergency services.

- **Forecast Analysis Period:** DISPLAY_FORECAST_START_DATE & DISPLAY_FORECAST_END_DATE. The specific range (e.g., August 2025) used in Step 7 to test if the warning system correctly identified heatwaves during a specific month.

2. Load Data. The code automatically connects to the Open-Meteo API and downloads weather history for the last 15 years. Students can observe how "raw" data is converted into a structured Pandas table (Figure 6.6).

```
Fetching data from Open-Meteo...
Data loaded: 5844 records.
```

	Date	MaxTemp	SolarRad
0	2010-01-01	-6.2	1.83
1	2010-01-02	-6.2	2.66
2	2010-01-03	-11.4	3.29
3	2010-01-04	-12.4	3.87
4	2010-01-05	-8.7	3.60

Figure 6.6 Fetched data from Open-Meteo

3. Preprocessing: The "Sliding Window". This is the key difference from classical algorithms. Instead of teaching the model based on *"Yesterday was warm -> Today will be warm,"* we create a **Sliding Window** (Lookback Period) of **14 days**. The model looks at the sequence (Day 1 ... Day 14) to predict Day 15. *Why:* This *allows* the neural network to perceive the *inertia* of the climate – how heat gradually accumulates before a peak.

4. Build and Train LSTM Model (Figure 6.7**).** We construct the "brain" of our system:

- We use an **LSTM layer**, which acts as the system's memory, retaining important trends while discarding random noise.
- A **Dropout** mechanism is added to prevent **overfitting** – a situation where the model simply memorizes past years but fails to understand general weather laws.

```
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0029 - mae: 0.0414 - val_loss: 0.0025 - val_mae: 0.0388
Epoch 38/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 0.0030 - mae: 0.0423 - val_loss: 0.0025 - val_mae: 0.0388
Epoch 39/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0029 - mae: 0.0413 - val_loss: 0.0025 - val_mae: 0.0389
Epoch 40/50
117/117 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.0030 - mae: 0.0423 - val_loss: 0.0025 - val_mae: 0.0389
Epoch 40: early stopping
Restoring model weights from the end of the best epoch: 35.
Training complete.
```

Figure 6.7 LSTM training with Early Stopping

5. Visualizing Training Performance. We plot the "Learning Curve".

The Goal: we want both lines to go down over time. This means the model's error (Loss) is decreasing.

- **Blue** Line **(Train Loss):** How well the model is learning from the "study material" (historical data),
- **Red Line (**Validation **Loss):** How well the model performs on "new questions" it hasn't seen before.

What to watch on Figure 6.8:

- **Good Result:** Both lines go down and stay close together.
- **Overfitting:** If the Blue line goes down but the Red line starts going **up**, the model is memorizing the data instead of learning patterns.

