Introduction to Deep Learning: Introduction, Deep learning Model, Historical Trends in Deep Learning, Machine Learning Basics: Learning Algorithms, Supervised Learning Algorithms, Unsupervised Learning Algorithms.

1.1 Introduction to Deep Learning

1.1.1 The Motivation for Deep Learning

Since the early days of AI, researchers have tried to create machines that can think and learn like humans. Early AI systems were good at solving problems that could be explicitly defined, such as solving mathematical equations or playing chess. However, these systems struggled with tasks that are intuitive to humans, like recognizing faces or understanding natural language.

The fundamental problem was that early AI required humans to specify every rule. Tasks such as image recognition involve too much complexity for simple rule-based systems, as they require a deep understanding of the relationships between features. This is where deep learning comes in.

1.1.2 What is Deep Learning?

Deep learning allows computers to learn from experience by organizing data into a **hierarchy of concepts**. Each concept in this hierarchy is built upon simpler ones, leading to a network of many layers, hence the term "deep" learning. The main idea is to let the system automatically discover the representations needed for tasks, without manual input from humans on which features are important.

For example, in image recognition, instead of manually designing features to recognize edges, shapes, and objects, deep learning models learn these hierarchies from raw data. The model starts by detecting simple patterns (like edges), then combines them into more complex patterns (like shapes), and eventually builds a high-level understanding (like recognizing objects or faces).

1.1.3 The Role of Neural Networks in Deep Learning

Deep learning is often associated with **neural networks**, which are designed to mimic the way neurons in the human brain work. Neural networks consist of multiple layers of simple units (neurons) that pass information to each other. These networks can "learn" from data by adjusting the weights (importance) of the connections between neurons during training.

Each neuron in the network performs a simple mathematical operation. However, by stacking multiple layers of neurons, these networks can solve highly complex tasks. The deeper the network (i.e., the more layers it has), the more complex patterns it can learn to recognize.

1.1.4 Why "Deep" Learning?

The term **deep** refers to the number of layers in a neural network. Traditional neural networks, also known as shallow networks, have only one or two hidden layers between the input and output. Deep networks, on the other hand, can have many layers (sometimes hundreds), allowing them to model very complex relationships.

For example, in a deep learning model trained to recognize images:

- The first layer might detect basic features like edges.
- The second layer might combine these edges to detect simple shapes.
- The third layer might detect higher-level features like objects or faces.

The key advantage of deep learning is that it doesn't require humans to specify the features. The model learns the necessary features by itself through experience.

1.1.5 The Rise of Deep Learning

The concept of neural networks has been around since the 1940s, but deep learning became popular in the 2000s due to three major factors:

1. **Big Data**: The availability of large datasets has allowed deep learning models to train on more information, which leads to better performance.
2. **Computing Power**: Advances in hardware, especially the use of **Graphics Processing Units (GPUs)**, made it possible to train deep networks faster than before.
3. **Improved Algorithms**: Techniques such as backpropagation and better optimization algorithms enabled deeper networks to be trained more efficiently.

1.1.6 Applications of Deep Learning

Deep learning has had a significant impact on many industries and has led to breakthroughs in several areas:

- **Computer Vision**: Deep learning models can recognize objects, faces, and even emotions in images and videos. These models are widely used in applications like facial recognition and autonomous driving.
- **Natural Language Processing (NLP)**: Deep learning models are used to understand and generate human language. They power applications like machine translation (e.g., Google Translate) and speech recognition (e.g., Siri, Alexa).
- **Healthcare**: Deep learning helps in medical imaging for disease detection, personalized treatment plans, and even drug discovery.

1.1.7 Why Deep Learning is Important

Deep learning is important because it represents a major shift from the traditional rule-based approach in AI to one that can automatically learn from data. It has enabled machines to reach or exceed human-level performance in tasks such as image classification, speech recognition, and natural language understanding.

The success of deep learning is largely due to its ability to handle large amounts of data and automatically discover useful features. This has opened up possibilities for AI in areas that were once considered too complex for computers, leading to the rapid advancement of technology in various fields.

1.2 Historical Trends in Deep Learning

It is easiest to understand deep learning with some historical context. Rather than providing a detailed history of deep learning, we can identify a few key trends:

- **Deep learning has had a long and rich history**, but it has gone by many names reflecting different philosophical viewpoints. Throughout its history, deep learning's popularity has waxed and waned.
- **The availability of training data** has greatly increased, making deep learning more useful and effective. With more data, models can learn more complex patterns and perform better on a wide range of tasks.
- **Deep learning models have grown in size** as computer hardware and software infrastructure have improved, allowing for more layers and parameters in neural networks. This growth has enabled deeper networks to capture intricate data patterns more effectively.
- **Deep learning has been applied to increasingly complicated applications** over time, with continuous improvements in accuracy. These applications span various fields like image recognition, speech processing, natural language understanding, and beyond.

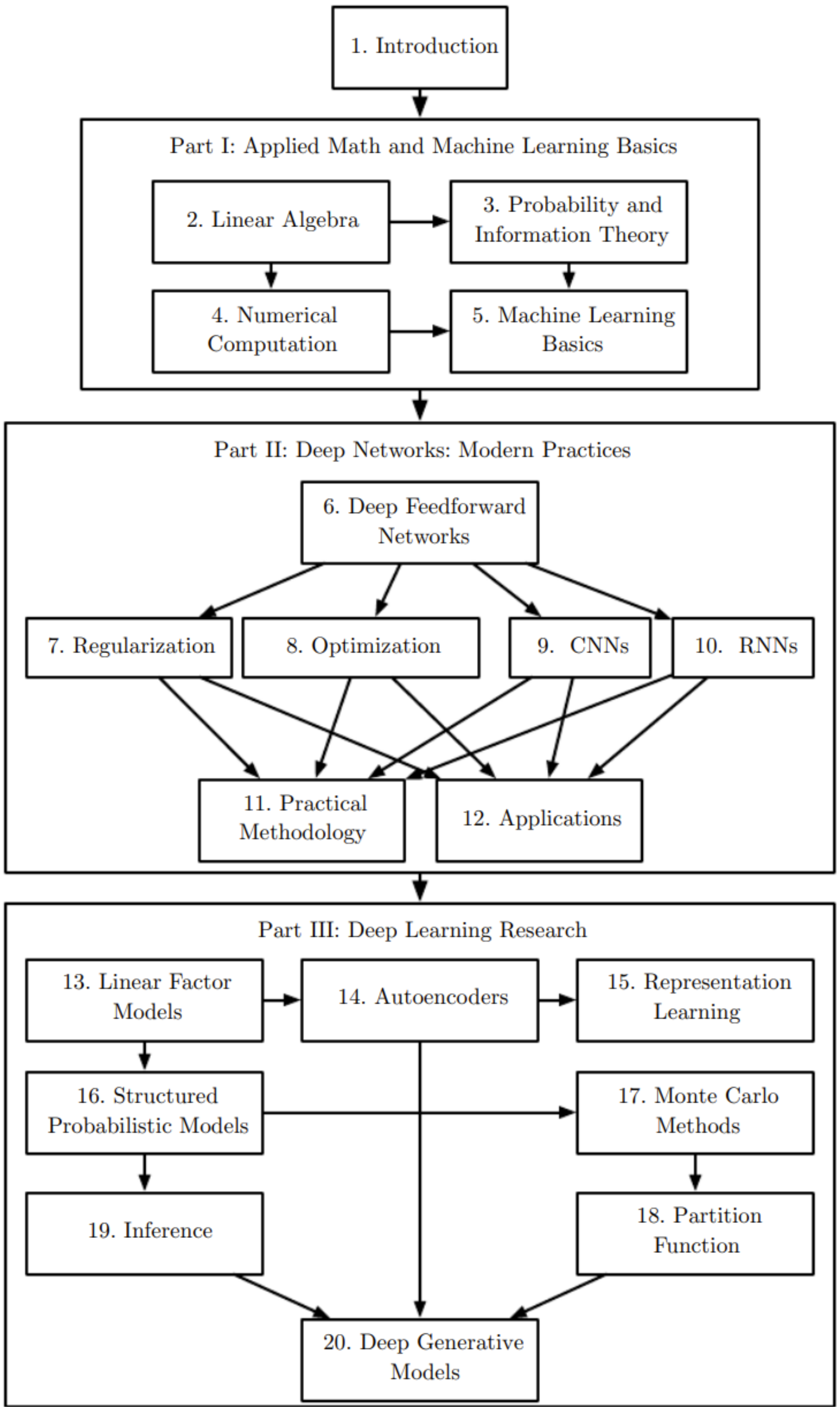
1. Introduction
Part I: Applied Math and Machine Learning Basics
2. Linear Algebra
3. Probability and Information Theory
4. Numerical Computation
5. Machine Learning Basics
Part II: Deep Networks: Modern Practices
6. Deep Feedforward Networks
7. Regularization
8. Optimization
9. CNNs
10. RNNs
11. Practical Methodology
12. Applications
Part III: Deep Learning Research
13. Linear Factor Models
14. Autoencoders
15. Representation Learning
16. Structured Probabilistic Models
17. Monte Carlo Methods
19. Inference
18. Partition Function
20. Deep Generative Models

Figure 1.6: The high-level organization of the book. An arrow from one chapter to another indicates that the former chapter is prerequisite material for understanding the latter

1.2.1 The Many Names and Changing Fortunes of Neural Networks

Deep learning, though now considered a cutting-edge technology, has actually existed for decades. It has evolved under different names:

- **Cybernetics (1940s–1960s)**: The earliest form of neural network research, cybernetics, was concerned with modeling how neurons in the brain might function. Early models like the **McCulloch-Pitts Neuron** were inspired by biological neurons and could recognize basic patterns using simple linear models.
- **Connectionism (1980s–1990s)**: During this period, the rise of **backpropagation** made it possible to train neural networks with hidden layers, marking a revival of interest in neural networks. This period is often referred to as connectionism or **parallel distributed processing (PDP)**.
- **Deep Learning (2006–present)**: The modern resurgence of neural networks, termed "deep learning," began in the mid-2000s. Advances in training techniques, such as **greedy layer-wise pretraining**, combined with improvements in hardware and data availability, enabled the training of much deeper networks, leading to a surge in the popularity of deep learning.

Despite its long history, deep learning only gained widespread recognition and success in the 21st century due to technological advancements that made training larger and deeper networks feasible.

1.2.2 The Importance of Data Availability

Deep learning's effectiveness relies heavily on the availability of vast amounts of data. The ability to train models on large datasets has been a major factor in its success:

- **Big Data**: As data availability increased, deep learning models could train on larger, more diverse datasets, leading to better performance and generalization. The rise of **big data** from sources such as the internet, social media, and sensor technologies has enabled models to learn from rich datasets.
- **Data-Driven Learning**: Deep learning models excel in learning directly from data, extracting features automatically rather than relying on human-engineered features. This ability makes large datasets particularly valuable for training deep models, as they allow the network to uncover complex patterns that would be difficult to design manually.

1.2.3 Advances in Hardware and Software Infrastructure

The resurgence of deep learning in the 21st century has been fueled by advances in hardware and software, which made it possible to train deep networks more efficiently:

- **GPUs (Graphics Processing Units)**: GPUs have played a crucial role in accelerating deep learning by enabling faster computation of the large matrix operations required by neural networks. Compared to traditional CPUs, GPUs can handle the parallel processing needed for training deep models in a fraction of the time.

- **Before GPUs**: Training deep networks was too slow on CPUs, which made it impractical to experiment with deeper architectures.
- **With GPUs**: The use of GPUs revolutionized deep learning, enabling models with many layers and parameters to be trained efficiently.

- **Software Libraries**: The development of libraries such as **TensorFlow**, **PyTorch**, and **Keras** has made implementing deep learning models much more accessible. These frameworks abstract much of the complexity involved in training neural networks, allowing researchers and developers to focus on designing architectures and experimenting with models.
  - **TensorFlow**: Developed by Google Brain, TensorFlow became one of the most popular deep learning frameworks due to its flexibility and scalability for building and training deep models.
  - **PyTorch**: Known for its ease of use and dynamic computational graphs, PyTorch is favored in research settings, allowing rapid experimentation with different architectures.

1.2.4 Increasingly Complex Applications

As deep learning models became more powerful and capable of capturing complex relationships in data, their applications expanded to more sophisticated tasks:

- **Early Applications**: Early successes of deep learning were in tasks like **image classification**, **speech recognition**, and **natural language processing** (NLP). For example, deep learning played a key role in achieving breakthroughs in **image recognition** during the 2012 ImageNet competition, where convolutional neural networks (CNNs) dramatically improved performance.
- **Recent Applications**: More recently, deep learning has been applied to even more complex and high-impact domains, such as:
  - **Autonomous Driving**: Deep learning models are used to detect objects, recognize lanes, and make real-time decisions in self-driving cars.
  - **Medical Diagnostics**: In healthcare, deep learning is used for tasks such as detecting diseases in medical images, predicting patient outcomes, and assisting in drug discovery.
  - **Natural Language Processing**: Models like **BERT** and **GPT** have achieved state-of-the-art results in various NLP tasks, including machine translation, text summarization, and question-answering systems.

1.2.5 Growing Model Size and Accuracy

Deep learning models have not only become deeper but also much larger in terms of the number of parameters and complexity:

- **Model Depth**: The depth of neural networks, which refers to the number of layers, has grown significantly. Deeper models can capture more abstract patterns in data, leading to higher accuracy on complex tasks. For example, **ResNet** (Residual Networks) allowed for the training of models with hundreds of layers, setting new standards for accuracy in tasks like image classification.
- **Model Parameters**: The number of parameters in deep learning models has exploded. Modern architectures like **transformers** (e.g., **GPT-3**, **BERT**) have billions of parameters, enabling them to tackle extremely complex tasks with remarkable accuracy.
  - **Transformers**: These models have revolutionized NLP by capturing intricate relationships in text data using attention mechanisms, leading to breakthroughs in tasks such as language translation and text generation.

5.1 Learning Algorithms

A machine learning algorithm is designed to learn from data. But what does it mean to learn? According to Mitchell (1997), "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." This definition highlights three critical components:

- **Experience (E)**: The data or information the algorithm learns from.
- **Tasks (T)**: The specific problems or functions the algorithm is attempting to solve.
- **Performance Measure (P)**: The metrics used to evaluate how well the algorithm performs its tasks.