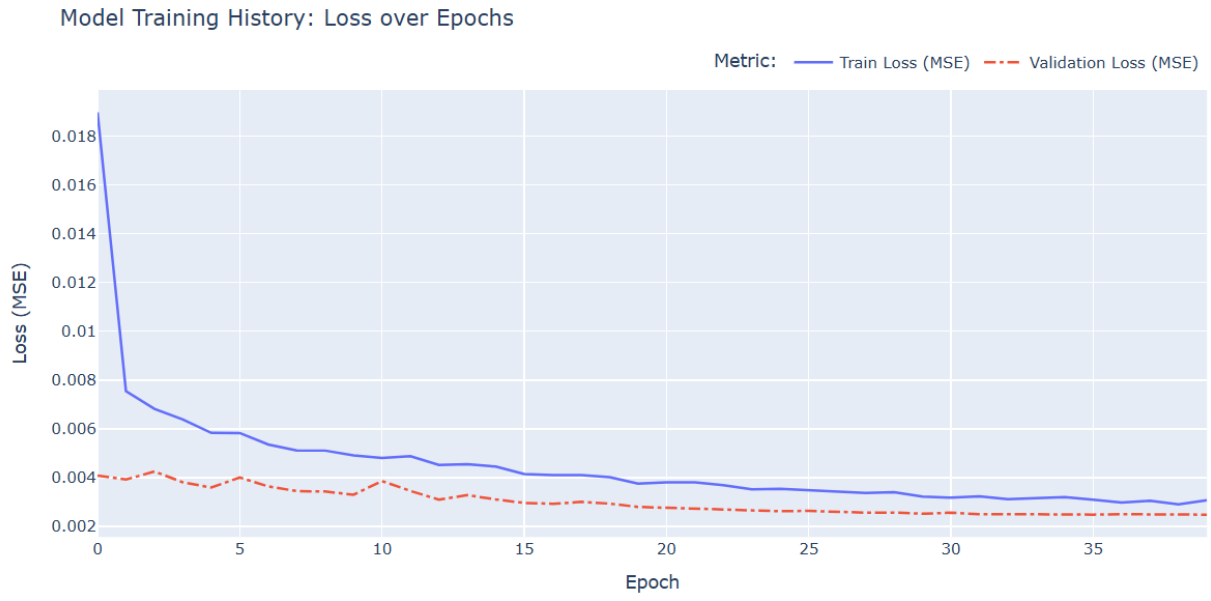

Figure 6.8 Model Training History: Loss over Epochs

6. Evaluation & Interactive Forecast. We use the **Plotly** library to generate an interactive chart.

6a. Model Report Card (Metrics):

Before looking at charts, we calculate the mathematical score of the model (Figure 6.9).

- **MAE (**Mean **Absolute Error):** The average "miss" in degrees (e.g., if the model is usually off by 1.5°C).
- **RMSE (Root Mean Squared Error):** A metric that punishes large mistakes more heavily than small ones.
- **R^2 (Accuracy Score):** A percentage score (0.0 to 1.0) indicating how much of the temperature variation the model successfully explains.

```
--- 📊 WEATHER MODEL REPORT CARD ---

1. Average Error (MAE): 2.14°C
   👉 Explanation: On average, the model's prediction is off by about 2.1°C.
--------------------------------------------------
2. Penalty for Big Mistakes (RMSE): 2.74°C
   👉 Explanation: This metric punishes large errors more heavily.
      If this number is much higher than the Average Error, the model occasionally makes big misses.
--------------------------------------------------
3. Accuracy Score (R²): 0.93 (or 93.1%)
   👉 Explanation: How much of the temperature variation does the model explain?
      (1.0 = Perfect, 0.0 = Useless, Negative = Worse than guessing)
   ✅ Quality: High Confidence.
--------------------------------------------------
```

Figure 6.9 Weather model report card

6b. Predictions (Interactive Forecast)

This generates a dynamic **Time-Series Plot** using Plotly (Figure 6.10).

- **Red** Line**:** The AI's predicted temperature,
- **Blue** Dashed **Line:** The actual recorded temperature,
- Interactive **Slider:** Located at the bottom, this allows students to zoom into specific weeks or months to see if the red line accurately follows the sudden spikes of a heatwave.

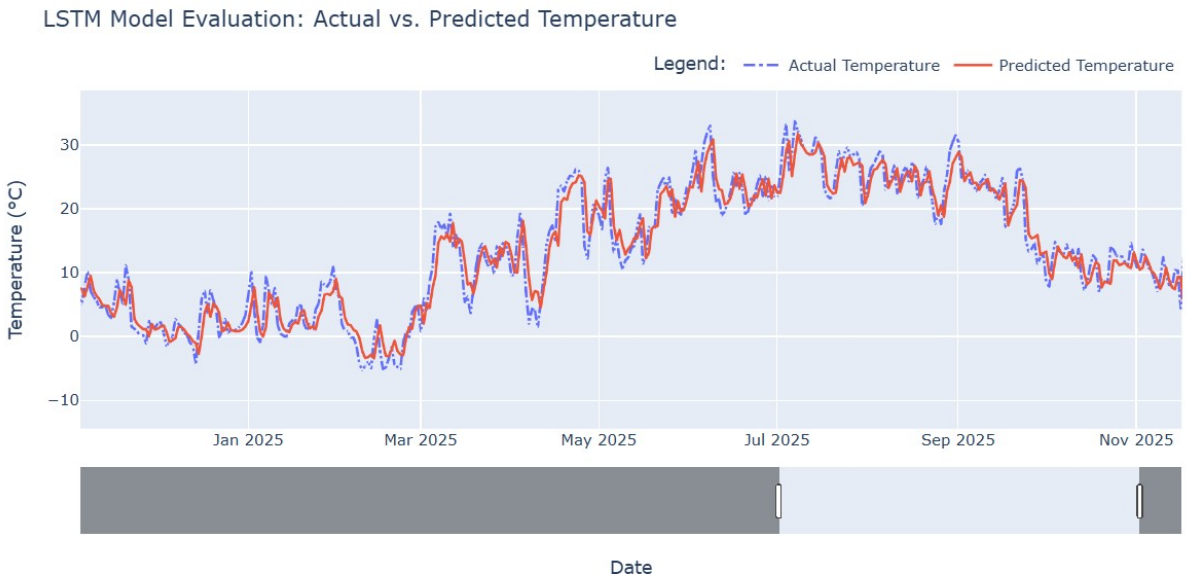

Figure 6.10 LSTM Model Evaluation: Actual versus Predicted Temperature

6c. Actual vs. Predicted (Goodness of Fit)

This generates a **Scatter Plot** to check for bias.

- **X-**Axis**:** Actual Temperature.
- **Y-Axis:** Predicted Temperature.
- **The "Line of Perfection":** A diagonal red dashed line. If the model were perfect, every blue dot would land exactly on this red line. This visualization helps us see if the model consistently overestimates (dots above the line) or underestimates (dots below the line) the heat.

6d. Residual Plot

This generates a plot showing the **Residuals** (or Errors) over time.

- **The Zero Line:** Represents a perfect prediction (0 error),
- **Purpose:** We look for randomness. If the dots look like random "noise" around the zero line, the model is healthy. If we see clear repeating wave patterns, it means the model is missing a key factor (like seasonality).

7. Climate-Adaptive Decision System

In this block, we model the operation of an **Early Warning System**. The Artificial Intelligence does not merely predict the temperature; it acts as a "digital advisor" for the city administration, signaling when safety protocols need to be activated.

The system operates on a traffic light principle, comparing the AI's forecast against the safety thresholds (THRESHOLDS) we defined in **Block 1**.

- 🟢 Level "Normal" (Green)
  - Condition: Predicted temperature is below the warning threshold (for example < 30°C).
  - Community Actions:
    - Routine monitoring of weather conditions,
    - Standard operation of municipal services,
    - *Explanation:* The situation is safe; no intervention is required.
- 🟠 Level "Advisory / Warning" (Orange)
  - Condition: Temperature crosses the first limit (for example ≥ 30°C, variable HEATWAVE_WARNING),
  - Community Actions:
    - Communication: Sending SMS alerts to vulnerable population groups (the elderly, people with chronic illnesses),
    - Healthcare: Placing ambulance services on high alert,
    - Social Services: Social workers conduct extra visits to isolated elderly residents.
- 🔴 Level "Critical Emergency" (Red)
  - Condition: Temperature reaches extreme values (for example ≥ 35°C, variable CRITICAL_EMERGENCY).
  - Community Actions:
    - Infrastructure: Opening municipal "cooling centers" – air-conditioned spaces in libraries, schools, or community centers open to the public,
    - Labor Safety: Restricting or banning heavy outdoor labor (construction, road repair) during peak sun hours,
    - Water Supply: Setting up mobile drinking water distribution stations in city squares,

- Energy Grid: Preparing for potential rolling blackouts due to grid overload from air conditioners.

```
--- HEATWAVE EARLY WARNING SYSTEM (LSTM) ---
Analyzing forecast for the period: 2025-08-01 to 2025-08-31
Date: 2025-08-01 | Forecast=22.2°C | Actual=24.3°C | Error=-2.1°C
  [ACTION]: 🟢 Status Normal.
------------------------------------------------------------
Date: 2025-08-02 | Forecast=24.7°C | Actual=26.4°C | Error=-1.7°C
  [ACTION]: 🟢 Status Normal.
------------------------------------------------------------
Date: 2025-08-03 | Forecast=26.1°C | Actual=27.0°C | Error=-0.9°C
  [ACTION]: 🟢 Status Normal.
------------------------------------------------------------
Date: 2025-08-04 | Forecast=26.3°C | Actual=28.1°C | Error=-1.8°C
  [ACTION]: 🟢 Status Normal.
------------------------------------------------------------
Date: 2025-08-05 | Forecast=27.1°C | Actual=29.0°C | Error=-1.9°C
  [ACTION]: 🟠 ADVISORY. Issue elderly warnings.
------------------------------------------------------------
Date: 2025-08-06 | Forecast=27.7°C | Actual=28.4°C | Error=-0.7°C
  [ACTION]: 🟠 ADVISORY. Issue elderly warnings.
------------------------------------------------------------
Date: 2025-08-07 | Forecast=27.0°C | Actual=23.0°C | Error=4.0°C
  [ACTION]: 🟠 ADVISORY. Issue elderly warnings.
------------------------------------------------------------
```