The book does not aim to provide formal definitions for every potential experience, task, or performance measure, but it will offer intuitive descriptions and examples of various types of tasks, performance measures, and experiences in constructing machine learning algorithms.

5.1.1 The Task, T

Machine learning enables us to tackle tasks that are too complex for fixed, manually written programs. The ability to learn from data is what distinguishes machine learning from traditional programming, where rules are explicitly defined by humans.

In this context, the **task** is the objective that the algorithm is trying to accomplish. Importantly, the process of learning itself is not the task; rather, learning is the method by which the algorithm attains the ability to perform the task. For instance, if we want a robot to walk, walking is the task, and we could either program the robot with explicit instructions on how to walk or allow it to learn through experience.

Machine learning tasks are typically framed in terms of how the system processes an example. An example is a set of features quantitatively measured from an object or event that the system will process. We often represent an example as a vector x∈Rnx \in \mathbb{R}^nx∈Rn, where each entry xix_ixi of the vector corresponds to a specific feature.

Several common machine learning tasks include:

- **Classification**: In classification tasks, the algorithm is asked to determine which of kkk categories an input belongs to. This requires the learning algorithm to produce a function f:R^n→{1,…,k}. For example, in object recognition, the input might be an image (represented by pixel brightness values), and the output is a numeric code identifying the object. Modern deep learning techniques are particularly effective for classification tasks, enabling applications like facial recognition and automated tagging in photo collections.
- **Classification with Missing Inputs**: Classification becomes more complicated when some input features may be missing. The algorithm must learn to handle various subsets of inputs, which is common in fields like medical diagnosis, where tests may not always be available. The solution involves learning a probability distribution over the relevant variables, which allows for flexible handling of missing inputs.
- **Regression**: In regression tasks, the goal is to predict a continuous numerical value based on input data. The learning algorithm outputs a function $f : \mathbb{R}^n \to \mathbb{R}$. Examples include predicting insurance claims or future stock prices.
- **Transcription**: This task involves converting unstructured data into a structured textual format. Examples include optical character recognition (OCR), where an image of text is transcribed into machine-readable format, and speech recognition, where audio waveforms are translated into text. Deep learning is pivotal in improving the accuracy of these systems.
- **Machine Translation**: Here, the algorithm translates a sequence of symbols in one language into a sequence in another language, such as converting English text to French. Deep learning has significantly improved the accuracy and fluency of machine translation.
- **Structured Output**: Tasks that require the output to be a structured form, such as vectors or trees, fall under this category. Examples include parsing natural language sentences into grammatical trees or pixel-wise segmentation in images, where every pixel is assigned a class (e.g., road, sidewalk). In image captioning, the model generates descriptive sentences for images, showcasing how outputs are tightly inter-related.

- **Anomaly Detection**: In this task, the algorithm identifies unusual or atypical patterns within a dataset. For instance, credit card fraud detection models analyze spending behavior to flag transactions that deviate from established patterns.
- **Synthesis and Sampling**: The task involves generating new examples similar to those in the training data. Applications include automatic texture generation in video games and speech synthesis, where the model produces audio waveforms from text input.
- **Imputation of Missing Values**: The algorithm predicts missing entries in an input vector, which is common in many real-world datasets where information may be incomplete.
- **Denoising** is a type of machine learning task where the goal is to take an input that has been corrupted or altered in some way and restore it to its original, clean state. This process is particularly important in real-world applications where data often gets "noisy" due to various imperfections.
- **Density Estimation**: Here, the algorithm learns to estimate the probability density function of the input data, allowing for the capture of the underlying distribution of the data. This is critical for tasks such as missing value imputation or anomaly detection.

This section presents a broad overview of the various tasks that machine learning can handle, showcasing the versatility of machine learning algorithms in different applications.

5.1.2 The Performance Measure, P

To evaluate the effectiveness of a machine learning algorithm, a quantitative performance measure must be established. The performance measure PPP is tailored to the specific task TTT that the system is designed to perform.

- **Common Metrics**: For classification and transcription tasks, performance is often measured by accuracy, defined as the proportion of correct predictions made by the model. Alternatively, the error rate (the proportion of incorrect predictions) is frequently used, commonly referred to as expected 0-1 loss. The 0-1 loss function assigns a value of 0 for correct classifications and 1 for incorrect classifications.
- **Continuous Metrics**: For tasks like density estimation, accuracy and error rate are not suitable. Instead, the average log-probability assigned to a set of examples serves as a more appropriate performance metric, as it provides a continuous score reflecting the model's predictive capabilities.

When evaluating performance, it's crucial to assess how well the model generalizes to unseen data, as this determines its effectiveness in real-world applications. To achieve this, a separate test dataset, distinct from the training set, is used for evaluation.