Figure 6.11 Heatwave Early Warning System

8. Geospatial Visualization

In this final block, we transform our temporal forecast into a **Geospatial Decision Support Tool**. Dots on the map change color (Blue/Orange/Red) depending on the predicted danger level. This demonstrates how city officials can monitor specific districts. Usually, static maps show a snapshot, but this animated map shows the *story*.

Community Application: If this model were expanded to a network of sensors (e.g., Downtown, Suburbs, Industrial Zone), city managers could see which specific districts are overheating (Urban Heat Island effect) and send water trucks *only* to those specific neighborhoods, saving resources.

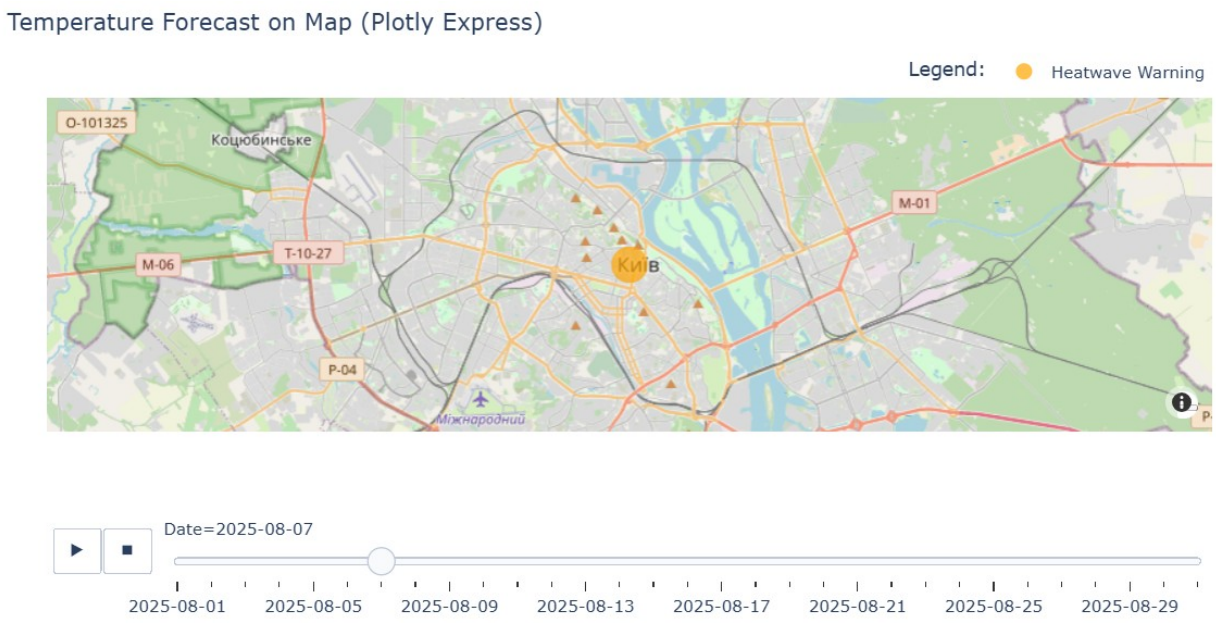

Figure 6.12 Map visualization based on CRITICAL_EMERGENCY and HEATWAVE_WARNING parameters

Exercise Tasks

In this exercise, you will use Deep Learning to predict extreme weather patterns in a city of your choice. Then, you will use these results to discuss practical solutions for protecting the city's infrastructure and population.

Task 1: Geographic Adaptation

- **Open** the notebook file provided in Google Colab.
- **Locate** the LATITUDE and LONGITUDE variables in **Block 1**.
- **Change** these coordinates to a city of your choice (e.g., your hometown, or a hot location like Rome, Athens, or e.t.c.).
- **Run** the notebook to generate new predictions.
- **Analyze the Decision System results in Block 7:**
  - Does the system output too many **"CRITICAL ALERT"** (Red) messages? (This suggests your thresholds are too low for this hot climate, potentially causing "alarm fatigue").

- Or are there no alerts at all despite hot weather? (This suggests your thresholds might be too high).

- **Map Visualization (Block 8):** Scroll down to the interactive map. Do you see RED or ORANGE dots? Does the visual distribution of "danger zones" match your expectations for this city?
- **Calibrate:** Go back to Block 1 and adjust the HEATWAVE_WARNING and CRITICAL_EMERGENCY variables to better suit your chosen city's climate.
- **Re-run the notebook.**

Question: How did changing the thresholds improve the usefulness of the Early Warning System for this specific location? What **urban planning policy** would you propose for that specific area to lower the heat risk over the next 5 years? (e.g., planting more trees, installing green roofs).

Task 2: Testing the "Memory" (Hyperparameter Tuning)

- **Locate** the LOOKBACK_DAYS = 14 variable in **Block 1**. This represents the AI's "memory span" (how many past days it analyzes).
- **Change** the value from 14 to 3 days.
- **Re-run** the entire notebook (Runtime -> Run all) to retrain the model with this "short-term memory."
- **Scroll down** to the **Evaluation Chart in Block 6b**.

Question: Compare the behavior of the Red Line (Prediction) now versus the previous 14-day run:

- **Lag Effect:** Does the red line shift to the right? (i.e., does the model only realize it's hot *after* the temperature has already risen?)
- **Peak Accuracy:** Does the model fail to reach the highest temperature peaks (underestimating the heat)?
- **Hypothesis:** Heatwaves don't happen instantly. Why does the AI need to see the last 14 days to predict them, instead of just the last 3 days?

Task 3 (⚠️ Advanced and Optional task): Scenario Flipping (The "Winter Anticyclone" Test)

Context: Climate change brings not just heatwaves, but also extreme cold snaps (e.g., polar vortex intrusions). Your current system is "blind" to cold because it only looks for high temperatures.

- **Step 1 (Time Travel):** In **Block 1**, change DISPLAY_FORECAST_START_DATE and DISPLAY_FORECAST_END_DATE to a known cold winter period (e.g., **"2012-01-15"** to **"2012-02-15"**). Change LATITUDE and LONGITUDE to a northern city known for harsh winters (e.g., **Helsinki**, **Oslo**, or **Winnipeg**). *Tip: You can use Google Maps to find the coordinates.*
- **Step 2 (Thresholds):** In **Block 1**, change CRITICAL_EMERGENCY to a freezing value, for example, **-15.0** (°C).
- **Step 3 (Logic Inversion):** Go to **Block 7** (Decision System).
    - Find the code line: if forecast_temp >= CRITICAL_EMERGENCY:
    - **Action:** Change the operator from >= (greater than) to <= (less than).
    - *Note:* Now the system triggers an alert only when the temperature *drops below* the freezing limit.
- **Run:** Execute Block 7.

Question: Does the system now correctly flag the coldest days as "CRITICAL ALERTS"? Discuss why adapting AI logic is just as important as having good data when facing different types of climate risks.

Task 4: Discussion

Interpret the results of Tasks 1 - 3 in an environmental context. Based on the frequency and distribution of alerts (Block 7, Block 8), propose a corresponding urban planning policy to mitigate the identified risks.

Assessment

What is the primary purpose of using an LSTM (Long Short-Term Memory) model instead of a simple average calculation for heatwave prediction?

A. To smooth out daily fluctuations and create a clean annual temperature chart.
B. To identify technical errors and noise in the raw sensor data.
C. To capture complex temporal patterns and the "accumulation effect" of heat over time.
D. To replace the need for physical weather stations in remote areas.

What is the most likely negative consequence of reducing the AI's "memory" (Lookback Days) from 14 days to just 3 days?

A. The model will become more sensitive to noise and produce frequent false alarms.
B. The model will suffer from a "lag effect," realizing a heatwave is happening only after the temperature has already peaked.
C. The model will automatically switch from Deep Learning to Linear Regression.
D. The model will fail to run because the dataset becomes too small for processing.

In environmental forecasting, what is the main risk of an "overfitted" Deep Learning model?