Choosing the right performance measure can be challenging. Sometimes, it may be difficult to determine what should be measured. For instance, in transcription tasks, should we assess the accuracy of the entire sequence or allow for partial credit? In regression tasks, should we emphasize medium-sized errors or large ones? These decisions depend on the specific application context.

Moreover, in some situations, the desired performance measure may be impractical to compute. For example, in density estimation tasks, probabilistic models may represent distributions implicitly, making it difficult to compute the actual probability assigned to specific data points. In such cases, alternative criteria or approximations must be devised to ensure alignment with the design objectives.

5.1.3 The Experience, E

In machine learning, **experience (E)** refers to the data that the algorithm learns from during its training process. Based on the nature of the experience, machine learning algorithms can be broadly categorized as **supervised** or **unsupervised**.

Supervised Learning Experience

In **supervised learning**, the algorithm is provided with examples (data points), each associated with a label or target. The goal of the algorithm is to learn a function that maps the input data (features) to the correct output (labels or targets). For example, when working with the **Iris dataset** (a classic dataset where 150 iris plants are measured based on four features: sepal length, sepal width, petal length, and petal width), each plant is also labeled with its species. This is a supervised learning task because the algorithm uses both the features and the label (species) during training.

Supervised learning algorithms aim to estimate the probability distribution $p(y|x)$, where x represents the input features and y represents the label or target. The label y is considered the "supervision" or "instruction" given to the algorithm, hence the name "supervised learning."

Unsupervised Learning Experience

In **unsupervised learning**, the algorithm is given a dataset that contains many features, but no associated labels. The goal is to discover patterns, structures, or important properties within the dataset. For instance, in unsupervised learning, you may want to cluster the data into groups based on the similarities between examples, even though no labels are provided to indicate the correct grouping.

One of the main objectives in unsupervised learning is to learn the probability distribution $p(x)$, where x represents the features of the data. This could involve tasks such as **density estimation**, where the algorithm tries to understand how the data is distributed, or **clustering**, where the algorithm groups similar examples together.

For example, the Iris dataset could be processed using unsupervised learning to find natural groupings of plants based on their measurements, without knowing which species each plant belongs to.

Blurred Lines Between Supervised and Unsupervised Learning

While supervised and unsupervised learning are distinct categories, the line between them can sometimes blur. For example, the **chain rule of probability** allows for decomposing a joint distribution p(x) into a product of conditional distributions:

$$p(x) = \prod_{i=1}^{n} p(x_i | x_1, \ldots, x_{i-1})$$

This means that the seemingly unsupervised task of modeling p(x) can be viewed as solving multiple supervised learning problems. Conversely, supervised learning tasks, like learning p(y|x), can be approached by first learning the joint distribution p(x,y) and then deriving p(y|x).

Semi-Supervised Learning and Other Variants

There are other variants of learning that combine supervised and unsupervised approaches, such as:

- **Semi-supervised learning**: In this approach, some examples have labels, but many do not. The algorithm must use both labeled and unlabeled data to improve its performance.
- **Multi-instance learning**: A collection of examples is labeled as containing or not containing a particular class, but the individual examples in the collection are not explicitly labeled.

Reinforcement Learning and Interaction with the Environment

Some machine learning algorithms, such as **reinforcement learning**, do not experience a fixed dataset. Instead, they interact with an environment and learn from the feedback (rewards or punishments) they receive based on their actions. This creates a feedback loop between the algorithm and its environment, allowing the algorithm to continuously learn and adapt. Reinforcement learning is beyond the scope of this book, but is commonly used in applications such as robotics, gaming, and autonomous systems.

Representing Experience Using Datasets

In most machine learning algorithms, experience is represented as a dataset—a collection of examples. Each example is composed of features (quantitative measurements) that describe an object or event. There are two common ways to represent datasets:

1. **Design Matrix**: A design matrix is a matrix where each row represents an example, and each column represents a feature. For example, in the Iris dataset, the design matrix would have 150 rows (one for

each plant) and 4 columns (for the four measurements: sepal length, sepal width, petal length, and petal width).

$$X \in \mathbb{R}^{150 \times 4}$$

2. **Set Representation**: When the data points have variable sizes (such as images with different dimensions), it's not possible to represent the dataset using a fixed-size matrix. Instead, the dataset is represented as a set of examples, each with potentially different sizes.

In supervised learning, the dataset also includes a vector of labels y, where each yi represents the label corresponding to example xi. The learning algorithm uses this data to map the features to the correct labels.

5.7 Supervised Learning Algorithms

In supervised learning, the algorithm is trained to learn from examples that have both **inputs** (data) and **outputs** (answers). Think of it like a teacher showing a student both questions and answers during practice, so the student can learn to answer new questions later on.

- **Input**: This is the information or data the model gets, like an image of a flower.
- **Output**: This is the answer the model gives, like the name of the flower species (e.g., iris or daisy).

The goal of the algorithm is to learn a **function** that can give the correct output for new inputs it has never seen before.

5.7.1 Probabilistic Supervised Learning

One common approach in supervised learning is to **predict probabilities**. This means the model learns how likely each answer is, based on the data it sees.

- **Example**: If you show a model an image of a cat, it might predict that there's a 90% chance it's a cat and a 10% chance it's a dog.

Linear Regression (a simple supervised learning algorithm):

- In **linear regression**, the model tries to predict a continuous value (like a number). For example, it might predict the price of a house based on its size.
- The prediction is a straight line (hence "linear"). The model uses the formula: y=θTxy = \theta^T xy=θTx where xxx is the input (e.g., house size) and θ\thetaθ is a set of parameters (weights the model learns).

Logistic Regression (for classification):

- In **logistic regression**, the model tries to classify data into categories, like “yes” or “no” (e.g., is an email spam or not?).
- Since the output needs to be between 0 and 1 (like a probability), the model uses the **sigmoid function**: p(y=1∣x)=11+e−θTxp(y = 1 | x) = \frac{1}{1 + e^{-\theta^T x}}p(y=1∣x)=1+e−θTx1 This function "squashes" any number into the range of 0 to 1, which can be interpreted as a probability.

Finding the Best Parameters:

- For both linear and logistic regression, the model needs to learn the best **parameters** (like the weights θ\thetaθ) that help it make the most accurate predictions. For logistic regression, there isn’t a simple formula to find these weights, so we use a method called **gradient descent** to slowly adjust the weights until the model performs well.

5.7.2 Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are another type of supervised learning algorithm. Instead of predicting probabilities, SVMs make a clear **decision**: they separate data into two categories by drawing a line (or in higher dimensions, a boundary).

- **How SVMs Work**:
    - SVMs use a formula similar to logistic regression: wTx+bw^T x + bwTx+b.
    - But instead of giving a probability, SVMs directly classify an input. If the result is **positive**, the input belongs to one category (e.g., class 1), and if it’s **negative**, it belongs to the other (e.g., class 0).

The Kernel Trick:

SVMs are powerful because they can handle **non-linear data** (data that can't be separated by a straight line). This is where the **kernel trick** comes in:

- Normally, you would try to transform your data into a new space where it’s easier to separate. But the kernel trick does this transformation **without you needing to explicitly calculate it**.
- Instead of working with the raw data, the kernel trick calculates similarities between data points, allowing SVMs to work in much higher dimensions without extra computation.
- **Example**: Imagine you have two groups of dots that are shaped like two rings, one inside the other. A straight line can’t separate them. But by using a kernel, SVMs can handle this non-linear separation and find a boundary that works in a higher dimension.

Types of Kernels:

- **Gaussian Kernel (RBF Kernel)**: One common kernel is the **Gaussian kernel**, which calculates how close two data points are. The closer they are, the more influence one has on the other. It's like comparing the distance between two points and determining how similar they are. k(u,v)=e−||u−v||22σ2k(u, v) = e^{-\frac{||u - v||^2}{2 \sigma^2}}k(u,v)=e−2σ2||u−v||2 This kernel helps SVMs classify data in complicated, non-linear ways.

Why Use SVMs?

- **Advantages**: SVMs are great for finding the **best boundary** between two classes and can handle non-linear data using the kernel trick.
- **Disadvantages**: SVMs can be computationally expensive, especially when there's a lot of data, because they rely on calculations between every pair of data points (called **support vectors**).

5.7.3 Other Simple Supervised Learning Algorithms

There are several simple supervised learning algorithms that help in making predictions or classifications. Two common types of these algorithms are **k-nearest neighbors (k-NN)** and **decision trees**.

1. k-Nearest Neighbors (k-NN)

k-NN is like having a new object and deciding what it is based on its closest neighbors. Imagine you have a new fruit and you want to figure out if it's an apple or a pear. Instead of using some formula or calculation, you look at the fruits that are most similar to it and make your decision based on those.

How k-NN Works in More Detail:

- When you have a new **test point** (the fruit you're trying to classify), k-NN looks at the training examples (known fruits) to find which ones are most similar. The similarity is usually based on how "close" the points are in terms of distance. The closer the points, the more similar they are.
- **Example**: If you're using k-NN for classification and k=3, the algorithm looks at the 3 nearest neighbors. If two of them are apples and one is a pear, it classifies the new fruit as an apple, because that's the majority vote.

For **regression**, instead of voting for the most common class, k-NN takes the **average** of the nearest neighbors' values. For example, if you're predicting the price of a house, k-NN looks at the prices of the 3 nearest houses and predicts the price of the new house as the average of those.