A. It produces predictions that are too generalized and ignore local weather specifics.
B. It memorizes specific historical events (noise) and fails to predict accurately when new, unseen weather patterns occur.
C. It requires significantly more computational power than the hardware allows.
D. It cannot process non-linear data such as humidity or wind speed.

From a city management perspective, what is the primary strategic advantage of using this Predictive AI System compared to a traditional "Reactive" approach?

A. It allows emergency resources (e.g., cooling centers, grid repairs) to be mobilized days before the crisis hits, reducing harm to the population.
B. It guarantees 100% accuracy, completely eliminating the possibility of false alarms or missed events.
C. It removes the need for human decision-making, allowing the city to be run entirely by computer algorithms without supervision.
D. It automatically lowers the physical temperature of the city by optimizing traffic lights.

Which phenomenon best explains why the AI model might predict significantly higher temperatures for a concrete-heavy city center compared to a nearby forest, even if they are geographically close?

A. The Urban Heat Island (UHI) effect, where artificial surfaces retain and re-radiate heat more than natural vegetation.

B. A localized atmospheric pressure anomaly that frequently forms over tall buildings.
C. A calibration bias in the satellite sensors that overestimates reflection from glass surfaces.
D. The "Lag Effect" caused by high traffic density delaying the airflow.

References / Further Reading

[1] F. Chollet, Deep Learning with Python, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2021 [Online]. Available: https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf, Accessed: Dec. 15, 2025.

[2] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. arXiv:2106.11342, 2021. [Online]. Available: https://arxiv.org/abs/2106.11342, Accessed: Dec. 15, 2025.

[3] Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016 [Online]. Available: http://www.deeplearningbook.org, Accessed: Dec. 15, 2025

[4] J. Heaton, *Applications of Deep Neural Networks*. arXiv:2009.05673, 2020. [Online]. Available: https://arxiv.org/abs/2009.05673, Accessed: Dec. 15, 2025.

[5] Bilal Hassan, Areg Karapetyan, Aaron Chung Hin Chow, and Samer Madanat. Climate Adaptation-Aware Flood Prediction for Coastal Cities Using Deep Learning. arXiv:2510.26017 [cs.CV], 2025 [Online]. Available: https://arxiv.org/abs/2510.26017, Accessed: Dec. 15, 2025

[6] Qianqian Zhou, Shuai Teng, Zuxiang Situ, Xiaoting Liao, Junman Feng, Gongfa Chen, Jianliang Zhang, and Zonglei Lu. A deep-learning-technique-based data-driven model for accurate and rapid flood predictions in temporal and spatial dimensions. Hydrol. Earth Syst. Sci., 27, 1791–1808, 2023 [Online]. Available: https://hess.copernicus.org/articles/27/1791/2023/hess-27-1791-2023.pdf , Accessed: Dec. 15, 2025

[7] M. Nasourinia and K. Passi, “Wildfire Prediction in British Columbia Using Machine Learning and Deep Learning Models: A Data-Driven Framework,” *Big Data and Cognitive Computing*, vol. 9, no. 11, p. 290, 2025. [Online]. Available: https://doi.org/10.3390/bdcc9110290, Accessed: Dec. 15, 2025.

[8] Hyunglok Seo and Yeonjoo Kim. Enhancing Alaskan wildfire prediction and carbon flux estimation: a two-stage deep learning approach within a process-based model. *Environ. Res.*

Lett., 19, 124040, 2024 [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-9326/ad8bdc, Accessed: Dec. 15, 2025

[9] Babak Mohammadi, Mohammed Abdallah, Rachid Oucheikh, Okan Mert Katipoğlu, and Majid Cheraghalizadeh. Enhancing streamflow drought prediction: integrating wavelet decomposition with deep learning and quantile regression neural network models. *Earth Sci. Inform.*, 18, 232, 2025 [Online]. Available: https://link.springer.com/article/10.1007/s12145-025-01736-w, Accessed: Dec. 15, 2025

[10] Elisa Felsche and Ralf Ludwig. Applying machine learning for drought prediction in a perfect model framework using data from a large ensemble of climate simulations. *Nat. Hazards Earth Syst. Sci.*, 21, 3679–3691, 2021 [Online]. Available: https://nhess.copernicus.org/articles/21/3679/2021/nhess-21-3679-2021.pdf, Accessed: Dec. 15, 2025