Pros and Cons of k-NN:

- **Advantages**:
  - **No training needed**: k-NN doesn't require any training, meaning that you don't have to build a complex model beforehand. It's all about comparing the new data point to the stored training data.
  - **High capacity**: If you have a lot of data, k-NN can be very flexible and accurate.
- **Disadvantages**:
  - **Slow for large datasets**: Since k-NN has to compare the new point to every training example, it can become **slow** if you have a large dataset.
  - **Can't prioritize important features**: k-NN treats all features equally. So, if one feature is really important but others are just noise, k-NN might make wrong predictions. For example, if you're trying to predict house prices and only one feature (house size) is important, k-NN could get confused by irrelevant features like the color of the house.

---

2. Decision Trees

A **decision tree** works by asking a series of yes/no questions to make a decision. Think of it like a flowchart where you start at the top, ask questions, and follow the branches until you reach a conclusion.

How Decision Trees Work in More Detail:

- At each point (called a **node**) in the tree, the algorithm asks a **question** based on one of the input features. For example, if you're trying to classify animals, the first question might be, "Does it have feathers?".
- If the answer is "yes," it moves to the next node where it asks another question (e.g., "Can it fly?"). If the answer is "no," it asks a different question.
- Eventually, you reach a **leaf node**, which gives you the final decision (e.g., "It's a bird").

Decision trees are great because they're easy to understand, and you can visualize the process as a series of decisions.

Pros and Cons of Decision Trees:

- **Advantages**:
  - **Easy to interpret**: You can easily explain how the decision was made because you can follow the path of decisions.
  - **Handles different types of data**: Decision trees can handle both numerical and categorical data (e.g., "Yes/No" answers or actual numbers).
- **Disadvantages**:
  - **Struggles with complex boundaries**: If the decision boundary between classes isn't a straight line (e.g., it's curved or diagonal), the decision tree needs many nodes to approximate the boundary. This can lead to a very large, complicated tree, which makes it harder to interpret and less efficient.
  - **Overfitting**: Decision trees can easily overfit the training data, meaning they perform well on the training set but poorly on new data. This happens when the tree grows too complex and starts memorizing the data instead of learning general patterns.

Why Decision Trees Struggle with Some Problems:

- Imagine you're trying to classify data where class 1 occurs when x2>x1x_2 > x_1x2>x1 (like a diagonal line in a 2D space). Since decision trees split data along the axes (like "Is x1x_1x1 greater than 5?"), they will need many splits to approximate a diagonal boundary, which makes the tree complicated and less effective.

Comparing k-NN and Decision Trees

- **k-NN** is good for when you want a simple algorithm that doesn't need training but can become slow if there's a lot of data.
- **Decision trees** are easy to understand and interpret but can struggle with complex patterns and might overfit the data.

5.8 Unsupervised Learning Algorithms

Unlike **supervised learning** (where we give the algorithm labeled data—inputs with known outputs), in **unsupervised learning**, the algorithm only gets the **features** (input data) but no labels (correct answers). The algorithm's goal is to find patterns or structure in the data without any human-provided labels.

What Can Unsupervised Learning Do?

Unsupervised learning can be used for a variety of tasks:

- **Clustering**: Grouping similar data points together.
- **Denoising**: Cleaning noisy data to recover its original form.
- **Dimensionality Reduction**: Finding a simpler version of the data by reducing the number of features while keeping important information.

One major goal in unsupervised learning is to find the **best representation** of the data. A **representation** is a way of organizing or simplifying data so that it becomes easier to work with or understand.

Types of Representations:

1. **Low-Dimensional Representations**: Reducing the number of features while keeping as much important information as possible.
2. **Sparse Representations**: Using representations where most values are zero, which makes them easier to analyze.
3. **Independent Representations**: Finding features that don't depend on each other, making the representation simpler to interpret.

Now let's dive into some common **unsupervised learning algorithms**.

5.8.1 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a popular unsupervised learning algorithm used for **dimensionality reduction**. It finds a way to represent the data in fewer dimensions while preserving the most important information.

How PCA Works:

- Imagine you have a dataset with many features, but not all of them are important. For example, you have a 2D dataset where the data points mostly stretch along a diagonal line. PCA finds this line and creates a new **axis** along that direction, which captures most of the variance (spread) in the data.
- **Dimensionality Reduction**: PCA transforms the original high-dimensional data into a lower-dimensional space by keeping only the most important directions, known as **principal components**.

Why Use PCA?

- PCA is useful when you have too many features and want to simplify the data without losing important information. It makes the data easier to work with and speeds up computations.

Example:

Imagine you’re working with images, and each image has thousands of pixels (features). Instead of working with all the pixel values, PCA can reduce the dimensionality by finding patterns and compressing the data while retaining important information.

5.8.2 k-Means Clustering

k-Means Clustering is another popular unsupervised learning algorithm. It groups data points into **clusters** based on how similar they are to each other.

How k-Means Clustering Works:

- You start by choosing a number kkk, which represents the number of clusters you want.
- The algorithm assigns each data point to one of the kkk clusters by finding the **centroid** (center point) of each cluster.
- The algorithm repeats two steps until it converges:
    1. **Assignment Step**: Each data point is assigned to the nearest centroid (cluster center).
    2. **Update Step**: The centroids are updated based on the mean of the data points in each cluster.

At the end, you have kkk clusters, and each data point belongs to one of those clusters.

Example:

If you have a dataset of customer purchase histories, k-means can group customers with similar buying patterns into different clusters. For instance, one cluster might represent customers who buy tech products, while another might represent customers who buy groceries.

Strengths and Weaknesses of k-Means:

- **Strength**: k-Means is simple and easy to implement.
- **Weakness**: The number of clusters kkk needs to be defined ahead of time, and the results can vary depending on how the centroids are initialized. Also, it might struggle with finding more complex structures in the data (like overlapping clusters).

Limitations of Clustering:

Clustering isn’t always perfect because there are many ways to group data, and there may not be one "correct" answer. For example, if you’re clustering cars based on color and type, you might get clusters like "red cars" and "blue cars" or "trucks" and "sedans." The algorithm won't know which grouping is more useful for your task.

PCA is a mathematical way of **reducing the dimensions** of data. It helps simplify complex data by projecting it onto fewer dimensions while preserving as much variance (spread of data) as possible.

Key Concepts and Equations in PCA:

- **Variance** is a measure of how spread out the data is. PCA finds directions (called **principal components**) that capture the maximum variance in the data.
- **Design Matrix** X:
  - Let's say we have a dataset with m examples (rows) and n features (columns). The design matrix X is an $m \times n$ matrix where each row is a data point and each column is a feature

Step-by-Step Process of PCA:

1. **Center the Data**:
   - First, subtract the **mean** from each feature to center the data around zero:

$$E[x] = 0$$

2. **Covariance Matrix**:
   - The **covariance matrix** tells us how different features are related to each other (i.e., how much two features change together). The covariance matrix of X is given by:

$$\mathrm{Var}(x) = \frac{1}{m-1} X^T X$$

   Here, X^T is the **transpose** of the matrix X, and multiplying X^T by X gives us the covariance matrix.

3. **Find the Principal Components**:
   - To find the directions of maximum variance, we need to compute the **eigenvectors** of the covariance matrix. These eigenvectors form the basis for the new space, and the eigenvalues tell us how much variance each principal component captures.
   - The principal components of X are the **eigenvectors** W of the covariance matrix $X^T X$, and the eigenvalues form the diagonal matrix Λ:

$$X^T X = W \Lambda W^T$$

- **Singular Value Decomposition (SVD)**: Another way to find the principal components is through SVD, where we decompose X into three matrices:

  $$X = U\Sigma W^T$$

  Here:

  - U and W are orthogonal matrices containing the left and right singular vectors (similar to eigenvectors).
  - Σ is a diagonal matrix containing the singular values (similar to eigenvalues).

4. **Project Data into the New Space**:
   - Finally, we project the original data X onto the new principal component space Z:

     $$z = XW$$

     The result is a new representation of the data in fewer dimensions (using the top principal components that capture the most variance).

5. **Diagonal Covariance**:
   - PCA ensures that the covariance matrix of the new representation z is **diagonal**, meaning the components are uncorrelated:

     $$\mathrm{Var}(z) = \frac{1}{m-1}\Sigma^2$$

     This property is key to reducing redundancy in the data.

k-Means Clustering is used to group data points into k clusters based on their similarity. It works by assigning each data point to the nearest **centroid** (center of a cluster).

Key Concepts and Equations in k-Means:

1. **Centroids** $\mu(i)$:
   - Each cluster has a **centroid** $\mu(i)$, which is the mean of all data points assigned to that cluster.
2. **Objective**:
   - The goal of k-means is to minimize the total distance between the data points and their corresponding centroids. This is done by minimizing the **within-cluster sum of squares** (WCSS):

$$\sum_{i=1}^{k} \sum_{x \in \text{cluster}_i} ||x - \mu(i)||^2$$

   Here, $||x - \mu(i)||^2$ represents the squared distance between a data point x and the centroid $\mu(i)$.

3. **Algorithm Steps**:
   - **Initialization**: Choose k random points as the initial centroids.
   - **Assignment Step**: Assign each data point x to the nearest centroid by minimizing the Euclidean distance.
   - **Update Step**: Recompute the centroids by averaging the data points in each cluster:

$$\mu(i) = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

   Here, C_i is the set of points in cluster i, and $|C_i|$ is the number of points in that cluster.

4. **Repeat Until Convergence**:
   - The algorithm iterates between the assignment and update steps until the centroids no longer change significantly, meaning the clusters are stable.